Beyond RISC - The Post-RISC Architecture

Authors: Mark Brehob, Travis Doom, Richard Enbody, William H. Moore, Sherry Q. Moore, Ron Sass, Charles Severance Michigan State University Department of Computer Science

Keywords:
CPU Architecture, CISC, RISC, Dataflow, Pipeline, Superscalar

You may be interested in an on-line version of a Post-RISC lecture which is based on this paper.

Abstract

The principles of the RISC architecture guided the design of the previous generation of processors. These principles have accelerated the performance gains of these processors over their predecessors. The current generation of CPUs is poised to continue this rapid acceleration of performance. In this paper we assert that the performance gains of the current generation are due to changes that are decidedly *not* RISC. We call this current generation of processors Post-RISC and, we describe the characteristics of Post-RISC processors. In addition, we survey six processors of the current generation that highlight the principles of Post-RISC.

1. Introduction

The current generation of processors introduces an exciting new era of high performance processors. These processors uniformly show dramatic increases in performance of a scale that has not been seen since the late 1980s when RISC processors first became available [PatHen]. The changes that spurred the performance gains of RISC were clear because they were changes in instruction set architecture (ISA). The changes that are spurring the performance gains of recent processors, however, are more subtle. In this paper, we survey processors from DEC, HP, IBM, Intel, MIPS, HAL, and Sun. While there are some changes that impact the ISA such as the Intel MMX and Sun VIS instructions, the performance gains are mostly due to features that are decidedly not RISC. We refer to these new features as Post-RISC. To study and characterize Post-RISC, we propose a generic Post-RISC processor in Section 2 and then use it, in Section 3, as a framework to explain the differences among the processors.

Post-RISC Characteristics

The most significant Post-RISC changes are to the implementation of the architecture. Superscalar RISC processors relied on the compiler to order instructions for maximum performance and hardware checked the legality of multiple simultaneous instruction issue. Post-RISC processors are much more aggressive at issuing instructions using hardware to dynamically perform the instruction reordering. The new processors find more parallelism by executing instructions out of program order.

Out-of-order execution is not a new concept in computing -- it existed twenty years ago on IBM and CDC computers -- but it is innovative for single-chip implementations. The result is a RISC ISA with an execution core that is similar to a dataflow implementation. However, these processors still adhere to most of the RISC concepts. For example, the execution units of these processors are optimized to complete most instructions in a single cycle.

2. The Generic Post-RISC Processor

Figure 1 shows the flow of processing in a generic Post-RISC processor. Each of the elements is described in the indicated section below. None of the real post-RISC CPUs are implemented precisely the same way as this discussion. It is an amalgam of features intended to illustrate many of the techniques used to implement the CPUs discussed in this paper. In this section, we describe the components generally.

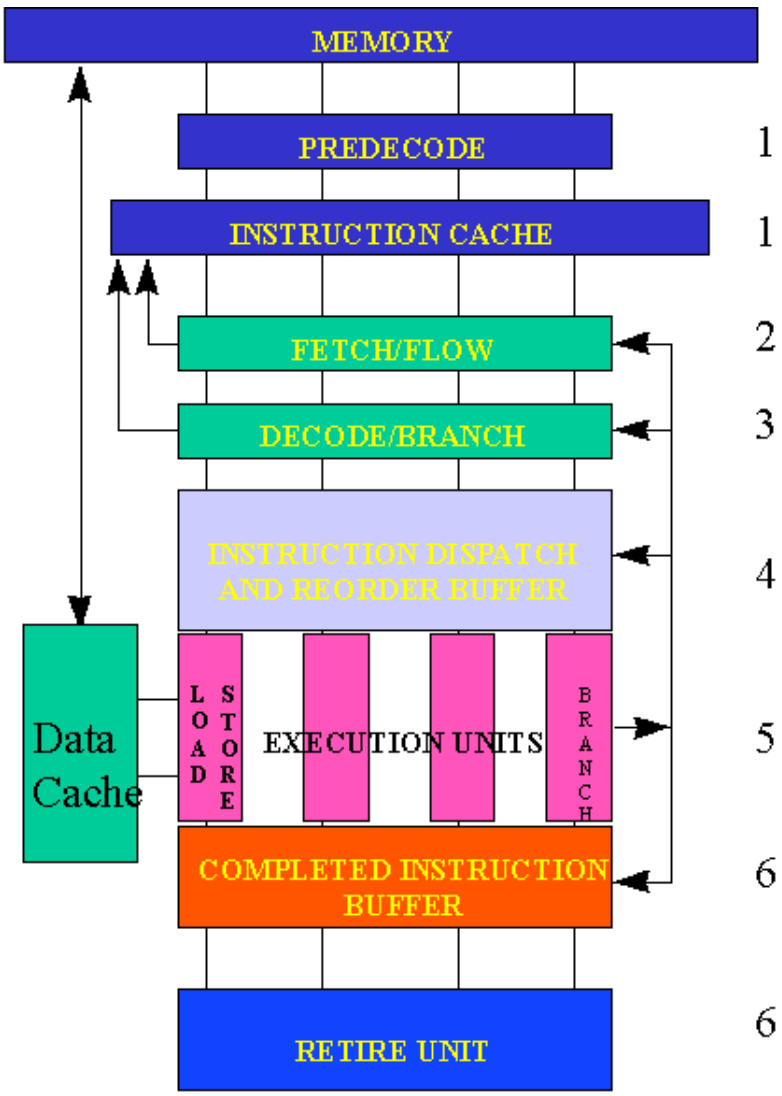

Figure 1: The Post-RISC Architecture

In this section we will follow an instruction through the pipeline, highlighting the Post-RISC features. The example Post-RISC CPU is a 4-way superscalar processor so each pipeline step processes groups of four instructions. Instructions are first fetched, decoded, and then buffered. Instructions can be dispatched to execution units out of program order as resources and operands become available. Instructions can be fetched and dispatched speculatively based on predictions about branches taken. The result is a pool of instructions in varying stages of execution, none of which have completed by writing final results. As resources become available and branches are resolved, instructions are retired in program order. This preserves the appearance of a machine that executes the instructions in program order. We begin with an instruction being fetched from memory by the predecode unit into the instruction cache (I-cache).

Predecode and I-cache (1)

Predecode occurs as the instructions are fetched from memory in groups of four. The instruction is augmented with a few characteristic bits and stored in the I-cache. These additional bits (1) identify the instruction as a branch or not, (2) indicate

the type of execution unit needed, and (3) determine whether or not the instruction will make a memory reference. Predecode provides a performance advantage by reducing the logical complexity of the regular decode stage and by supplying information about the instruction to stages before the decode stage.

The I-cache (Figure 2) is also used to store other information about the instructions. Every group of four instructions has several additional bits used for flow history. Every instruction also has a branch history. The use of these two fields is described below.

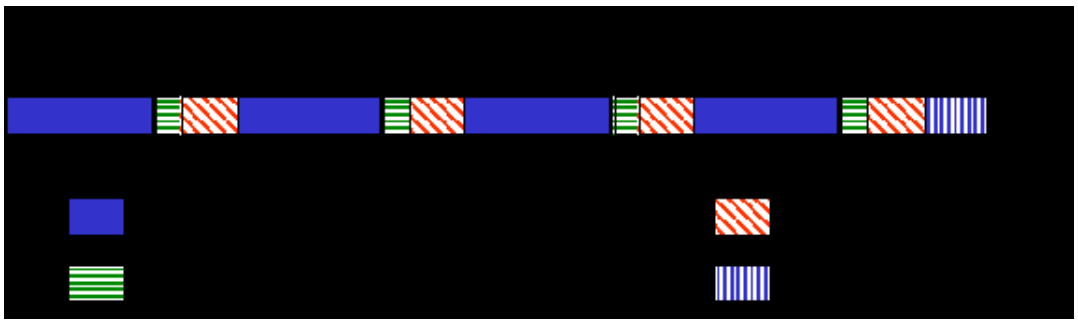

Figure 2: I-cache Entry

Fetch/Flow Prediction (2)

As is true with most pipelined processors, deciding which instructions to fetch from the I-cache is an educated guess and a wrong guess is costly.

In the Post-RISC processor, branch prediction does not occur early enough in the pipeline to be used to predict the next fetch. Typically, the branch prediction is done as part of the decode operation. This information is available to the instruction fetch unit 1-3 cycles too late. The instruction fetch unit must know the next address to fetch before the decode phase even starts.

To allow the instruction fetch unit to fetch the next instruction without waiting for decode phase and branch lookup, flow history bits are added to each group of four instructions. These bits point to the next group of four instructions to fetch. If the bits point to the wrong group of instructions, these bits may get updated later based on the prediction algorithm or branch execution.

Decode/Branch (3)

At this stage of the pipeline, the instruction is decoded and more accurate branch prediction is performed.
At each branch instruction, the flow of the program diverges into two separate instruction streams. Until the branch instruction is resolved, the actual instruction stream to be executed is unknown. Rather than wait idly, the CPU makes a prediction based upon certain heuristics and the past behavior of that instruction. For example, a single bit in the branch prediction table can indicate whether to take a branch or not. As the CPU decodes each branch, it changes the color of the following instructions. If the prediction is later found to be in error, using instruction color, all results of speculatively executed instructions beyond the branch will be discarded.

There may be a number of unresolved speculative branches outstanding. In Figure 3 a number of instructions are shown with the branch prediction indications. The branch in instruction 3 is predicted as taken and the branch at instruction 9 is predicted as not taken. The decode stage simply issues instructions as quickly as possible marking each instruction with the colors shown. Instruction 10 may be speculatively executed at any time but if the branch at instruction if the branch at instruction 9 is taken (prediction incorrect) then all the green instruction results will be discarded.

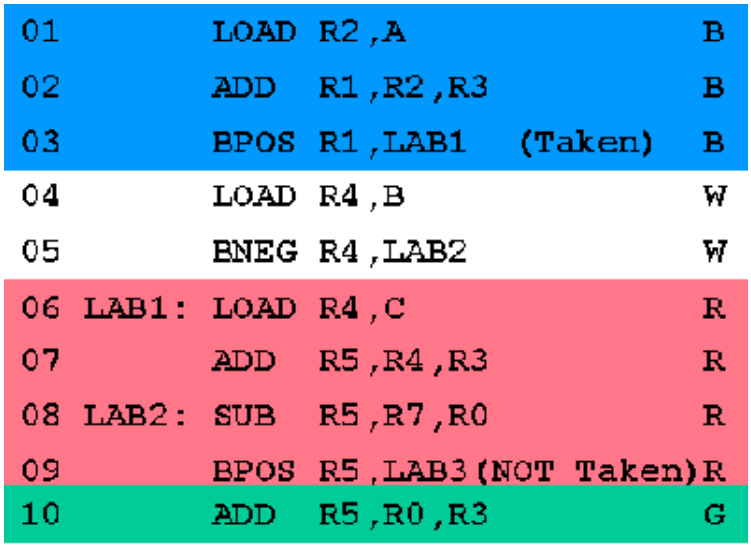

Figure 3: Decode and Dispatch

Instruction Dispatch and Reorder Buffer (4)

In the instruction dispatch and reorder buffer, the instructions are queued waiting to be dispatched. Instructions are ready to be dispatched when their input values are available, an output register is available, and an execution unit is available. Older instructions and load instructions are given priority. Once instructions enter the buffer, program order is a secondary concern.

Integral to this scheme is the use of rename registers. Rename registers hold results until the instruction retires. At retirement, the rename registers are either copied to the architectural register named in the instruction or a table is updated. Rename registers are important because they eliminate anti-dependencies and output dependencies in the instruction stream. Instructions can read and write to rename registers so their execution can proceed while earlier instructions have locked the architectural registers. Register renaming also makes implementing speculative execution easier. When a branch is resolved, all of the rename registers allocated for the wrong path are freed, and only instructions on the correct path (that is, with the appropriate color) are allowed to retire.

Figure 4 shows an example of the execution order of the previous code. We begin after the fetch, decode, and branch predict all have occurred and the instruction dispatch and reorder buffer is filled, following the branch prediction. In the first cycle, instructions 1, 6, 8, and 10 can execute. In the second cycle instructions 2, 7, and 9 can execute. In the third cycle, instruction 3 executes.

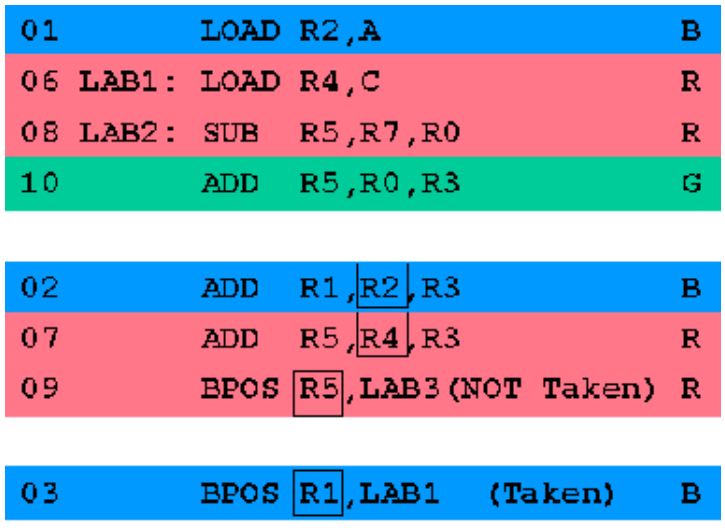

Figure 4: Execution Order

Execution Units (5)

Execution units in the generic Post-RISC processor are similar to RISC. As in RISC, most integer execution units complete in a single cycle. Units with a latency greater than one cycle are pipelined. Thus, every unit is capable of accepting a new operation every cycle. Some execution units are multi-cycle and not pipelined such as division or block-move.

The load/store units are capable of having quite a few (10-20) memory instructions in progress. These can complete in almost any order based on the locality of the data and the availability of the cache hardware or memory banks. With the ability to have multiple memory operations in progress at one time, the Post-RISC processor can dynamically create memory pipelines similar to the Gather/Scatter facilities present in supercomputer architectures.

The branch unit must communicate the results of a branch evaluation to the other units in the pipeline:

- The Fetch/Flow Unit must be informed of mispredictions so that it can update the flow history in the I-cache and begin fetching at the correct target address.
- The Branch/Decode unit must be informed so that it updates the branch history table. Updates happen even in the case of correctly predicted branches.
- The instruction dispatch and reorder buffer must be informed so that it can discard any instructions that should not be executed.
- The completed instruction buffer (see below) must be informed to discard instructions which will never be retired.
- If there are multi-cycle execution units working on instructions that will never be executed because of a branch, they also must be notified.
- If an instruction generates an exception, the exception is stored as a flag associated with the instruction. The exception is raised when the instruction is retired.

Completed Instruction Buffer and Retire Unit (6)

The Completed Instruction Buffer holds instructions that have been speculatively executed. Associated with each executed instruction in the buffer are its results in rename registers and any exception flags. The retire unit removes these executed instructions from the buffer in program order at a rate of up to four instructions per cycle. The retire unit updates the architected registers with the computed results from the rename registers.

When instructions are executed out-of-order, a special problem arises when one has an exception. The processor’s architecture stipulates that if an instruction has an exception, then the processor must stop at that point in the program. That is, effects from instructions after it should neither be reflected in the state of the machine nor should there be an unexecuted instructions before it. This characteristic is known as a "precise exception" or a "precise interrupt." By retiring instructions in order, this stage can maintain precise exceptions. Essentially, every instruction is executed speculatively: instructions are executed with the assumption that no instruction before it has caused an exception. When the retirement unit can verify this assumption is true, it retires the instruction.

In Post-RISC architecture, an important performance metric is the number of instructions retired in parallel. In contrast when looking at a RISC processor, the number of instructions which are issued in parallel is the important performance metric. Because the RISC processor executes instructions in order the issue rate is the same as the retire rate for the instructions.

```
01          LOAD  R2,A                  B

02          ADD   R1,R2,R3               B

03          BPOS  R1,LAB1   (Taken)     B
06  LAB1:   LOAD  R4,C                   R
07          ADD   R5,R4,R3               R
08  LAB2:   SUB   R5,R7,R0               R

09          BPOS  R5,LAB3(NOT)           R
10          ADD   R5,R0,R3               C
```

Figure 5: Retire Order

Continuing with the same example examining the retire operation, we assume that all 10 instructions have completely executed and are in the Completed Instruction Buffer. In Figure 5 only instruction 1 can retire during the first possible retire cycle. The other instructions remain in the buffer. Only instruction 2 can retire on the second retire cycle. On the third retire cycle, instruction 3 retires and assuming the branch prediction was correct, all of the red instructions become eligible to retire so instructions 6,7,8, and 9 retire. The branch in 9 is now resolved so instruction 10 is retired on the next cycle.

Post-RISC Pipeline

A pipeline diagram (Figure 6) for the Post-RISC Architecture The Post-RISC Pipeline consists of three connected three connected components: (1) Fetch/Decode section, (2) Execution Units, and (3) Retire Units. Between each of these components, there is a flexible queue of instructions. The Instruction Reorder Buffer connects the Fetch/Decode components and the Execution Units. The Completed Instruction Buffer connects the Execution units to the Retire Unit.

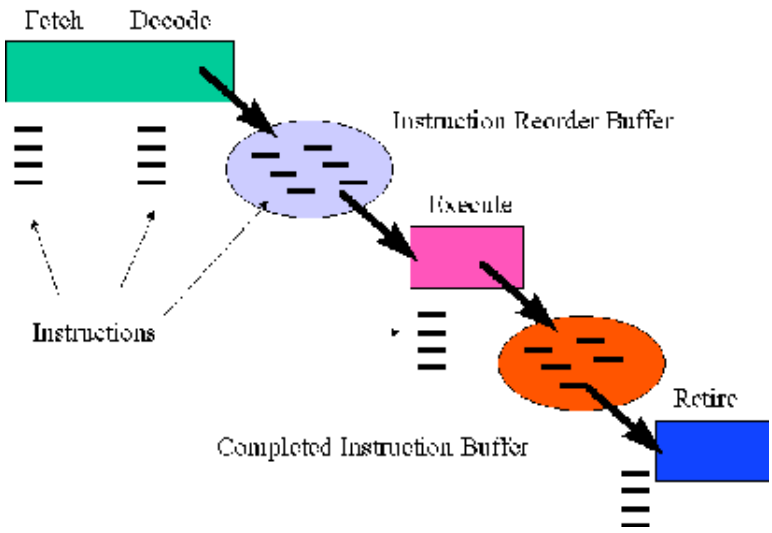

Figure 6: Post-RISC pipeline

Comparison to Dataflow Architectures

In many ways the instruction scheduling and execution of the generic Post-RISC processor is a dataflow architecture. Once in the instruction reorder and dispatch buffer, instructions are dispatched based on the availability of operands. In the instruction reorder buffer a value is either available or undefined.

In a classic dataflow machine (See Figure 6), a program consists of a set of instructions. The instructions can be executed in any order as long as the input operands required for the instruction are available. There can be many units executing instructions. As each instruction executes, it either produces a result or possibly a new instruction to be executed. New instructions are added to the instruction queue for future execution.

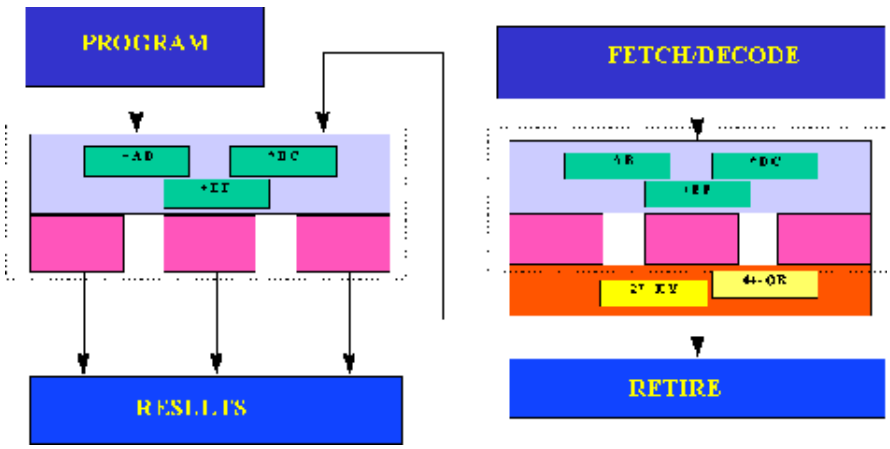

Figure 7: Classic Dataflow CPU compared with Post-RISC

A classic dataflow processor has two drawbacks [Dataflow] that do not exist in Post-RISC processors. One is the problem of determining when operands are available (called "matching" in dataflow terminology). Since all operands within a microprocessor are in registers, that determination is easy. The other major problem with dataflow is the handling of structures such as arrays. Since the core of a microprocessor deals with only simple data types (one per register), this tremendously complex problem for a general-purpose dataflow processor is simplified here. The result is a processor that can take advantage of the flexibility of dataflow at a small scale without its disadvantages.

The Post-RISC processor differs in two primary ways from a pure dataflow processor.

- By the virtue of the fact that the instruction reorder buffer is fixed in size, the "dataflow" aspects of the processor core only have to be able to handle a fixed number of active instructions. In a true dataflow architecture, there is no bound on the number of "active" instructions. This limitation makes the Post-RISC processor able to identify the "ready" instructions much more efficiently.
- After the execution units have executed the instructions from the queue, the results are retired in order enforcing the

expected sequential nature of the program, hiding the dataflow nature of the processor core.

4. Processor Families Overview

When we began this survey early in 1996, we classified two processors (DEC 21164 and Sun Ultrasparc-1) as traditional RISC processors and four processors (PowerPC 604, MIPS R10000, HP PA-8000, and Intel Pentium) as having some level of Post-RISC features. It seemed unclear as to whether or not the superior approach would be the Post-RISC or the traditional RISC approach. A year later there are Post-RISC processors under development or available for both the DEC [21264] and SPARC [HAL] architectures. In this section, we describe some of the features of some current and upcoming processors and highlight some of the features that each processor has relative to our Post-RISC CPU.

DEC Alpha 21164

The DEC 21164 is a traditional four-way superscalar RISC processor with very few of the Post-RISC features. The Alpha competes with the others because it has an extremely fast clock rate. At any point in time, DEC is typically shipping processors that have a clock rate twice as fast as the other CPU vendors. Although some of the parallelism is lost, the cost is less significant when each cycle is so quick. This is one of the primary tenets of the RISC and DEC 21164 philosophy.

The branch history table in the Alpha is maintained by the decode unit. The table has 2048 single-bit entries, and the bit indicates the direction of the most recent conditional branch. There is no additional flow prediction in the fetch unit.

The Alpha 21164 will not issue instructions out-of-order nor will the instruction issue stage combine partial blocks of instructions. So, if all instructions but one are issued, then in the next clock tick only that one instruction will be issued. Although instructions are issued in-order, multicycle instructions can complete out-of-order. Allowing out-of-order completion provides a low latency, simple, and fast pipeline, but does not provide precise interrupts.

Because of the strict in-order issue, this architecture relies heavily on the compilers to schedule instructions to best utilize their functional units and prevent data dependencies from stalling the pipeline.

DEC Alpha 21264

While the DEC Alpha 21164 processor pursues the 600Mhz performance and beyond, DEC is also developing the Alpha 21264 [21264] with out-of-order execution. While this processor is not available until at least the end of 1997, it demonstrates that out-of-order execution does not necessarily imply low clock speeds. The target clock speed of the 21264 is 500Mhz. Like most of the post-RISC processors, the 21264 is a 4-way super-scalar processor with out-of-order execution. While the 21164 had simple branch prediction, the 21264 adds a "next fetch predictor" (flow prediction) in each cache line. To allow for register renaming, the processor implements 80 integer and 72 floating point registers.

The 21264 can have 20 integer instructions and 15 floating point instructions in queues. Up to four integer and two floating point instructions can be issued per cycle. The floating-point add and multiply units are fully pipelined with a four cycle latency. Up to 32 Load/Store instructions can be active at one time. Of those memory operations, up to 8 off-chip reads and 8 off-chip writes can be active at one time.

While the 21264 (500Mhz) is not available, the estimates of its floating point performance are at 50 SpecFP95. As a comparison both the Alpha 21164 (500Mhz) and HP Pa-8000 (200Mhz) have around 20.0 SpecFP95. This is a factor of 2.5 increase in floating point performance at the same clock rate when compared to the Alpha 21164.

SUN UltraSPARC

The UltraSPARC-1[UltraSPARC] does not issue instructions out-of-order but it does have many of the features of a Post-RISC processor. Instructions are retrieved from memory, predecoded, and placed in the I-cache. Both branch history and flow history are also stored in the I-cache. Because the Sparc architecture uses a branch delay slot, four instructions can never have more than two branches. In each four instruction segment in the I-cache, there are two branch history entries (2 bits) and one "Next Field" entry which points to the next group of four instructions in the I-cache.

Instructions are issued in-order and retired in-order *except* for several multi-cycle instructions (such as Floating Point Divide,

Load/Store, and instructions issued to the Multiple Cycle Integer Unit). These long-latency instructions are retired out-of-order.

The UltraSPARC has two integer ALUs and five floating-point units. Any combination of two integer and two floating point instructions can be issued each cycle. Loads, stores, and branches are executed by an integer ALU but outstanding loads are tracked by the Load/Store Unit. The Load/Store unit has a 9-entry load buffer and an 8-entry store buffer. These buffers allow overlapping of load and store operations reducing the apparent memory latency.

Unlike the other instruction set architectures with simple register model, the SPARC architecture uses movable "register windows". Register windows add some complexity to register renaming. However as the following processor shows, it is not impossible to have out-of-order execution with register windows.

HAL SPARC64

The HAL SPARC64 [HAL] processor implements the SPARCV9 architecture with out-of-order-execution. The SPARC64 can have up to 64 instructions active at any time. There are ten execution units. Any combination of four fixed-point, two floating-point, two load/store, or one branch instruction can be issued in one cycle. Up to 12 load/store instructions can be active simultaneously. Branch prediction is done using a standard two-bit counter mechanism.

IBM PowerPC 604

The PowerPC 604 [PowerPC,Weiss] and other members of the PowerPC family incorporate most features of our generic Post-RISC CPU. The PPC 604, is four-way superscalar and instructions are fetched from an on-chip cache in four-instruction segments. Instructions are pre-decoded as they are loaded into the cache. During the fetch stage, basic flow prediction is performed using a 256-entry Branch Target Address Cache (BTAC).

Fetched instructions go into an eight-instruction queue where they are decoded and then dispatched. In one cycle, up to four instructions can be fetched into the queue and up to four can be dispatched from the queue. Branch prediction is handled in the dispatch stage using a 2-bit prediction scheme. The bits are stored in the 512-entry Branch History Table (BHT).

The dispatch unit forwards instructions to the appropriate functional units from the instruction queue. The reorder buffer of the PowerPC is unique in that it is distributed across the six functional units, each having its own two-entry "reservation station." Instructions in the reservations stations can be issued in any order as resources become available. When an instruction is dispatched, a logical rename register is allocated for the result of the operation. The PPC 604 has eight rename registers for floating point and twelve for general purpose registers. In addition, four loads and six stores can be buffered further increasing the number of instructions issued out of order. Results are moved into the appropriate architected register when the writing instruction has completed, i.e. it is no longer speculative. Upon dispatch, the dispatch unit also allocates an entry in the Completed Instruction Buffer in the Completion Unit. The Completion Unit provides in-order retirement of instructions for the PowerPC.

MIPS R10000

The MIPS R10000 [R10000] is an excellent example of the Post-RISC architecture. Targeted toward high-end graphics processing and fast floating-point operation, the MIPS R10000 aggressively executes instructions out of order, while guaranteeing in-order completion and precise interrupts.

The R10000 is a fully 4-way superscalar architecture. It can fetch, decode (with branch prediction), and schedule up to four instructions per cycle. Each instruction is pre-decoded as it is loaded into the I-cache. During the decode, branches are predicted (although only one branch may be predicted per cycle), output dependencies are identified, and rename registers are allocated.

After being decoded, the instruction is placed into one of three 16-entry instruction queues (integer/branch, floating point, or address) for scheduling. These queues supply instructions for five execution pipelines. Instructions may execute from these queues in any order. An instruction is dispatched from the queue when the required inputs and execution units are available.

Note that the R10000 does not implement flow prediction but instead relies on a Branch Resume Cache for mis-predicted branches. Branch prediction uses the 2-bit scheme and the branch prediction bits are stored in the I-cache. Each branch is

followed by a delay slot, so no more than two branches can be fetched in one cycle. Only one branch can be issued per cycle.

HP PA-8000

The heart of the PA-8000 [PA8000] is an out-of-order execution core with a 56-entry Instruction Reorder Buffer (IRB). This buffer is divided into a 28-entry buffer for arithmetic instructions and a 28-entry buffer for memory instructions. Instructions are fetched from a large off-chip synchronous SRAM cache. Access to the I-cache takes two cycles, so the PA-8000 uses flow prediction to pipeline the fetches. This flow prediction implementation uses a 32-entry, fully associative Branch Target Address Cache (BTAC). The PA-8000 differs from our abstract CPU in that the BTAC must be stored *inside* the CPU rather as a part of the cache because the BTAC must be accessible on every cycle.

Each cycle, up to four instructions can be inserted into either the ALU IRB or the memory IRB. Branch instructions go into both IRBs to help recover from mispredicted branches. The PA 8000 will place four instructions into the IRBs on every cycle, but branches count as two instructions, since they go into both IRBs. Branches are predicted at this point using a 256-entry Branch History Table (BHT).

Instructions in the IRB arbitrate for the execution units. Availability of input operands is the first priority. The second priority is to choose the "oldest" instruction. The arbitration is a set associative operation: there is no sequential search of the IRB to determine the appropriate instruction. During each cycle, at most two arithmetic and two memory instructions can be dispatched simultaneously.

There are two units that perform address computations associated with loads and stores. Once an address is computed, it is stored in the 28-entry Address Reorder Buffer. This buffer allows the memory control to track up to 28 loads, stores, and pre-fetches simultaneously and these memory operations can be completed in any order from the off-chip data cache or from main memory. Up to 10 off-chip data cache misses can be outstanding at one time.

Intel Pentium

The Intel Pentium-Pro [IntelP6] is an interesting blend of architectures. It still executes the x86 CISC instruction set, but the internal implementation is a high performance "Post-RISC" CPU. The P6 has an execution core which supports out-of-order execution much like the abstract CPU described in this paper. The most dramatic aspect of the P6 architecture is that the execution core is not executing Intel 486 instructions. The actual instructions are called "uops" or micro-operations. Each Intel x86 instruction is decoded and translated into a one or more uops, which are executed by the out-of-order core.

Instructions go through the P6 in three stages: First, the instruction is fetched and decoded. Second, it is executed in the out-of-order core. The final stage retires the instruction in original program order.

Fetching and decoding in the P6 is more complex than in other processors. Some x86 instructions are very complex and can decode into hundreds of micro-ops, although the number is usually less than three. As Figure 15 illustrates, the P6 takes a number of cycles to do this work. At the end of this first pipeline segment instructions are handled in order. Next, the uops are passed to the out-of-order engine.

The heart of the out-of-order core is the Reservation Station (RS). The P6 has a Register Allocation Table which maps the architected registers onto a larger set of physical registers located in the Reorder Buffer.

The RS will hold up to 20 micro-ops until they are ready for execution. The RS is the buffer between the first and second stages. If an instruction’s operands are ready and an appropriate execution unit is available, the instruction is dispatched. The P6 has five execution units: two integer ALUs, 2 load/store units, and one floating point unit. Thus, the instructions may be executed out of order.

The third stage retires instructions. When the micro-ops complete, they wait in the reorder buffer until all of the instructions that came before them (in program order) have been retired. Once that has happened then that instruction may retire. This activity occurs during the third segment of our pipeline.

The most impressive aspect of this architecture is that the P6 maintains x86 compatibility. The designers handle a number of archaic (early 1980’s) architectural characteristics, such as self-modifying code, which did not concern other designers. Nonetheless, the Pentium-Pro performance is competitive with other processors in the Post-RISC generation.

5. Conclusion

The architectures of modern processors are moving toward a fundamental change in approach that we have called the Post-RISC architecture. In order to utilize increasing numbers of high-speed functional units, the processors must optimize instruction schedules at run-time. Dynamic, out-of-order scheduling is a promising technique to keep functional units busy and it is being used in many new processors.

Three interesting concepts emerged from our investigation of Post-RISC architectures. (1) An out-of-order scheduling implementation is quite similar to classic data flow architecture. The restricted internal environment of a microprocessor trivializes two of the major problems of data flow implementations: matching (determining which instructions can be executed) and the handling of structures (they do not exist). (2) Once the infrastructure is in place to allow dynamic scheduling, an implementation can have a different ISA in the core of the microprocessor. Intel used this feature to create a dynamically scheduled RISC core with its own easier-to-schedule instruction set (micro-ops) to implement its external CISC architecture. Keeping the cycles short mitigates the penalty of unused cycles in functional units. Time will determine which implementation is best, especially as more functional units are added.

These Post-RISC processors reduce the growing disparity between processor and memory speeds. The combination of rescheduled instructions with buffered loads and stores allows some memory latency to be hidden. However, the fundamental problem of growing memory latency with respect to processor speed remains.

It is important that as we study the performance characteristics of these new architectures and develop applications and compilers, we continuously evaluate how well these new architectures fare.

References

[21164] Keller, J., "The 21264: A Superscalar Alpha Processor with Out-of-Order Execution", 1996 Microprocessor Forum, http://www.digital.com/info/semiconductor/a264up1/index.html

[Alpha] Edmondson, J, Rubinfeld P, Preston R, Rajagopalan V, "Superscalar Instruction Execution in the 21164 Alpha Microprocessor", IEEE Micro, April 1995, page 33-44.

[Dataflow] Veen, A H, "Dataflow Machine Architecture", ACM Computing Surveys, Volume 18, No 4, December 1986, pages 365-396.

[PatHen] Patterson D, Hennessy J, *Computer Organization & Design: The Hardware/Software Interface*, ISBN-1-55860-281-X, 1994, Morgan Kaufmann Publishers.

[HAL] HAL Computer Systems, "The SPARC64 Processor", http://www.hal.com/docs/spar64.html

[IntelP6] Colwell R, Steck R, "A Technical Tour of the Intel Pentium(R) Pro Processor: A 0.6um BiCMOS Processor Employing Dynamic Execution", International Solid State Circuits Conference (ISSCC), Feb. 1995. See also http://www.intel.com/procs/p6/isscc/

[R10000] Heinrich, Joe. "MIPS R10000 Microprocessor User's Manual." MIPS Technologies, Inc. Alpha Draft of October 11, 1994.

[PA8000] Hunt, D., "Advanced Features of the 64-bit PA-8000", Proceedings 1995 IEEE COMPCON, Pages 123-128.

[PowerPC] Song S, Denman M, Chang J, "The PowerPC 604 RISC Microprocessor", IEEE Micro October 1994, Pages 8-17.

[Specint-92] "CPU Info & System Performance Summary", http://infopad.eecs.berkeley.edu/CIC/summary/

[Weiss] Weiss, Shlomo and James E. Smith. "POWER and PowerPC: Principles, Architecture, and Implementation." Morgan

Kaufmann Publishers, Inc., 1994.

[UltraSPARC] "The UltraSPARC Processor -- Technology White Paper",
http://www.sun.com/sparc/whitepapers/UltraSPARCtechnology/

Submitted to IEEE Micro May 1996
Submitted to IEEE Computer May 1997