Performance Evaluation of Configurable Hardware Features on the AMD-K5

Mike Clark and Lizy Kurian John*
Department of Electrical and Computer Engineering
University of Texas at Austin
ljohn@ece.utexas.edu

Abstract

Many modern processors incorporate certain configurable hardware features, although these features are never publicized. For instance, the AMD-K5 incorporates the ability to disable branch prediction, put caches into write allocate mode, etc. The ability to configure the features by software combined with the availability of on-chip performance counters allow the direct measurement of the performance of these features on real benchmarks running with real operating systems. This paper evaluates several configurable features that exist on the AMD-K5 processor using industry standard benchmarks. The features evaluated in this research are (i) cache predecode (ii) branch prediction, (iii) write allocate mode, and (iv) speculative loads bypassing stores. These features are evaluated on a benchmark suite that was relevant when K5 debuted (Winstone 96), as well as two subsequent benchmark suites (Winstone 97 and 98). This allows an evaluation of how the features have performed as software has progressed. The results show that write allocate was a big performance improvement on Winstone benchmarks, almost equal to the value of branch prediction. The bypassing loads feature did not return much in performance and the continuation of speculative predecode did not really help until Winstone 98. It was also interesting that all features showed positive improvement on Winstone 98. We feel that understanding the performance implication of these features and how they have matured over time gives us an increased insight into the future design of microprocessors and how application code is evolving.

1. Introduction

Most modern processors are equipped with features that can be enabled or disabled through software. There are usually two reasons to make the feature configurable. When a design trade off is evaluated, it might not be obvious which is the correct direction to take based on presilicon performance evaluation which is often approximate. Due to "time to market" issues, there might not be enough time to evaluate each feature and come up with a good choice before production of silicon. In addition, the feature must run through the full validation suite to make sure that it is functional. The risk of turning on a new feature to improve performance without full validation is a major concern. Hence, it is attractive if designers can make the future configurable and then it can be studied on the first silicon. Based on the first silicon results, the designers can then choose a default mode for production silicon. If the feature has a bug in first silicon, it can be disabled and validation can progress on the rest of the processor. In this paper, several hardware configurable features on the AMD-K5 processor are evaluated using industry standard benchmarks. Since the applications, the operating systems, and the compilers have all possibly changed since the designer's first evaluation of the feature, it is interesting to see how these features behave on newer benchmarks.

These configurable features are often not well publicized. Most processor companies do not want users to change these configurations. In addition, these features have been incorporated for use by the designer (and not users). However, some information on how to enable these features could be obtained from [1]. The absence of information on activating the features in newer processors prevented us from doing the studies on a processor newer than the K-5. However, we believe that this study yields valuable insight into a few interesting hardware features.

2. Description of Features

We chose four features to evaluate in this paper: allowing predecode to continue beyond unconditional branches, branch prediction, write allocate mode, and loads bypassing stores. To understand the **"continuation of predecode"** option, some background x86 knowledge

* Lizy John is supported by National Science Foundation under grant CCR-9796098 (CAREER Award), EIA-9807112, State of Texas Advanced Technology Program, Dell, IBM, Intel, AMD and Microsoft. Mike Clark is with AMD and the University of Texas. We express our gratitude to AMD for supporting this project.

is required. The uses a variable length instruction set. This makes it very hard to quickly find a group of instructions and put them together for simultaneous dispatch. Because of this problem, the K5 stores some extra information with each line of instructions called predecode [2]. This information marks critical bytes for the decoders, such as the beginning and end of instructions. Since this information is stored in an actual array, if the information is ever detected to be incorrect, a small penalty ensues while the offending information is removed. It turns out that incorrect information can get stored in this array for several legitimate reasons. One reason is inlined data. Since all data can be interpreted as instructions, inlined data can be speculatively predecoded as instructions and can pollute the predecode cache. For example, assume that a five-byte instruction exists at offset zero, which is a branch around the inlined data. The inlined data starts at offset five and is four bytes long and the following instruction is at offset nine and is five bytes long. Speculatively predecoding the inlined data could produce a five-byte instruction. This would cause the instruction that should start at offset nine to be ignored and a new instruction created starting at offset ten.

Original Code
Speculative Predecode

0:	**start: jmp +00000004**
0:	*start: jmp +00000004*
5:	**my_data: e9 01 02 03**
5:	*my_data: jmp +01020389*
9:	**mov eax, ecx**
10:	*enter*

This predecode could be speculatively written into the predecode array. Once the branch redirects the processor to the real destination of the branch, offset nine, the predecode array will have to be corrected. This is because the predecode array says offset nine is the end of an instruction not the beginning of an instruction. Another cause of bad predecode is the mixing of 16-bit and 32-bit segments. If 16-bit and 32-bit segments are adjacent, speculative predecoding into the other segment will cause bad predecode because the lengths of instructions are different when interpreted in 16-bit mode versus 32-bit mode. Because of these two reasons, the default mode on the K5 processor is to stop predecoding when an unconditional branch is decoded. However, when the processor stops predecoding, it waits for the branch to resolve. This wasted time could be used to effectively predecode speculative instructions, assuming the two cases stated earlier do not exist. The operating system used for the evaluation in this paper was Windows 95, which is a 32-bit operating system capable of running both 16-bit and 32-bit applications. Since Winstone 96 is comprised of 16-bit applications, we would expect a lot of 16 and 32 bit segment problems with speculative predecode. Since Winstone 97 and 98 are supposed to be all 32-bit applications, we expect fewer of these problems and a greater performance boost from speculative predecoding in these benchmarks.

K5's **branch prediction** uses a dynamic scheme that integrates branch prediction with the instruction cache. This approach of branch prediction is described in Johnson [7]. The K-5 scheme allocates one byte of branch prediction information per 16 bytes. The instruction cache is 16K bytes in size, which means the K5 has 1K storage for branch prediction information. It should be noted that the prediction information is essentially a component of the cache line. If the cache line gets replaced due to another cache miss, the branch prediction for the line getting displaced is also lost. When the K5 processor is run with branch prediction off, it predicts every branch it encounters as not taken.

Write allocate mode is a cache-based feature that allows stores that miss in the cache to do a read for the line that missed. Once the line is in the cache, the store then writes the data into the cache. When write allocate mode is off, store misses just enter the store buffer as non-cacheable and are sent externally to the bus. This is very inefficient if following the store there exist several stores and loads to the same cache line. Since the load will bring the line into the cache anyway, doing several non-cacheable stores to the bus is slow and inefficient. However, there are some disadvantages to write allocate mode. If the program has write once variables, reading the whole line into the cache and displacing another line incurs a performance loss. Also, if large arrays are being initialized but not immediately used, the stores to initialize the array will displace all other useful data. One other negative to write allocate mode is that it can only be used for actual memory. If the operating system sets up memory mapped I/O for some range of addresses, writes to those addresses cannot be allowed to do reads to the bus first, because an actual I/O device may be at that address. Because of this, the K5 processor implements both a write allocate enable bit and two registers for specifying the regions of the address space that are allowed to be used in write allocate mode. Any store to an address not within this range behaves as if the write allocate feature is turned off.

The last feature investigated in this paper is **speculative loads bypassing stores**. The K5 micro-architecture has a four-entry load-store queue. All loads and stores first enter this queue upon dispatch and calculate their linear address. From there they access the cache and return their data. In the case of stores, the K5 has a four entry store buffer. This buffer contains stores that have resolved their address but have not updated the cache or been sent to the bus yet. The store buffer is to allow load hits to continue while stores are still outstanding. The K5 also has a blocking cache. This means that while a cache miss is being processed, no other load or store can access the cache. In default mode, the K5 does not allow load misses in the cache to bypass the store buffer. The load miss waits until all store buffer entries have emptied before issuing its request to the bus, thereby insuring that it is non-speculative. By enabling this feature bit, load misses no longer wait for the store buffer to drain and they speculatively make a request for the data. This helps to improve performance by getting in advance the data that the program needs on a read operation. The downside of this feature is that speculative loads can displace frequently used data out of the cache and the load data brought in may be speculative and never used. This can also increase bus traffic and cause subsequent non-speculative loads and stores to stall, waiting for the bus to be free.

3. Methodology

The Winstone suite of benchmarks was chosen to measure how the different features actually effect performance. Since the main market focus of the K5 was the Windows compatible market, its features should naturally be tested by benchmarks that closely model that type of workload. The Winstone 96 benchmark was the current benchmark when the AMD K5 was introduced. Therefore, if an evaluation was done by the original designers, it would have been done using this benchmark as well as others. We also chose the Winstone 97 and Winstone 98 benchmarks to track how the features have performed over time.

Table 1. Description of the Benchmarks

Type	Winstone 96	Winstone 97	Winstone 98
Spread Sheet	Lotus 123 5.0		Lotus 123 97
	Excel 5.0c	Excel 7.0	Excel 97
	Quattro Pro 6.01		Quatro Pro 7
Word Processing	Lotus Ami Pro 3.1		
	Word 6.0c	Word 7.0	Word 97
	Wordperfect 6.1		Wordperfect 7.0
		Wordpro 96	
Graphics		Corel Draw 6.0	Corel Draw 7
		Page Maker 6.0	
		Power Point 7.0	Power Point 97

The Winstone suites are a collection of the most popular business applications of the time. Usage studies are performed and scripts that mimic the normal user tasks that are performed with the applications are developed. The software times how long it takes each application script to execute and then reports a single, relative score. The relative score is relative to the time it took the script to run on a baseline machine [3]. This baseline machine is not the same for the different versions and therefore the numbers cannot be compared across versions. Here is the list of applications for the three benchmarks that we used (Table 1). Table 1 shows how the applications continue to evolve from Winstone 96 through Winstone 98. Installing and running the Winstone software can be complex and it took a significant amount of time to get Winstone 97 and Winstone 98 to run to completion.

The software has a mode where sub-suites of the applications can be run. We encountered difficulties in running Winstone 96 suite to completion but we were able to run the Word Processing and Spreadsheet sub-suites. One application, Works 3.0b, was also included in both runs of the Winstone 96 sub-suites. The Winstone 97 benchmark splits itself into two categories, business and high end. The business section is similar to the Winstone 96 suite, except with newer applications. The high-end portion is a new class of applications that represent a more demanding group of software focusing on application development, image editing, and CAD/3-D software. Winstone 98 has the same split of business and high-end applications. However, the high-end portion of Winstone 98 only runs under Windows NT. Winstone 98 also added a new concept called task switching. Previously, Winstone opened the application, timed and ran the scripts and closed the application. With the task switching tests, the scripts open several applications at once and switch back and forth, further modeling a normal user workload. As table 1 shows, there are no identical versions of applications across the three benchmarks. However, there is a natural progression from 16-bit applications in Winstone 96 to 32-bit applications in Winstone 98.

To run the benchmarks, we used a standard PC equipped with the following features:

- an AMD-K5 Pr166 (CPUID 524)
- 48 MB of memory
- 256 K of front side bus L2 cache
- PCI-VGA compatible Display adapter
- Display mode 1024 X 768 8 bits/pixel
- Windows 95 Build 950.
- 66 MHZ system bus

The tools we used to turn on the features were the MultiProbe utility and the K5 performance monitor tools obtained from AMD. These tools are Windows based applications that allow user level access to the supervisor level registers that are necessary to turn on the features. K5 has only one performance monitor register so it is not very useful for measuring many interactions at a time. It was useful in more of a validation role. We could enable or disable the hardware feature and then make sure it was having the expected effect. For example, we could turn the branch prediction bit off and then use the monitors to confirm that when running a small benchmark, we got a larger number of branch mispredictions. We also needed the performance monitor because the "continuing speculative predecode" feature is enabled through the performance monitor.

4. Results

Every benchmark was run twice with each individual feature enabled. The runs were then averaged the two runs to produce a score for that feature on that benchmark. The scores of the two runs were observed to be very consistent. The runs averaged from 20 minutes for the Winstone 96 runs to one hour for the Winstone 98. Within an individual benchmark, a larger score means a performance improvement and a smaller score a performance degradation. Figure 1 shows the actual average Winstone scores for all the features.

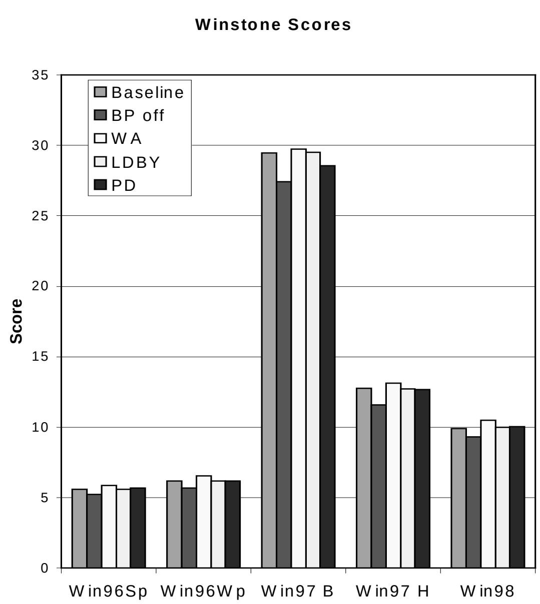

Figure 1. Average Winstone score with each individual feature.

The abbreviations are interpreted as follows: Win96 SP is the Winstone 96 spreadsheet benchmark, Win96 Wp is the Winstone 96 Word processing benchmark, Win97 B is the Winstone 97 Business benchmark, Win97 H is the Winstone 97 High End benchmark, and the Win98 is the Winstone 98 benchmark. The feature abbreviations are: LDBY is loads bypassing stores, PD is continuing predecode, WA is write allocate, and BP off is branch prediction off. The value of the scores between the associated benchmarks is irrelevant. The important thing is the increase or decrease of the score within each benchmark based on the feature being enabled or disabled.

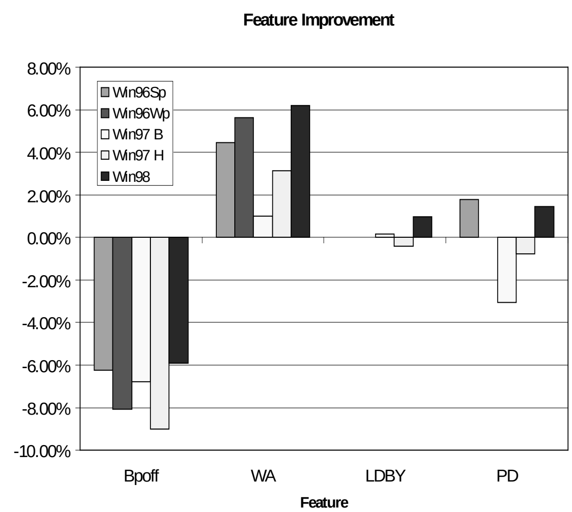

Figure 2. Percent Change in Performance for each individual feature, in various Winstone suites.

Figure 2 illustrates the relative performance improvement of each individual feature. The scores are compared relative to scores obtained from the baseline configuration with write allocate off, speculative predecode off, load bypassing off and branch prediction on. Branch prediction on the K5 ranged from an increase as high as 9% on High End Winstone 97 to an increase of only 6% on Winstone 98. The K5 has a relatively short super-scalar pipeline of five stages. This reduces the penalty of mispredicting to three clocks. We still thought branch prediction would be a much higher performance lever than 6-9%. Also, a 3-5% improvement on Winstone benchmarks is typically the performance improvement obtained by one higher speed grade of a processor. It would be hard to imagine any of the other features exceeding the performance improvement of branch prediction, since branches are very frequent in the applications we studied. However, write allocate came close.

The write-allocate feature resulted in a performance increase of approximately 1% on Winstone 97 Business to about 6.25% on Winstone 98. Since the x86 architecture has a limited integer register set of eight 32-bit registers, there are frequent register spills of data to memory. The K5 has a one cycle L1 cache access, which is in contrast to the memory system which typically would be in the range of 6 cycles for a L2 hit and 18 cycles for a L2 miss. Because of all this, it is no surprise the write allocate is a big performance improvement. If the first instance of an address is a store, there is a high probability that there are more loads and stores to the same address and to the same cache line. The performance improvement on Unix based applications should be even greater, based on the information from Bekerman and Mendelson [4]. Their performance evaluation showed Unix applications to have better spatial and temporal locality for their memory references than Windows applications. The only 1% improvement on Winstone 97 was less than expected. However, the 6.25% improvement on Winstone 98 is encouraging. The task switching feature of Winstone 98 increases the chance of register spills and causes context switch information to be continually stored to memory. Write allocate is definitely the highest performing feature that was tested in this research but the most complicated to configure. Since the write-allocate memory ranges have to be programmed according to the individual computer's memory map, the feature cannot be enabled by the processor without PC BIOS support or operating system support.

Loads bypassing stores as implemented in K-5 did not turn out to be a very performance enhancing feature. It showed no improvement on the Winstone 96 benchmarks. It provided a small improvement on the business portion of Winstone 97 and a small performance degradation on high end Winstone 97. It was also a small improvement on Winstone 98. Conceptually, this feature is worth more. However, the bottleneck here seems to be the size of the store buffer and the blocking cache. Since the store buffer is only four entries deep, this does not allow for significant buffering relative to speculative loads. In addition, with speculative loads bypassing the stores in the store buffer, a stall is caused when the store buffer has become full because loads are blocking the cache. If the store buffer size was enlarged, or if the cache was non-blocking, we expect that this feature would have been more useful.

The predecode feature produced the most mixed results. The Winstone 96 runs showed a decent improvement on the Winstone 96 spreadsheet but no improvement on the word processor run. The Winstone 97 business shows a 3% degradation of performance and the high-end run showed a .75% degradation. Finally, Winstone 98 loves the feature with a 1.5% improvement. We certainly expected this feature to improve performance over time. As software moves more uniformly to 32-bit code and with better compilers generating less in-lined data, this feature should become a performance improvement. Our explanation is as follows. The Winstone 96 runs were all 16-bit applications and therefore were able to show a modest improvement, although the underlying operating system was 32-bit code. Winstone 97 is supposed to be all 32-bit applications. However, this means the majority of their code is 32-bit but they are not 100% 32-bit code. This would explain the decrease in performance the predecode feature suffered on Winstone 97. Then, Winstone 98 is truly 100% 32-bit applications running under a 32-bit OS and is able to speculatively predecode without many penalties. One unfortunate thing about the predecode feature was that it was enabled through the performance monitor counter. The penalty caused by speculatively predecoding could have been monitored except that there is only one performance monitoring counter on K5. It is not possible to turn on the feature and count the penalty events at the same time.

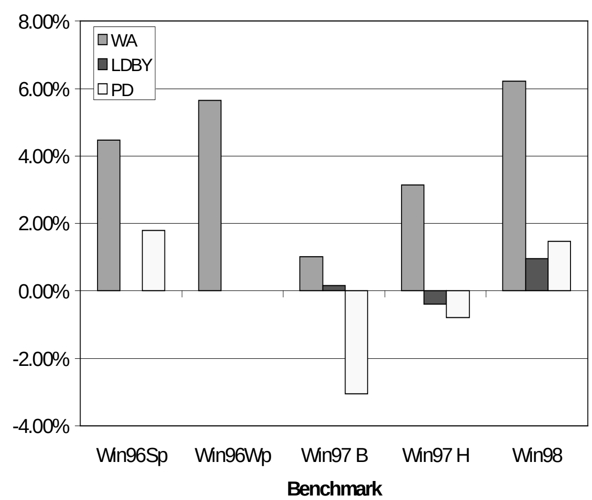

Figure 3. Performance Improvement of Write Allocate, Load Bypassing and Predecode features.

Figure 3 illustrates how the features progressed over time. In this graph, the performance improvement of individual features except branch prediction is illustrated. The graph shows that with the exception of write-allocate, none of the features showed a significant increase for Winstone 96. This may explain why these bits have the default state that they do. Of course, write allocate cannot be the default state of the processor since it needs to be enabled by software that is aware of the memory configuration of the machine. It is not clear why the normal bios and OS do not enable the write allocate feature when they detect a K5 processor. Perhaps this feature was incorporated in later BIOS's but it was not in the specific system used for our study. For Winstone 97, the results were even worse with the features, excluding write-allocate, actually causing performance degradation. However, by the time we reach Winstone 98, all the features have finally started having significant positive effects on performance. We believe that this is because the applications are now fully 32-bit and are using better compilers that are generating better code to utilize these features.

5. Related Research

Papworth analyzed design trade-offs for the Pentium Pro microarchitecture [5]. However, there were major differences with our approach. First, Papworth's analysis was done pre-silicon on a trace driven simulator. Second, the analysis was concerned with speed, die size and schedule as well as overall performance. Still the paper is interesting both as a comparison and as a contrast to this approach. It is interesting to note that perusing through Intel manuals, several hardware configurable bits do exist, including a streaming buffer disable bit, a disable lock assertion for split locked access bit and a fast string enable bit [6]. Papworth's paper did not refer to these bits nor could we find a paper quantifying their performance.

Bekerman and Mendelson [4] used the Pentium performance monitoring counters and some software monitoring to measure how the Pentium was performing on some Unix and Windows benchmarks. This is slightly different from the approach in this paper because they were not changing features inside the processor but merely measuring how the standard mode of the processor was performing. They focus their attention on the CPI of the benchmark, instead of the true execution time of the benchmark. One interesting observation made by this paper related to the differences between Windows applications and Unix applications. Windows applications are reported to show lower temporal locality and spatial locality, a higher memory reference rate, a higher instance of non-cacheable references, less predictability, and a higher context switch rate. This helped validate the belief that Winstone is the correct benchmark for evaluating the K5.

6. Conclusion

Although not well-publicized, many microprocessors incorporate several configurable hardware features. Studying such configurable hardware features gives valuable insight into how a feature will behave on real benchmarks. Of course the evaluation is limited by the configurability built into the hardware itself, which is usually the obligatory on or off. There is no capability to change buffer sizes and other hardware constraints that may be limiting the performance effect of the feature. Within these limitations, this study shows that these features in a small increase or even a decrease in the benchmarks at the time the processor was built. However, as the software has evolved over time, the features now create significant performance improvement on benchmarks of today. Unfortunately for the K5, its micro-architecture was not able to reach the clocks speeds necessary to make it a viable option on the benchmarks of 1998, when these features could really be unleashed. However, designers of modern microprocessors can use this data to guide their decisions for the future. We feel that understanding the performance implication of these features and how they have matured over time gives us a great insight into the future design of microprocessors and how application code is evolving. This paper also points to the need of providing more configurable hardware features. As processor microarchitectures are getting increasingly complex and accurate presilicon performance evaluation is becoming a challenge, the ability to evaluate and choose features at silicon level is becoming increasingly important.

7.References

[1] http://www.sandpile.org/

[2] AMD5k86 Processor Technical Reference Manual, Advanced Micro Devices, Sunnyvale, Cal., 1996.

[3] Understanding and Using Winstone 97, Ziff Davis Publishing Company, 1996.

[4] Michael Bekerman, Avi Mendelson, "A Performance Analysis of Pentium Processor Systems", IEEE Micro, October 1995, Vol. 15, No. 5, pg. 72.

[5] David B. Papworth, "Tuning the Pentium Pro Micro-architecture", IEEE Micro, April 1996, Vol. 16, No. 2, pg. 8.

[6] Intel Architecture Software Developer's Manual Volume 3: System Programming Guide, Intel Corp., Denver, CO., 1997.

[7] M. Johnson, *Superscalar Microprocessor Design*, Prentice Hall, 1990, ISBN 0-13-875634-1.