**PowerPC 620 Micro architecture**

**Abstract**

The PowerPC 620 superscalar microprocessor is the most recent and performance leading member of the PowerPC family, which is being jointly developed by IBM and Motorola. The 64-bit 620 represents the most aggressive micro architecture for superscalar processors to date. It employs a two-level branch prediction scheme, dynamic renaming for all the register files, distributed multi-entry reservation stations, true out-of-order execution by six pipelined execution units, and a completion buffer for ensuring precise interrupts.

A performance simulator for the 620 is developed using the VMW (Visualization-based Micro architecture Workbench) remerge table framework. The VMW-based simulator accurately models of a 620 micro architecture down to the machine cycle level. Extensive trace-driven simulation is performed using the SPEC9b2 benchmarks he experimental results indicate that the 620 is a well balanced design and achieves a maximum IPC rating of 1.94 on one of the benchmarks.

Detailed quantitative analyses of the effectiveness of all the key micro architecture features are presented. A brief philosophical comparison with the Alpha AXP 21164 is also including.

**Introduction**

The latest announcement by the IBM-Motorola-Apple alliance is the PowerPC 620, the first 64-bit member and performance leader of the PowerPC family. While ~he latest Alpha AXP 21164 {8] (@ 300MHz) from DEC may edge out the 620 (@ 150MHz) as the industry performance leader~ the 620 employs the most aggressive micro architecture and achieves the highest level of instruction-level parallelism of any microprocessor currently on the market. The 620 is the first 64-hit superscalar microprocessor to employ true out-of-order execution, aggressive branch prediction, distributed multi-entry reservation stations, dynamic renaming for all register files~ six pipelined execution units, and a completion buffer to ensure precise interrupts. Most of these features have not been previously implemented in a single-chip microprocessor.

Their actual effectiveness is of great interest to both academic researchers as well as industry designers. This paper presents an instruction-level, or machine-cycle level, performance evaluation of the PowerPC 620 micro architecture.

**The PowerPC 620 Architecture**

The PowerPC architecture is the result of the PowerPC alliance among Apple, IBM, and Motorola. It

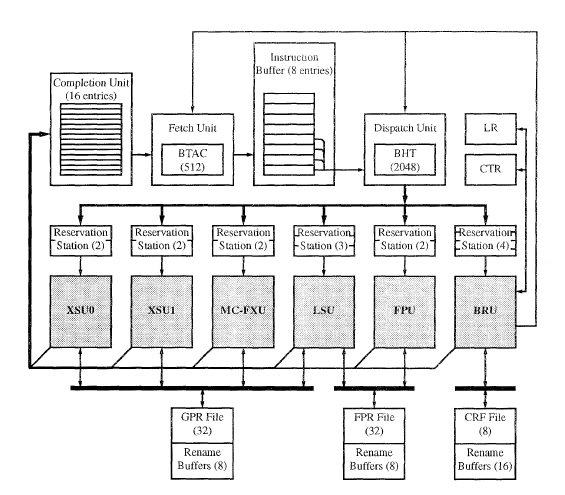
is based on IBM’s RS/6000 POWER architecture, designed to facilitate parallel instruction execution and be able to scale well with advancing technology. Motorola and IBM are the chief designers of the PowerPC architecture and the family of PowerPC chips, while Apple is focusing on PowerPC systems and software.

As of this writing, the PowerPC alliance has released three chips and announced a fourth. The first, which provided a transition from the POWER architecture to *PowerPC,* is the PowerPC601 [101. The second, low-power chip is the PowerPC60 3. Recently, a more advanced chip for desktop systems has begun production, the PowerP6C0 4. The fourth chip, the last one that will be designed by the alliance, is a high-performance chip. This chip, the PowerPC62 0 , was only recently announced.

The PowerPC architecture contains 32 integer registers (GPRs) and 32 floating point registers (FPRs). Also contains 32 condition register bits which can be addressed as one 32-bit register (CR), as a register with 8 four-bit fields (CRFs), or as 32 single-bit fields (CRBs). It also contains a count register (CTR) a link register (LR), both primarily used for branch instructions, and an integer exception register (XER) and a floating point status and control register (FPSCR) which s used to control the operation and record the exception status of the appropriate instruction types. PowerPC instructions are typical RISC, with the addition of floating point multiply-add fused (FMA) instructions and instructions to set, manipulate, and branch off of the condition register bits.

The most dominant architecture today, in terms of installed base, is the Intel x86. However the PowerPC appears to be making a serious challenge to this dominance. IBM is looking to the PowerPC as the new ISA for the entire company. Motorola has an agreement to supply Ford with PowerPC-based processors for future automotive electronics. Motorola is also planning on employing PowerPC processors in their products ranging from hand-held devices to multiprocessor servers. Future systems from Apple will employ PowerPC processors. In addition, Apple, IBM and Motorola have recently announced their agreement on developing common/compatible hardware platforms based on the PowerPC processors.

The 620 is a 4-wide superscalar machine. It uses aggressive branch prediction to fetch instructions as soon as possible and a generalized dispatch scheme to get those instructions to the execution units.



**Fig 1: Power PC micro architecture diagram**

The PowerPC 620 uses six parallel execution units. It has two simple (single-cycle) integer units, one complex (multi-cycle) integer unit, one floating-point unit (3 stages), one load/store unit (2 stages), branch unit. The branch unit accepts condition register logical instructions as well as branches. To keep these execution units as full as possible, the 620 uses distributed reservation stations and register renaming to implement an aggressive out-of-order execution scheme.

The 620 processes instructions are five major stages, some of which are separated by buffers to take up slack in dynamic variations of available parallelism. Major stages can require multiple cycles, while minor stages are one cycle each. The pipeline stages are **Fetch, Dispatch, Execute, Complete,** and **Write back.** The first three stages are followed by the Instruction Buffer, Reservation Stations, and the Completion Buffer, respectively. See figure 2.



**Figure 2: power PC instruction Pipeline**

**Fetch Stage:** The fetch unit accesses the instruction cache to retrieve up to four instructions per cycle into the instruction buffer. The end of a cache line or a taken branch can prevent the fetch unit from getting four useful instructions in a cycle, and a miss predicted branch can waste fetch cycles while fetching down the wrong path. Also during the fetch stage a preliminary branch prediction is made via the Branch Target Address Cache (BTAC) and the predicted next address is used for fetching in the next cycle.

**Instruction:** Buffer: The instruction buffer holds instructions between the fetch and dispatch stages. If the dispatch unit cannot keep up with the fetch unit, instructions will stay here for multiple cycles until the dispatch unit can get to them. A maximum of eight instructions can be buffered at a time. Instructions are buffered and shifted in groups of two to simplify the logic.

Dispatch Stage: The dispatch unit decodes instructions in the instruction buffer and checks if they can be dispatched to reservation stations. If all dispatch conditions are fulfilled for an instruction, the dispatch stage will allocate a reservation station entry, a completion buffer entry, and any needed destination rename registers for it. If these resources are available and each instruction goes to a different execution unit and all special serialization constraints are met, up to four instructions may be dispatched in order per cycle.

There are eight general-purpose register rename buffers, eight floating-point register rename buffers, and sixteen condition register field rename buffers. During dispatch, the necessary number of these buffers is allocated for the results of the instruction. Also during dispatch, any source operands which have been renamed by previous instructions are marked with the tags of the associated rename buffers. The reservation stations will then watch the appropriate result buses for forwarded results.

If a branch is being dispatched, resolution of the branch is attempted immediately. If the branch depends on an operand that is not yet available, it is predicted via the Branch History Table (BHT). If the predicted or resolved address does no match that made by the BTAC during the fetch stage, the speculatively fetched instructions are canceled and fetching restarts at the new address.

**Reservation Stations:** Each execution unit contains a reservation station to hold those instructions waiting to execute there, each reservation station can hold two to four instruction entries, depending on the execution unit. Each dispatched instruction waits in a reservation station until its entire source operands have been read or forwarded and the function unit is ready for execution. An instruction may then leave the reservation station and enter its function unit out of order. Each execution unit contains one reservation station and one reservation unit.

Execute Stage: A function unit computes the results of an instruction. This major stage can require multiple minors stages (cycles) to produce its results, depending on the type of instruction being executed. At the end of execution, the instruction results are sent to the destination rename buffers and forwarded to any waiting instructions, and the instruction is marked finished in the completion buffer.

Completion Buffer: The completion buffer holds the states of the in-flight instructions until they are architecturally complete. The completion buffer has sixteen entries with which to hold instruction information. An entry is allocated for each instruction during the dispatch stage. The execute stage then marks it as finished when the unit is done executing the instruction. Once an instruction is finished, it is eligible for completion.

**Complete Stage:** During the completion stage, finished instructions are removed from the completion buffer in order, up to four at a time, and passed to the write back stage. Fewer instructions will complete in a cycle if there are an insufficient number of write ports or if fewer than four instructions are finished and ready to complete in order. By holding instructions in the completion buffer until write back, the 620 guarantees that the architected registers hold the correct state up to the most recently completed instruction, thus allowing precise interrupts even with aggressive out-of-order techniques.

**Write back Stage:** During this stage, the write back logic moves the results of those instructions co completed by the completion unit in the last cycle from the rename buffers to the architected register files.

**Machine Specification**

VMW is configurable to a specific machine implementation through four template tiles and a fifth C++ behavior tile. Two of the template files specify the syntax and semantics of the architected instructions, while the other two specify the organization and timing of the micro architecture. The C++ behavior file uses the specifications in the template files along with special-case routines that cannot be coded in the template files to model the execution of each instruction through the machine. To compare the micro architectural complexity of the PowerPC 620 with that of various other machines, we show a comparison with other simulators targeted using VMW.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Microprocessor** | **Alpha 21064** | **RS/6000** | **PowerPC 601** | **PowerPC 620** |
| Instruction Syntax (Templates) | 487 | 405 | 383 | 383 |
| Instruction Semantics (Templates) | 37 | 125 | 205 | 205 |
| Instruction Timing (Templates) | 24 | 87 | 112 | 113 |
| Machine Organization (Templates) | 39 | 60 | 51 | 75 |
| Machine Behavior (Lines of Code) | 668 | 1132 | 1080 | 3,093 |

**Table 1: Comparison of Sizes of Machine Description Files**

The five machine description files for the PowerPC 620 are generated based on internal confidential design documents provided and periodically updated by the PowerPC 620 design team. Correct interpretation of the design documents is checked by a member of the design team through a series of refinement cycles as the 620 design is finalized. Through this process, even late changes to the 620 design were incorporated into our machine description files and modeled by our performance simulator. Validation of the simulation model includes syntactic checking of the description files using tools provided by VMW and timing checking by comparing timing results on small benchmarks with lower-level hardware models.

**Experimental Framework**

To analyze the PowerPC6 20 micro architecture, we used the Visualization-based Micro architecture Workbench (VMW), a retarget table performance simulator. The targeting of VMW for a specific processor is performed by specifying a set of machine description files. Once the files are specified, VMW provides the generic simulation and visualization engines and automatically compiles the files into a performance simulator for the specified machine.

**PowerPC 620 Instruction Fetching**

Provided that there is enough room in the instruction buffer, the 620’s fetch unit is capable of fetching four instructions every cycle. That is, up until it encounters a branch. If the fetch unit were to wait for branch resolution before continuing to fetch non-speculatively, or if it were to bias naively fear branch not-taken., machine execution~ would be drastically slowed by the bottle neck in fetching down taken branches. Because of this, accurate branch prediction in a superscalar machine as wide as the PowerPC6 20 is extremely important. To demonstrate the branch prediction and speculation characteristics of the 620, we chose one of the integer benchmarks as an example case. We choose *espresso* because it has the most interesting characteristics.

**PowerPC 620 Instruction Dispatching**

The PowerPC 620 dispatches instructions in order. This causes the dispatching unit to be a potential bottleneck. To begin with, however, the dispatch unit must be suitably supplied with instructions by the fetch unit. The rest of the examples in Sections 5 through 7 are from the *Alvin* benchmark in order to display the use of the floating-point functional units.

**Summary**

The PowerPC6 20 does very well at branch prediction, often having zero delay cycles even for a taken branch. And even though precise interrupt is implemented, there is still a high degree of parallelism and out-of-order execution, especially with a well mixed stream of instructions, such as that in the *alvinn* benchmark. If the instruction mix is bustier, it tends to create a bottleneck in the dispatch unit. Even so, the integer benchmarks do quite well, and even the *hydro2d* benchmark it its high number of floating point divide instructions, still manages to achieve an IPC that RISC designers of a few years ago could only dream about.

One hot spot is the load/store unit. Although the difficulties of designing a two load/store unit system are myriad, the load/store bottleneck in the 620 is evident, and future, wider, processors will need that second unit. Having only one floating-point unit for four integer units is also a source of bottlenecks. The integer benchmarks rarely stall on the integer units, but the floating-point benchmarks stick waiting for FP resources often. The single dispatch to each reservation station in a cycle is also a serious source of dispatch stalls which can reduce the number of instructions considered for out-of-order execution.

**Improved Out-of-Order Execution**

Although other RISC vendors are just beginning to implement out-of-order execution, the 620 is the third

PowerPC chip to use this technique. Like its predecessors, the new design includes “reservation stations” in each of the function units to hold instructions that are waiting for operands. The integer and floating-point units each have two reservations stations, as in the 604; the new branch unit has four reservation stations, permitting speculative execution beyond as many as four conditional branches. The 620 improves up on its predecessor in the load/store unit by adding a third reservation station.

The load/store unit will also perform some loads speculatively, that is, when a preceding branch instruction has been predicted but not resolved. To prevent operations that cannot be undone, the 620 will not speculatively execute stores of any kind or loads from non catchable (I/O) segments. Because most loads and stores are queued, as in the 604, these operations typically will not stall the processor pipeline.

The 620 handles unaligned loads and stores in hardware in both big- and little-endian modes, allowing it to do well in mixed environments with older Apple Macintoshes and x86-based PCs.

The other function units are quite similar to those in the 604. The integer units, of course, are expanded to 64 bits in width. The latency of floating-point loads is reduced from three cycles to two, and the latency of floating- point divides is improved to 18 cycles. The 620 adds a hardware square-root function that completes in 22 cycles; in the 604, square roots are calculated in software. These changes will improve performance on Spice circuit models, among other applications. As in the 604, all FP operations are performed in double-precision; there is no speed improvement for single-precision calculations.

**Two-Level Cache Structure Added**

Although the 60x PowerPC chips can operate with second-level caches, all control functions for the L2 cache must be implemented in external logic. Furthermore, these chips share a single interface between the L2 cache and other system accesses. The 620 solves both of these problems, significantly improving performance.

**Memory Bandwidth Increased**

The 620 system bus permits a low-cost 64-bit interface or a high-performance 128-bit design. Even the 64- bit version will provide much better memory bandwidth than the 64-bit 60x bus, as all cache traffic is shifted to the dedicated cache bus. This mode should be adequate for uniprocessor servers or multiprocessor workstations.

The 128-bit mode may be required to meet the high bandwidth requirements of MP servers.

**Cost of Complexity Is High**

The 620 uses a slightly different manufacturing process than the 603 and 604: a four-layer-metal version of a process that IBM calls CMOS-5S. This process reduces the gate oxide thickness, speeding the transistors. The faster transistors, along with improved circuit designs, deliver a clock rate 33% faster than that of the 604. The added features increase the die size of the 620 to 311 mm2, 59% larger than the 604. The new chip would have been even larger, but Motorola agreed to adopt IBM’s C4 (flip-chip) bonding method.