Beyond RISC - The Post-RISC Architecture

Submitted to: IEEE Micro 3/96

An updated version of this paper is avaiable at http://www.egr.msu.edu/~crs/papers/postrisc2/

An on-line lecture based on this paper is available at http://www.egr.msu.edu/~crs/papers/talks/postrisc/

Authors: Mark Brehob, Travis Doom, Richard Enbody, William H. Moore, Sherry Q. Moore, Ron Sass, Charles Severance
Michigan State University Department of Computer Science
Technical Report CPS-96-11

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

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 which 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 which 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 six recently released processor from DEC, HP, IBM, Intel, MIPS, and Sun. We contend that 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 six recent processors.

Motivation

The transition from CISC to RISC was a radical change in architecture. Instruction sets were changed, sacrificing binary compatibility for performance. By reducing the instruction set, the processor fit onto a smaller chip which allowed designers to increase the clock speed. Furthermore, the processor could be pipelined and a microcoded control unit became unnecessary. RISC's "make the common case fast" principle led to a dramatic increase in performance over the previous CISC designs. The earliest RISC processors such as the IBM RT, SPARCv7, and MIPS R2000 adhered strictly to the fundamental principle of RISC. However, as the technology advanced to increase die sizes and increase transistor density, RISC processor designers began to consider ways to use this new chip space. Some of the uses for the space included:

- additional registers
- on-chip caches which are clocked as fast as the processor
- additional functional units for superscalar execution
- additional "non-RISC" (but fast) instructions
- on-chip support for floating-point operations
- increased pipeline depth
- branch prediction

Thus, the current generation of high performance processors bear little resemblance to the processors which started the RISC revolution. Any instruction or feature which improves the overall price/performance ratio is considered for inclusion.

As each processor family introduces its newest processor, an interesting trend in the relationship between Specint-92 and the clock rate can be observed. The performance as measured by Specint-92 increases much faster than the clock rate, as shown in Table 1 [Specint-92]. (Specint-92 was chosen in this table because of the availability of historical data.)

Company	Old Processor	SpecInt92	New Processor	SpecInt92	% Increase Clock Rate	% Increase SpecInt92
DEC	21064a 300 MHz	220	21164 333 MHz	400	11%	82%
HP	PA-7150 125MHz	136	PA-8000 133MHz	360	6%	164%
IBM	PPC-601 80MHz	91	PPC-604 133MHz	176	66%	93%
Intel	Pentium 166MHz	198	P6 200MHz	320	20%	62%
Sun	HyperSparc 125MHz	131	UltraSPARC 140MHz	200	12%	52%
MIPS	R-4400 200MHz	141	R-10000 200MHz	300	0%	113%

Table 1: Trends in Specint-92

By characterizing the processors, we can understand the source of their performance gains and lay the groundwork for future innovation.

Post-RISC Characteristics

The Post-RISC generation is distinguished from superscalar RISC processors in several ways. First, the ISA was further augmented. Some decidedly non-RISC instructions have been introduced to increase performance. We call this trend in ISA "Fast Instruction Set" or FISC because the instruction set is growing. The most significant changes are to the implementation of the architecture. Superscalar RISC processors relied on the compiler to order instructions and hardware checked the legality of multiple 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.

The new components in a Post-RISC implementation include: a predecode unit whose results are stored with the instruction in cache, the use of rename registers to remove output and anti-dependencies, an instruction reorder buffer, and a retire unit. These components and their performance advantages are discussed in the next section.

2. The Post-RISC Processor

Figure 1 shows the pipeline of a generic Post-RISC processor. Each of the elements is described in the indicated section below. None of the real CPUs in this paper 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. See later sections for specific information about each processor.

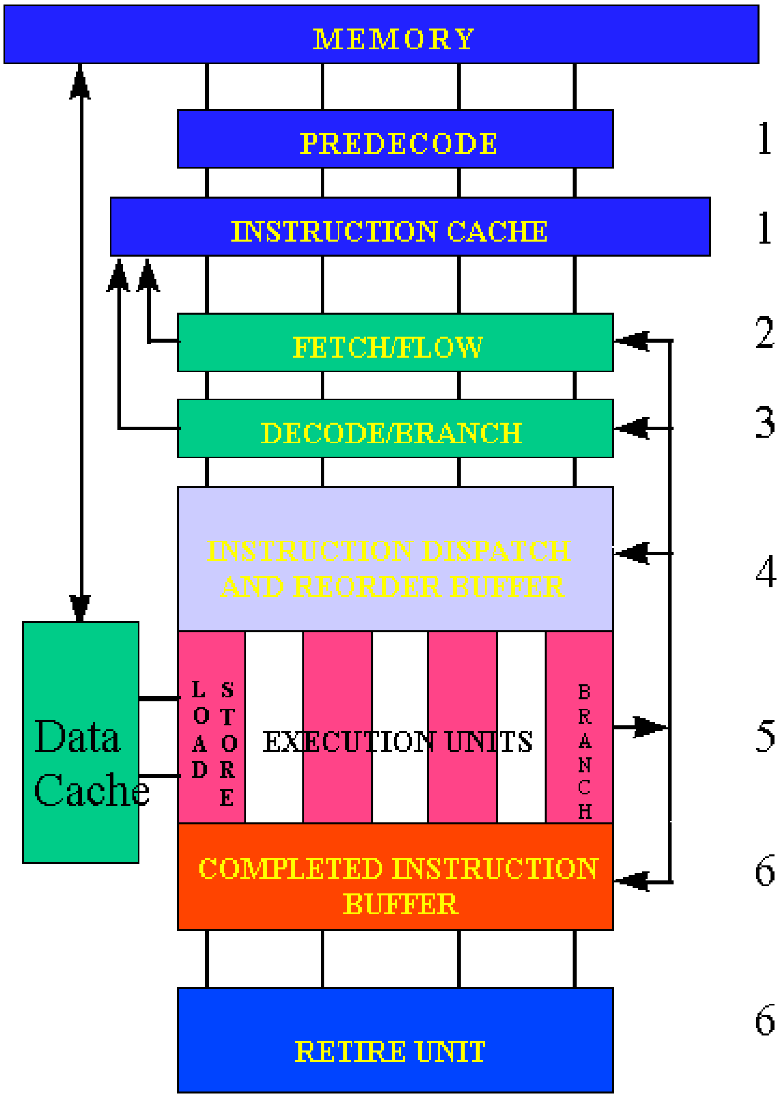

Figure 1: The Post-Risc Architecture

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 which executed the instructions in program order. We begin with an instruction being fetched from memory by the predecode unit into the instruction cache (I-cache).

1. Predecode and I-cache
   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. The 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.

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.

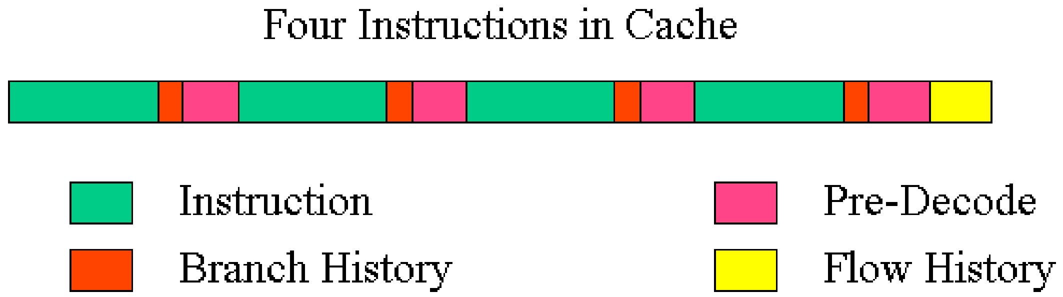

Figure 2: I-cache Entry

2. Fetch/Flow Prediction

   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 step even starts.

   To allow the instruction fetch unit to fetch the next instruction without waiting for decode 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.

3. Decode/Branch

   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 phase 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.

```
01        LOAD R2,A
02        ADD  R1,R2,R3
03        BPOS R1,LAB1   (Taken)
04        LOAD R4,B
05        BNEG R4,LAB2
06 LAB1:  LOAD R4,C
07        ADD  R5,R4,R3
08 LAB2:  SUB  R5,R7,R0
09        BPOS R5,LAB3  (NOT Taken)
10        ADD  R5,R0,R3
```

Figure 3: Decode and Dispatch

4. Instruction Dispatch and Reorder Buffer
   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 renamed registers so their execution can proceed while preceding 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.

```
01        LOAD R2,A
06 LAB1:  LOAD R4,C
08 LAB2:  SUB  R5,R7,R0
10        ADD  R5,R0,R3
```

```
02        ADD  R1,R2,R3
07        ADD  R5,R4,R3
09        BPOS R5,LAB3  (NOT Taken)
```

```
03        BPOS R1,LAB1   (Taken)
```

Figure 4: Execution Order

5. Execution Units
   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 often capable of having quite a few 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.
   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 which 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.
6. Completed Instruction Buffer and Retire Unit
   The Completed Instruction Buffer holds instructions which 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 a 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.

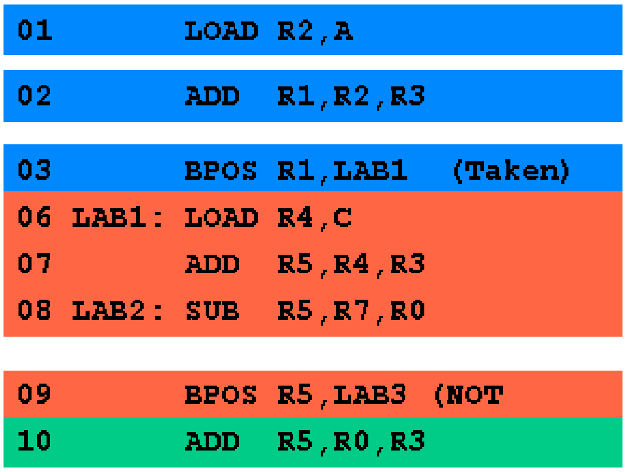

Figure 5: Retire Order

Continuing with the same example examining the retire operation, 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.

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.

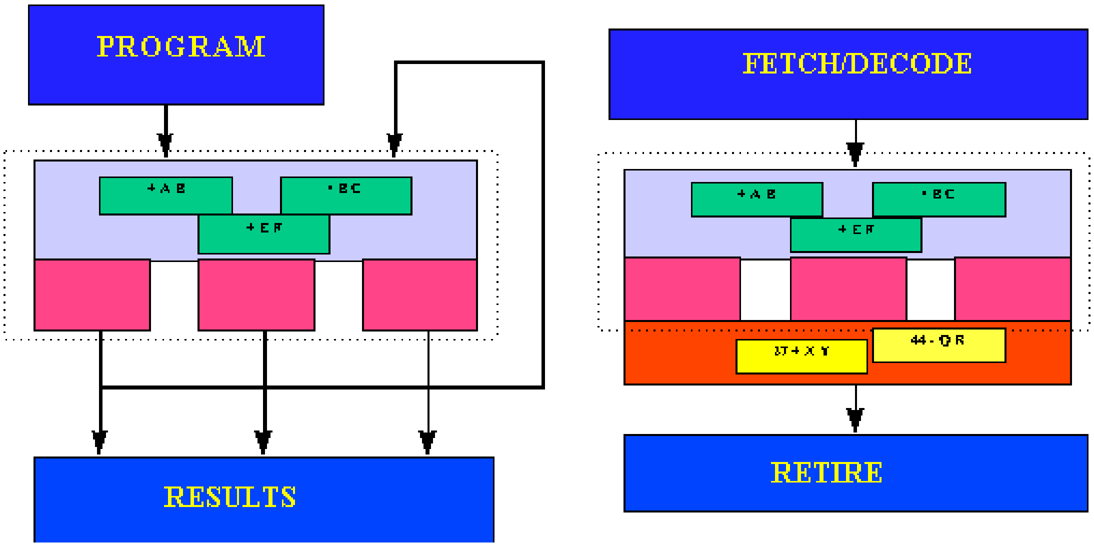

Figure 6: Classic Dataflow CPU compared with Post-RISC

A classic dataflow processor has two drawbacks[Dataflow] which 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 which 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 Overview

In this section we survey six processors. For each processor we discuss how it relates to the Post-RISC architecture. To some extent, every processor here incorporates some of the features we have described as Post-RISC. The processors chosen for this section range from traditional RISC to very Post-RISC in their architecture. We begin with the more traditional RISC processors.

DEC Alpha 21164

The DEC 21164 (Figure 7) 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 chips that have a clock rate twice as fast as the other CPU vendors. This high clock rate limits the complexity of each stage in the pipeline and forces the designers to avoid most Post-RISC features in order to maintain the high clock rate. Although some of the parallelism is lost, the cost is less significant when each cycle is so cheap. This is one of the primary tenets of the RISC and DEC Alpha 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 because instructions are in the on-chip cache. A misprediction by the decode unit only takes one extra cycle.

The Alpha has an 8K instruction cache, an 8K data cache, and a 96K L2 cache on chip. The L1 cache is relatively small compared to the others' 16K-32K L1 caches, but its small size allows single-cycle access even with the Alpha's fast clock. It also has a prefetch buffer containing four blocks of 32 bit instructions. According to DEC, having a two level on-chip cache gives better overall performance than a larger, but slower, L1 cache on chip. Since the 8K cache is less than a page and uses the virtual address as a tag, it has the further advantage of allowing partial overlap of the cache lookup and TLB operations.

The Alpha also supports an optional off-chip L3 cache. The L3 cache is wave pipelined in the sense that the address on the address bus can change before the data bus is read by the processor. This feature essentially allows the L3 cache to be in the process of fetching multiple words at the same time. The effect is quite similar to using pipelines to increase the rate of instruction execution.

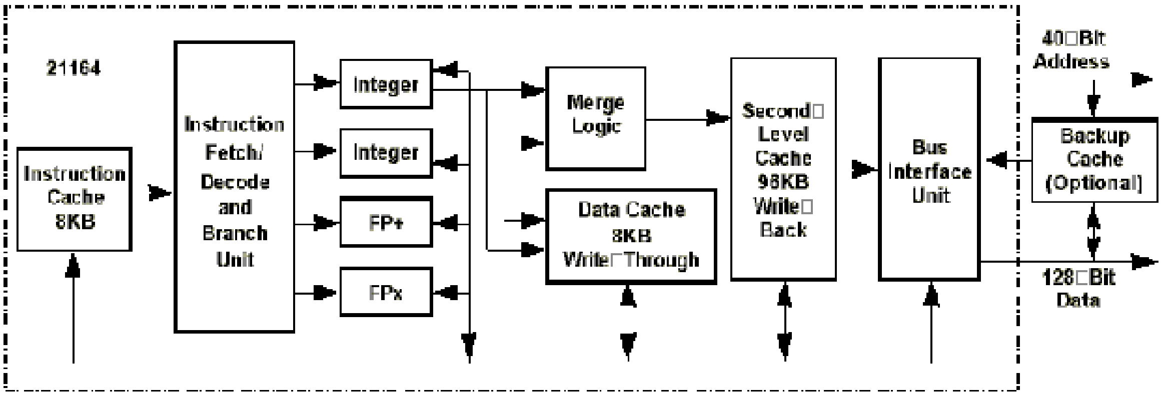

Figure 7:DEC 21164 Architecture

As Figure 7 shows, the Alpha's pipeline has 5 stages. Every stage takes a single cycle except, in some cases, the execution unit will take more than a one. The fetch and decode stages are similar to non-superscalar processors. A slot stage within the Fetch/Decode unit takes the next 4 decoded instructions. As resources become available instructions are issued in order from this group of four until all have been issued.

The Alpha will not issue instructions out-of-order nor will the slot and instruction issue stages combine partial blocks of instructions. So, if all instructions but one are sent from the slot stage to the instruction issue stage, then in the next clock tick only that one instruction will move to the instruction issue stage.

Even that one instruction will not move, if another instruction is still in the instruction issue stage. This is an example of sacrificing new features to maintain a very high clock rate.

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.

This architecture relies heavily on the compilers to schedule instructions to best utilize their functional units and prevent data dependencies from stalling the pipeline.

SUN UltraSPARC

The UltraSPARC-1[UltraSPARC] is more a RISC processor than a Post-RISC (Figure 8) processor, but it does have a number of FISC features in common with our generic Post-RISC CPU.

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.

Between the instruction fetch and dispatch units, the UltraSPARC has a 12 instruction queue. This queue allows fetching to continue while the execution pipeline is stalled due to a dependency or cache miss. Conversely, the execution units can continue to work even when the instruction fetch has a cache miss. Thus, the UltraSPARC's pipeline can be thought of as two connected pipelines with a buffer between them. If either pipeline stalls, the other can exploit the buffer.

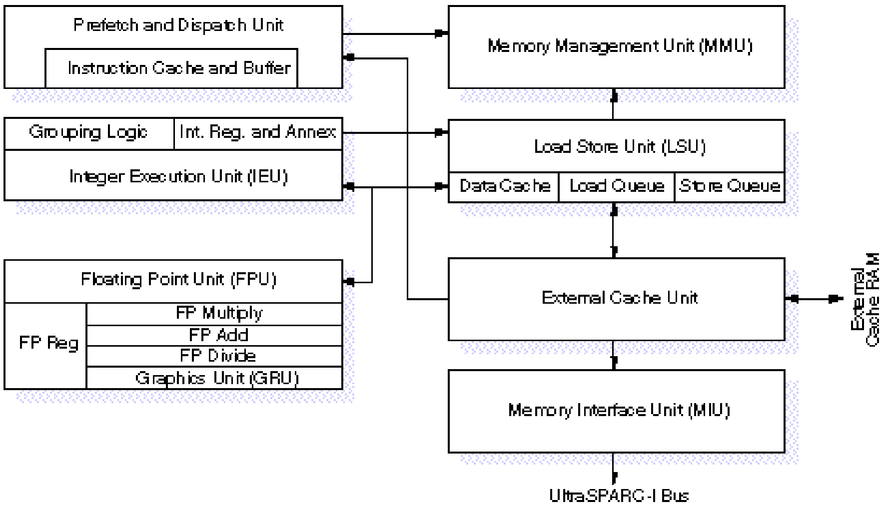

Figure 8: UltraSPARC-1 Architecture

Instructions are issued in-order and retired in-order *except* for several multicycle 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.

Since the UltraSPARC uses register windows, register renaming is extremely difficult to implement efficiently. Without register renaming, there is little to be gained from out-of-order execution. This is essentially the limiting factor in introducing Post-RISC features into the SPARC family of microprocessors.

Even though the UltraSPARC lacks out-of-order execution, it possesses many FISC features. The UltraSPARC's goal was to provide a high performance processor for desktop Unix systems. To this end, the UltraSPARC features support for very fast context switches, variable size pages, fast trap and interrupt handling, a block memory move instruction, instructions for quickly doing motion estimation (used for real-time MPEG encoding), and a full set of graphics manipulation instructions (the VIS instruction set).

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, Figure 9, is four-way superscalar and instructions are fetched from an on-chip cache in four-instruction segments. Instructions are predecoded 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). Later branch predictions override flow prediction.

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. Decoded instructions can indicate the success of predicted branches so corrections can be initiated at this stage. 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). Instructions can be dispatched speculatively. Unique to the PowerPC are eight condition-register rename buffers which are available to maintain condition codes for speculatively executed paths.

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 actual 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.

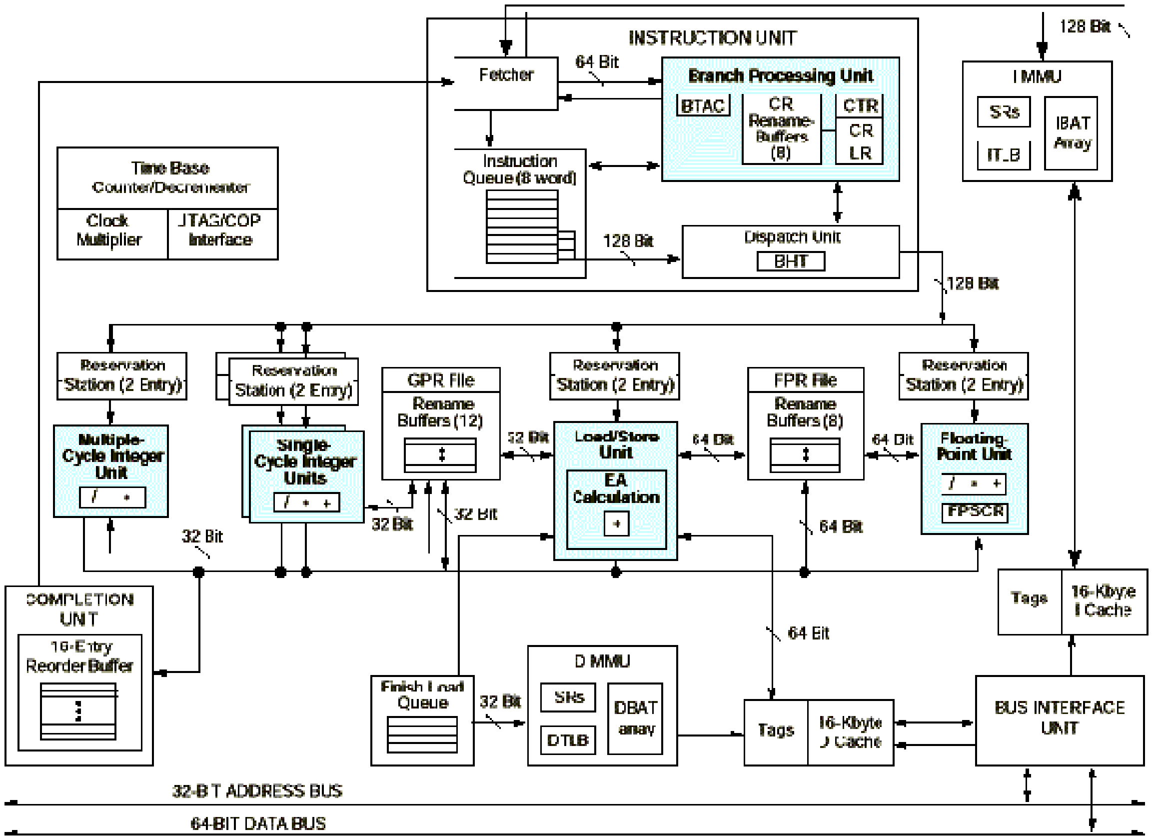

Figure 9: PowerPC 604 Architecture

A unique FISC feature of the PowerPC architecture is the "branch-on-counter" instruction which is used to efficiently implement counter-based loops. The loop Count Register is part of branch prediction, thus "branch-on-counter" instructions are effectively unconditional branches.

MIPS R10000

The MIPS R10000 [R10000] is an exceptionally 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 a fully 4-way superscalar architecture containing a 64k split 2-way cache on-chip. It can fetch, decode (with branch prediction), and schedule up to four instructions per cycle. Each instruction is predecoded 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 removed from the queue when the required inputs and execution unit are available. The general design of the R10000 CPU architecture and pipeline are detailed in Figures 10 and 11.

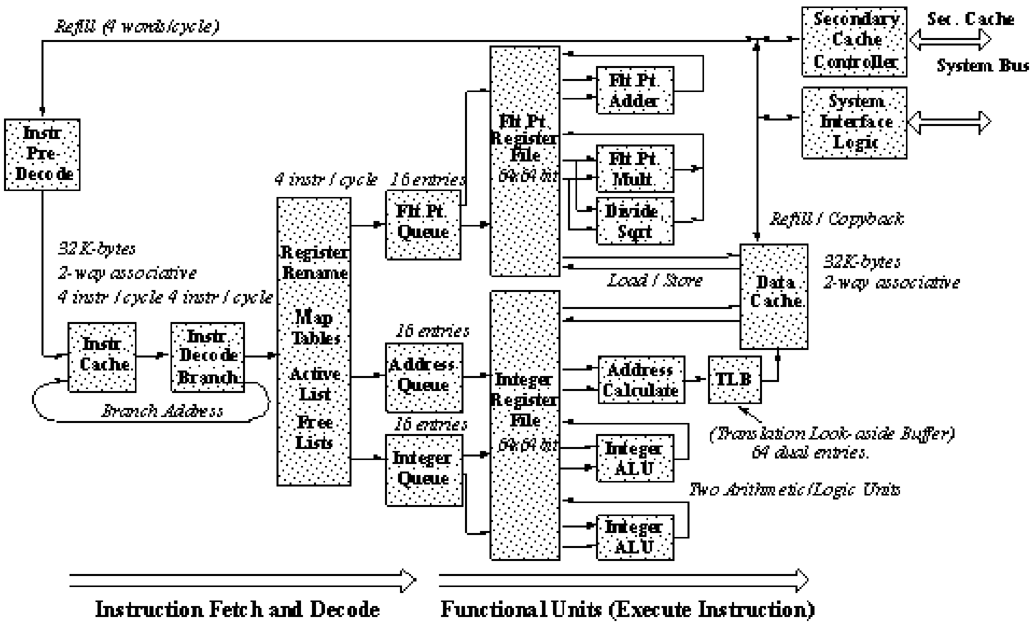

Figure 10: MIPS R10000 Architecture

Note that the R10000 does not implement Post-RISC flow prediction. Branch prediction uses the common 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.

Several FISC features have been introduced in the MIPS architecture. Their implementation of a "shadow map" and a "Branch Resume Cache" are of particular interest. Prior to each branch, the R10000 saves the state of the rename register map to a shadow map. In addition, the instructions following the branch (in the fetch stage) are stored to a Branch Resume Cache. If mispredicted, the rename register map table is copied from the shadow map, restoring its contents to the state following the branch delay slot. Furthermore, if the branch was mispredicted as taken, the next four instructions are fetched directly from the Branch Resume Cache. If this cache does not contain these instructions, the traditional extra cycle is required to fetch these instruction from the I-cache.

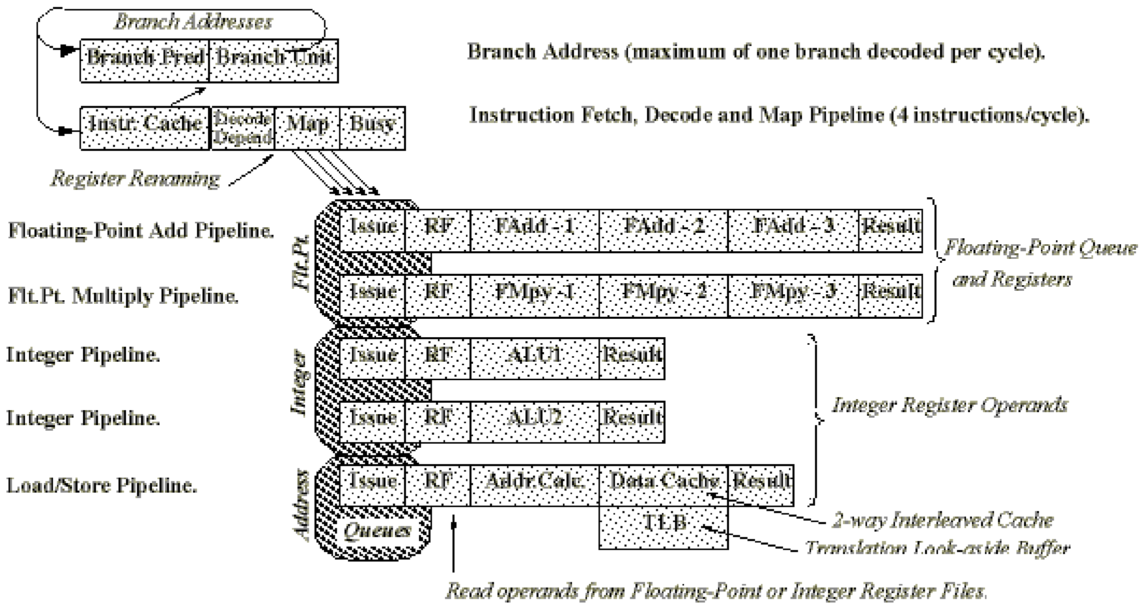

Figure 11: MIPS R10000 Pipeline

MIPS claims that the R10000 runs code compiled for the R4000 "like optimized code". Aggressive use of dynamic execution allows this Post-RISC CPU achieve excellent performance without necessarily depending heavily upon re-engineering compilers and hand-tuned assembly code.

HP PA-8000

The HP PA-8000 (Figure 12) is a dramatic departure from the HP PA-7000 processor family primarily because of its Post-RISC features. The heart of the PA-8000 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. An excellent overview of the PA-8000 is available in [PA8000].

In the PA-8000, instructions are fetched from a large off-chip synchronous SRAM cache. Instructions are fetched in four-instruction sets. 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 BTAC predicts the flow of the four instruction set rather than individually predicting branches. 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.

The "sort" unit decodes and inserts four instructions into the appropriate IRB, 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).

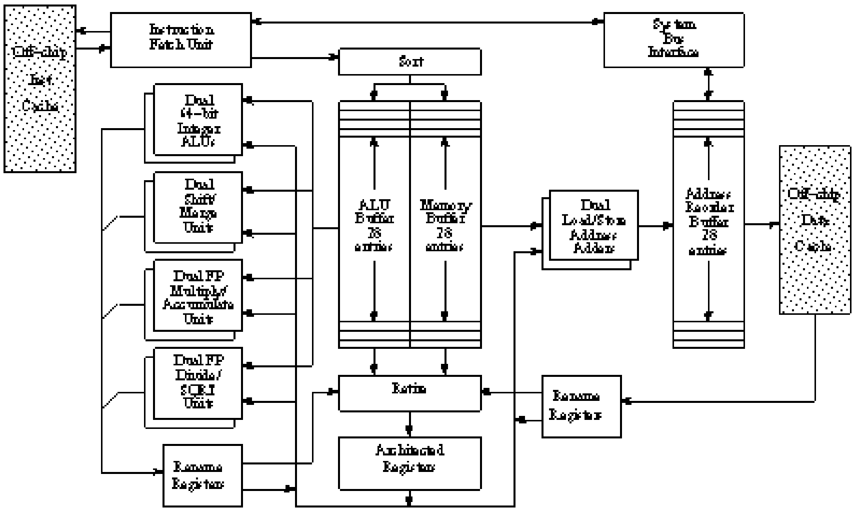

Figure 12: HP PA-8000 Architecture

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.

Because all the units (except the Divide/SQRT) can accept an instruction on every cycle and there are two of each type of unit, any combination of two arithmetic instructions can be started every cycle (except for Divide/SQRT).

There are two adder units which 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 prefetches simultaneously and these memory operations can be completed in any order from the off-chip data cache or from main memory. Thus, up to 10 data cache misses can be outstanding at one time.

One unique feature of the PA-8000 is its ability to turn off the dynamic branch prediction for portions of code. This can be done on a TLB boundary. This ability is quite useful because it allows shared libraries, the operating system, or even portions of user code to use profile information and static prediction, saving the limited number of BHT entries for the unprofiled code.

Intel P6

The Intel P6[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 (Figure 13) 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.

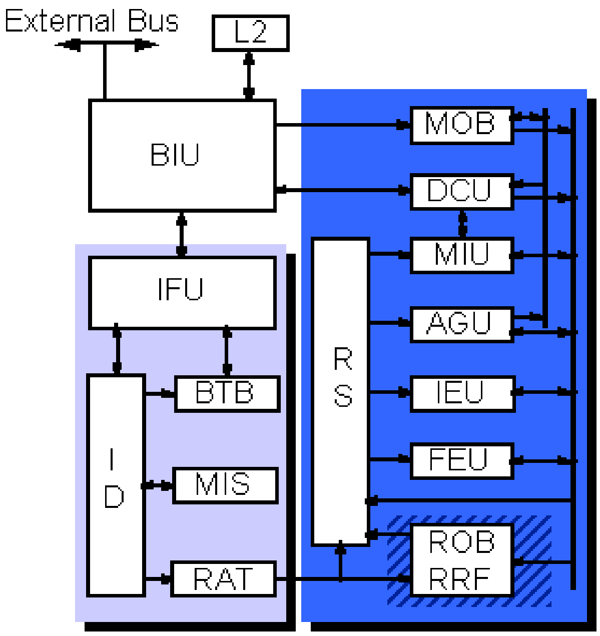

Figure 13: Intel P6 Architecture

The memory hierarchy uses the traditional two-level cache scheme. The level 1 on-chip cache is 16K separated into 8K instruction and 8K data. The L2 cache (Figure 14) is off-chip and comes in 256K and 512K sizes. One very interesting feature is that the L2 cache is a separate chip, but is in the same package as the CPU.

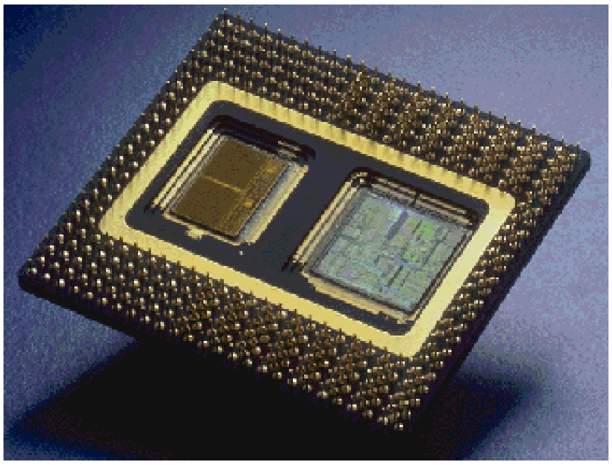

Figure 14: Intel P6 Multichip Module

Including the L2 cache in the same package allows for the interface between the processor and the L2 cache to run at processor clock speed. This cache traffic all occurs on a "back-side bus", allowing all of the front-side bus bandwidth to be used for memory and I/O accesses.

Instructions go through the P6 in three stages (Figure 15): 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.

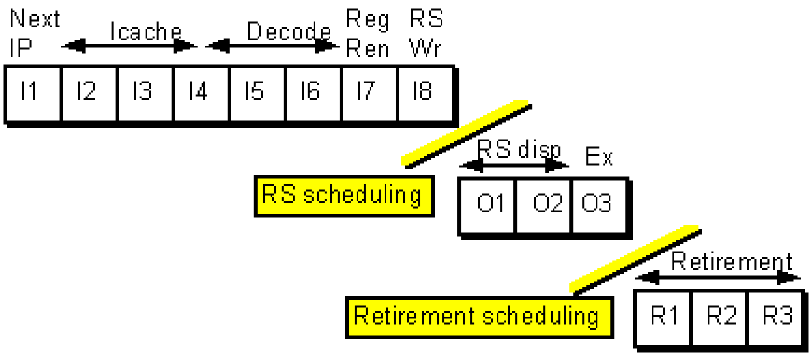

Figure 15: Intel P6 Pipeline

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 which 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, as Table 1 shows, the P6's 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 which 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 a 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. (3) Finally, Digital chose to avoid the complexity of out-of-order scheduling in order to maximize clock speed. The penalty of unused cycles in functional units is mitigated by keeping the cycles small. 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

[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.

[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/