Superscalar and advanced architectural features of PowerPC and Pentium family

Chan Kit Wai and Somasundaram Meiyappan

1. Introduction

Superscalar processors are processors that can issue and execute more than one instruction in-parallel through use of more than one execution unit taking an in-order program as input and also to produce the output in the same order. The PowerPC/Power and Pentium micro-processor families are the popular super-scalar processors for the desktop. Their key super-scalar features are discussed in the rest of the report. Other features like branch prediction that help the processor to make maximum use of the available ILP are also discussed.

2. Superscalar features in Pentium and PowerPC

Superscalar processors have multiple execution units. This enables them to execute more than one instruction at any clock cycle. The following table lists the super-scalar features like the issue width, retirement width and number of execution units in some processors of both Pentium and PowerPC families.

	PowerPC 620	PowerPC 750[2]	PowerPC 970[4]	Pentium III	Pentium 4[5]
#issue per clock	4	2	8	3	3
#retired per clock	4	2	5[3]	3	3
#Integer Units	2+1	2	2	2	5
#Floating Point Units	1	1	2	2	1
#Branch Units	1	1	1	-	-
#Load/Store Units	1	1	2	1	2 (1+1)
#In-flight instructions	App. 37	App. 16	>200	unspecified	126

Note:
1. All values are the maximum
2. PowerPC 750 has an additional System Register Unit
3. Five only if one of them is a branch instruction
4. PowerPC 970 has 4 other execution units if AltiVec is used and it also has a conditional register unit
5. Pentium 4 has a special floating point move unit. It has a load unit and a store unit. Branching is computed by the integer unit.

Super-scalar processors are primarily defined by the number of instructions that can be issued to the next stage of execution. Though normally the number of instructions that can be retired is the same as the number of instructions issued, it is not always the same. PowerPC 970 is an example

for that. Since the strength of a chain is always determined by that of the weakest link, PowerPC 970 should be regarded as a 5-way superscalar processor.

Please note that all the processors above have lots of available parallelism in-terms of number of execution units than the number of instructions that can be issued. Systems are designed this way for two purposes:

1. Have enough execution units for peak execution bandwidth
2. To execute any available instruction when slower instructions are being executed.

When issuing 3 instructions in a cycle, certain selection criteria must be met for the instructions to be chosen to be issued in that cycle. For example, Pentium 4 can issue 3 instructions only when it can find a combination of 1 load, 1 store and 1 manipulative non-memory operation. PowerPC 620 can issue 4 instructions only if one of them is a branch. Such restrictions, we think, are laid to give a preference to certain kind of high latency instructions and also taking into account the percentage of a type of instructions that the processor normally encounters. It is also based on the design of other features like the reservation stations or instruction queues. Timing considerations are taken into account when making these decisions.

In modern micro-processors, the number of instructions in the ISA keeps increasing. This makes instruction decoding a bit complex. So, instructions are pre-decoded and stored in a decoded format. Some processors like Pentium 4 go a step ahead and replace the traditional instruction cache with a cache that stores decoded instructions along a path (trace) that is predicted using advanced branch prediction techniques. To populate such a cache, aggressive branch prediction is used to speculatively select a trace dynamically.

Superscalar processors look for parallelism in the program and execute instructions that can be executed in parallel. One of the essential steps in identifying parallelism is to identify dependencies among instructions and possibly resolving them by using other techniques. Such activities take time, especially when the number of instructions that are analysed for parallelism grows higher. Also, the weakest link or the slowest part of the pipeline is the fetching of instructions from the memory. So, every fetch should be made use of effectively. Instruction caches help us reduce the latency involved with fetching. And often the number of instructions issued is less than the number of instructions that can be fetched from the cache. So, to continuously engage the decoder and to decouple the issue, the dispatch of instructions to the execution unit and to have a place where instructions could wait for its predecessors to complete, buffers are employed. One such technique is called shelving and in processors like Pentium 4 this buffer is the same as ROB (Re-Order Buffer). The ROB is a buffer that tracks the status of an instruction from the moment resources are allocated for the instruction till the instruction is retired. And processors like PowerPC which practice aggressive out-of-order execution, employ a slightly different kind of a buffer and is called reservation stations. PowerPC processors had been having reservations stations for each of the execution unit and instructions are dispatched and they wait in the reservation stations of that execution unit.

2.1. Re-Order Buffer

Re-order buffers are essential and as the name suggests it is used to re-order instructions that have been executed out-of-order. It is used so that the instructions can be retired in the same sequence as they were received by the processors eventhough they were executed out-of-order. In some ways, it is similar to the completion buffer that PowerPC has into which all instructions that are

completed are present. And the retirement unit picks the oldest instructions in the group and retires them.

Re-order buffers are also suitable to unify other buffers like the shelving buffers into one and thus can be used to track the execution status of an instruction in the processor with just one buffer. That is exactly what the P6 micro-architecture and the NetBurst micro-architecture does. In the P6 micro-architecture, the rename registers are also part of the ROB. This approach helps to reduce hardware cost involved in indexing several buffers.

2.2. Reservation Station

Reservation stations are used as waiting slots for instructions. They wait for not only for the execution unit to be ready; but also for the inputs it needs to be available before it can be executed. These slots not only have the internal op-code and tags to the registers; but sometimes the value of the operands themselves. This means that if the operands for an instruction are not ready when the instruction was dispatched into the reservation station, the instruction should snoop the "forwarding" bus to see if the operands are ready so that it can pick up the operand(s) as and when they are available.

In PowerPC 750, all execution units have only one entry in the reservation station except the load-store unit. So, each execution unit had a dedicated reservation station. But PowerPC 970 has queues that are shared by two execution units and each queue holds different number of entries depending on the execution units that it is feeding. PowerPC 620 also has dedicated reservation stations for each execution unit; but there can be more than one entry in the reservation stations. Execution units which take longer to produce the result have more reservation stations. It is done this way to reduce the number of dispatch stalls that would happen because of saturation of the reservation station of a particular type of EU.

Pentium 4 processors also have queues where the instructions wait to be scheduled. But it has only two queues viz., a memory operations queue and an arithmetic instruction queue. Instructions wait there to be scheduled; but the difference between the queues in Pentium 4 and the reservation station in PowerPC 750 is that the reservation stations in PowerPC hold instructions that have already been dispatched; but the instructions in Pentium 4's queues are still not dispatched. So, in a way, the queue is actually used as a shelving space.

2.3. Dynamic Instruction scheduling

When multiple instructions are ready to be executed at the same time, different processors use different ways to pick the instructions from the ready pool and pass them to the execution unit. In PowerPC 750, each execution unit has only one entry in the reservation station. So, scheduling has become little easier. The instruction gets scheduled as soon as it has all operands that are needed and as soon as the EU is available.

However, in Pentium 4 and PowerPC 970, it would be a bit different as they have more than instruction in their queues and in a position where one queue has to feed more than one execution unit.

In Pentium 4, schedulers operate on a queue in a first-in first-out (FIFO) order. The first instruction in the queue should be dispatched and scheduled first before other instructions in the queue can be scheduled. This might seem like in-order execution, which in fact is correct; but the presence of two different queues and the fact that instructions from one queue can be scheduled out-of-order with respect to instructions from the other queue. Thus, Pentium achieves out-of-order execution. PowerPC 970 schedules instructions also in the FIFO order. But it is more aggressively out-of-order as it has more queues per execution unit than Pentium4. And thus there is more opportunity for out-of-order execution.

Instruction scheduling not necessarily starts after the reservation stations, but infact, it starts at the decode and issue stages of the pipeline and more interestingly, it starts when the program is compiled. Before discussing about optimisations that a compiler can do when compiling for a superscalar processor, we would like to discuss what happens when resources are running low in the processor – stalls and then understand how a compiler can schedule instructions to prevent these stalls.

2.4. Stalls in a superscalar pipeline

Stalls that are normally seen in a scalar pipelined processor are caused by a memory load, by a branch mis-prediction or by an exception. In superscalar processors, the reasons can be much more. Apart from these stalls, other stalls in superscalar processors can be classified as issue stalls or dispatch stalls.

The type of stalls that is typical among superscalar processors are

- Unavailability of free entries in the reservation stations. If reservation stations of a particular execution unit are full, then a new instruction to be scheduled would have no place to go. In such cases, most processors just stall dispatch of the instruction. eg.: PowerPC 620
- Pentium4 allocates all resources needed by an instruction to execute before it puts the instruction in the queue. The resources include register ports, ROB entry, rename registers, etc… Even if one of the resources is not available, it stalls the pipeline. PowerPC does the same as well.
- Unavailability of free execution units. This could either be a cause for the absence of reservation stations or even if there are free entries in the reservation station, we might not have free port to send an instruction to the reservation station as one has just been issued in this cycle. Though one might argue that this is not really a stall, please note that in most processors, if one instruction in a queue cannot be issued/dispatched, the other instructions behind it in the queue cannot be issued/dispatched as well
- Another reason would be that the instruction is simply not ready to be executed as not all operands are available.
- Or it could be a really intentional stall cause by specific instructions in the instruction set that are to be used for synchronisation purposes.

These stalls cause a program to potentially underutilise the available ILP in the processor. This is where compiler optimisations can help the program to maximise the benefit of having ILP.

2.5. Static scheduling by compilers

Compilers have improved in their ability to analyse the program and produce an instruction stream that is highly optimised against certain criterions. Though vectorising or parallelising compilers that make use of SIMD operations are hard to see, there are compilers which would do that. But, we would restrict the scope of this discussion to just optimising for the superscalar features.

PowerPC 750 can dispatch only two instructions in a single clock cycle. But in-order to dispatch two instructions, the instructions must not need the same execution unit as it would lead to a dispatch stall otherwise. Since the window for dispatching out-of-order is just two consecutive instructions, the compiler must take care that two consecutive instructions do not need the same execution unit.

PowerPC 750, just as earlier PowerPC processors, places lots of importance in resolving a conditional branch early. So, to help the processor in resolving the branches early instead of predicting it, we instructions that compute the condition can be scheduled as early as possible before the branch instruction. This would ensure that the branch can be resolved with the result of the condition that is already available. Scheduling loads and stores by the compiler also plays a major role in the optimisation. Though always difficult, a general rule would be that the producer of a value be placed as ahead of its first consumer as possible.

Compilation based on "machine description" has proven to be extremely beneficial and is known to produce extremely optimised code. Information like how many execution units of each type are available and what is the minimum number of cycles that a particular operation can take are brought into the instruction scheduling phase of the compilation. Advances in VLIW compilers can only help when compiling for super-scalar processors. The cycle based list scheduler in the Trimaran research compiler[8] for example schedules instructions just enough as to avoid a stall when executing an instruction that was dependant on the other. Such schedulers could help boost performance of programs compiled for super-scalar processors as well.

Another popular concept among compilers that support predicated execution is software pipelining. Software pipelining is a concept that is closely associated with loop unrolling and in which more than one iteration of the loop is computed at the same time just like pipelining that is employed in the micro-processor itself. But for software pipelining to be really effective for super-scalar processors, the compiler definitely needs the machine description.

Please note that all optimisations that involve the use of machine descriptions are optimised only for that target. Though the code should work on all future successors of that superscalar target, the generated code is not optimised for the successors.

3. Advanced features that improve performance

There have always been two traditional factors that affect the performance of a program in a processor. These are the ways processors handle control and data dependencies. Though these factors should be considered even in non-superscalar pipelined processors, it becomes more important to handle control and data dependencies better in super-scalar processors in which a bubble in the early part of the pipeline is lots of ILP wasted and thus a performance loss. The

following concepts are widely used in various super-scalar processors to reduce the loss in performance.

3.1. Branch prediction

In super-scalar pipelined processors, branches and jumps mostly change the sequential flow of execution. But at the same time, conditional branches might not change the sequential flow. This means that the processor should either wait until it resolves the condition or assume that the branch would later be resolved as either taken or not-taken before continuing to fetch the instructions that would follow the branch. Since waiting for the branch to be resolved before fetching the next instructions introduce bubbles in the pipeline, modern processors predict the result of the conditional branch to keep the pipeline busy. Different processors employ different methods to predict the branches.

Modern processors like PowerPC and Pentium employ some kind of static prediction. Almost all processors use PC relative branching and thus have the value of offsets in their branch instruction. The static prediction used in most processors is such that the backward branches (if there are negative offsets in the instruction) are predicted as taken. The reason behind this is that backward branches are normally encountered in loops which are always taken except once when the loops terminal condition is met.

Some processors predict the forward conditional branches also as taken whereas some processors statically predict them as not-taken. Each approach has its own pros and cons. Instructions from the fall-through (not-taken) block of a conditional branch can potentially be in the instruction cache because the instructions in the fall-through block would be sequentially after the branch in the program. So, even if the prediction that the branch would be taken was later found to be incorrect, the probability of an instruction fetch from the not-taken path being a cache hit is higher because these instructions might be in the same cache line as the branch instruction. So, the mis-prediction penalty might be lesser if the forward branch is also statically predicted as taken. But the disadvantage is that even if the branch is predicted as taken (and even if the prediction is correct), the instructions from the taken path would have to be fetched from the memory hierarchy and the probability that the taken path would also be in the cache is less. So, if there is a cache miss, we might have some penalty even though the prediction is correct unless the we have BTIC (Branch target instruction cache). But if the forward conditional branch is predicted as not-taken, there might not be any penalty if the prediction is correct. So, different programs perform differently in these two approaches. This means that for good performance, the compiler/programmer must arrange the program correctly to experience low penalty.

With compilers able to perform profiling on a sample input set on the programs, the compilers can also give a good hint on the likely resolution of a branch to the processor. So, some instruction sets introduced hints in the instruction encoding of a conditional branch. PowerPC 620 is one such processor which enables the compiler to provide hints on branches. Even if the programmer is not equipped with profiling tools, the programmer if he/she understands the compiler and the processor can write code such that the code that would most frequently get executed is in the fall-through or the taken path depending on the processor's static prediction approach or set the branch hints appropriately.

Since the static branch prediction techniques are based on heuristics, performance would vary with programs. Dynamic branch prediction techniques help the processor to make predictions on

branches by following the program's branching behaviour. The following table lists the dynamic branch prediction techniques that are used in some PowerPC and Pentium processors.

From the figure below, it can be noted that the two main dynamic prediction techniques favoured by the processor designers are the 1-bit or 2-bit saturating counter based predictor and derivates of the two level adaptive algorithm proposed by Yeh and Patt.

The 1-bit predictor records the history of the last occurrence of a branch and the future branches are predicted using this value. When a conditional branch changes direction after the first time, there is bound to be a mis-prediction. When the branch is resolved, the history bit of the branch would be updated and this would force the predictor to change its prediction the next time this branch is encountered. The main advantage of this over static branch prediction schemes is that it follows the branches' last behaviour rather than assuming based on some heuristics. This seems to be increasing the branch prediction success rate. The 2-bit saturating counter based branch predictor which is also a single table based predictor increases the success rate even more. This is used to classify branches as taken, strongly taken, not-taken and strongly not-taken. So, the granularity of prediction now becomes higher from 2 level in the 1-bit predictor to 4-level in the 2-bit predictor. Thus, a frequently not-taken branch which is just once occasionally taken is always predicted as not-taken by the 2-bit predictor. But a 1-bit predictor would present a mis-prediction every time the conditional branch resolves in a way opposite to its last occurrence.

	PowerPC 620	PowerPC 750	PowerPC 970	Pentium 4	Pentium-M
Static Branch Prediction	Forward branches are not-taken and backward branches are taken	Same as PowerPC 620	Same as PowerPC 620	Same as PowerPC 620	Same as PowerPC 620
Branch hints from compiler	No	Yes. Software can also override the hardware branch prediction results with a hint.	Yes. Software can also override the hardware branch prediction results with this hint.	Yes. The support is available only for certain branch instructions.	Yes. The support is available only for certain branch instructions.
Dynamic Branch Prediction	1. 256 entry 2-way associative BTAC 2. 2048 direct mapped Branch History Table	1. 64-entry two instruction per entry BTIC 2. 512 entry 2-bit branch history table	1. 16K entry 1-bit predictor 2. 16K entry 1-bit global predictor based on group execution sequence, 3. 16K entry 1-bit selector table	1. Branch Target Buffer (one for speculative fetch) and the other one fetching instructions from the trace cache	1. Global Branch target buffer 2. Counting loop predictor 3. Indirect branch predictor

Another approach commonly used is a cache of all branch instruction's target addresses as in BTAC (and/or a few instructions along the taken path as in the case of BTIC). Predictions are made depending on the whether we have a hit or a miss in the cache. Please note that there is a higher cost associated towards employing such a cache. So, we can have only limited number of entries in the cache. So, processors do not normally just use BTAC/BTIC for prediction; but they

use this approach and another more accurate (but slightly slower) algorithm like the 2-bit predictor which has lesser associated hardware cost. Power PC 620 and PowerPC 750 are examples of processors that employ such methods. The decision as to which prediction to consider is normally based on a static rule. Normally, since BTAC/BTIC is faster, it is used in the fetch/prefetch stage and the regular 2-bit predictor is used at a later stage in the pipeline.

Yeh and Patt's paper [3] on a two level branch prediction scheme is an attempt to increase the accuracy of prediction based on a pattern of history with the predictor giving different predictions for different history patterns. The first table is a table/register that is indexed by the branch address (or just a part of the branch address) and each entry holds the true result of the last k occurrences of a conditional branch in a k-bit vector. This vector/pattern is used to reference another table of tables and a table (called the pattern table) is chosen based on the branch address once again. In one variant of the implementation, each branch would have its own pattern table and 2^k entries (all possible patterns with a k-bit vector). So, a prediction is based on the value that this present in the pattern table entry chosen by the k-bit vector of the branch in the history table. While this is found to give as much as 96% success rate for SPECInt programs, the success rate also depends on the implementation. If multiple branch instructions are made to share a history register or if the same pattern table is shared by multiple branches, then there would lots of interference in the prediction of a branch from other branches' history in the group of branches that share the same history register or pattern table. As it can be seen from the above table, the P6 micro-architecture used in Penitum Pro, Penitum II and Pentium III uses this algorithm and is implemented such that 4 branches share the same history register. This causes some amount of branch prediction interference. Looking at the declared success rate of Pentium 4's branch prediction scheme, we speculate that it would have used more hardware to reduce the interference.

PowerPC 970 (desktop version of IBM Power 4 processor) processor uses a similar two level approach in its global selector table algorithm. But the history is not based on the branches; but is based on the sequence of groups of instructions that lead to the execution of this group of instructions which has the branch. The actual prediction is stored in a table similar to the pattern history table in Yeh and Patt's algorithm and this is a hashed table and the key is obtained by performing an exclusive or with the branch address on the history register. One of the interesting points to note about 970's algorithm, it had moved from a 2-bit branch predictor used in earlier PowerPC micro-processors to a 1-bit branch predictor; but the difference is this table can hold more number of branch entries. Though PowerPC uses two dynamic branch prediction techniques like its predecessors, it continues to see which algorithm performs better for each branch using a selector table. So, the decision on which prediction scheme to use is not static as in the other processors; but dynamic. It would be surprising to know that when condition register is available for early branch resolution, PowerPC 750 resolves the conditional branch instead of predicting the direction and target of the branch. But PowerPC 970 always uses branch prediction even if the condition register is ready with the value. This might be because of the unusually deeper pipeline that 970 has for a RISC processor.

Pentium-M, the first processor from Intel designed with thermal and power constraints from the beginning; but is also designed for good performance. Branch prediction assumes more significance in mobile processors where a mis-prediction results in loss in performance; not just in lost processor cycles but lost power in all the switching involved in those cycles. If the branch prediction is more accurate, not only we can achieve a boost in performance; but save some power. In an attempt to achieve that, Pentium-M uses a counting loop predictor and another indirect branch predictor to assist in predicting all branches in a fixed strength loop and to predict targets even for the 'switch.. case' construct in C/C++ respectively[1]. Though there is an

associated hardware cost with this hardware, the boost in performance saves power. Please note that the main advantage of the counting loop predictor over Yeh and Patt's two-level adaptive algorithm is that it can support loops of high iteration count.

3.2. Register renaming and Register Alias Tables

Having looked the branch prediction techniques used in Pentium and PowerPC which try to speculatively resolve control dependencies, this section would deal with a concept to deal with data dependencies (false data dependencies to be precise).

False data dependencies arise because of insufficiency of registers in the system and because of compilers' tendency to use as few registers as possible even if there is no dearth for registers. Though this is not a big problem in scalar pipelined processors, but in superscalar processors, false data dependencies restrict the parallelism that can be realised from a program running on a superscalar processor. Register renaming attempts to eliminate the effect of false dependency by providing lots of shadow registers that can be used instead. Pentium 4 provides 128 shadow registers for its sets of 5 general purpose architectural registers. Every instruction (macro or micro op) that produces a result in a register is allocated a new register for the destination (or destination registers depending on the architecture). This requires a table (called a Register Alias Table) to track the allocation of shadow registers to the various instructions. It is required so that subsequent instructions, which consume the value produced by the instruction, whose destination was just renamed, would know where to fetch the data from.

As mentioned above, RAT holds the architectural register to the last rename register allocated. When this combines with branch prediction where speculation is also involved could potentially make the design of RAT more complex. But extent of complexity depends upon the mis-prediction recovery scheme adopted by the processor.

The P6 micro-architecture does not maintain a separate Rename register file; but the rename registers are a part of the Re-Order Buffer (ROB). This means there is also a separate architectural register file to which the values would be copied from the ROB entry that is to be retired. The P6 micro-architecture fetches instructions from the correct path after resolving a branch that was mis-predicted; but those instructions are not dispatched until all the instructions younger than and including the branch instruction are retired. This architecture increases the penalty on a mis-prediction; but reduces the need to maintain a stack of the latest RAT in every speculatively executed basic block. Instead, the RAT can be made to point to the architectural register file and dispatch of instructions from the next basic block can continue after that. Otherwise, we would need shadow RATs to store the state of RAT on every branch that is encountered in a speculative path.

Pentium 4, however, uses two separate RAT and the rename registers are decoupled from the ROB. One is used for tracking the speculative rename register allocations in the front-end during dispatching and the other is used to maintain the latest architectural register when instructions are retired. So, there is no need for a separate architectural register file and instead everything is merged into a huge register file with 128 registers. PowerPC 750 on the hand uses a smaller and a separate rename register file of just 6 rename registers. This might be sufficient for PowerPC because of two reasons. First, PowerPC has 32 general purpose registers exposed to the user for use in programs; but Pentium 4 having started as a traditional CISC has only 5 GPRs. This means that probability of seeing a false data dependency when code is generated from a compiler that

does superscalar optimisations is less in the case of a PowerPC. Second, Pentium 4 reduces the CISC instructions that it sees into smaller RISC like micro-ops. These micro-ops would also need registers to work and store their result. This means that internally Pentium 4 needs more registers. We think that these are the reasons behind this difference.

It would be interesting to look at an architecture like Pentium's, in which a common condition register would be changed by almost all instructions. In out-of-order execution, the integrity of the condition register must also be maintained. There might be number of ways to do this; but we speculate that one of the following implementations is used in the processors.

1. Use a condition register in every entry in ROB. This is similar to P6's concept of having rename registers in every entry in ROB. Thus, we also rename condition registers.
2. Another approach would be to compute the flags or condition register when the processor is about to retire the instruction and directly update the architectural conditional register. While this approach would save some hardware space, it would introduce additional timing burden in a pipeline stage and in high frequency designs, this stage could even spill out as a separate stage in the pipe.

3.3. Reordering memory access

In an aggressive out-of-order execution core, all instructions are capable of being executed in out-of-order. While it is easier to identify and resolve false data dependencies in the hardware, it is difficult to identify early if two memory accesses are made to the same address. Since memory loads have a longer latency than memory stores (because memory stores first made to a write-buffer normally before actually writing to the memory), some high performance processors like PowerPC 750 schedules a younger memory load before an older store hoping that the address they access would be different. Anyways, PowerPC 750 also maintains a load queue and a store queue that maintains a list of all stores and loads that have been executed; but not yet retired. So, all stores are compared against the loads in the load queue and if any load that is younger than the store is in the load queue and has accessed the same memory location, then all instructions younger than this store instruction is flushed and re-executed. Since such dependencies that need memory dis-ambiguation are lesser in probability, such re-ordering helps a super-scalar machine to improve performance.

4. Conclusion

Superscalar processors employ more aggressive techniques like branch prediction and enable speculative fetching of instruction streams far beyond the current sequence of instructions. This speculative fetch needs to be predicted correctly for future deeply pipelined high frequency designs to post good performance numbers. Using aggressive techniques not only improve performance; but could also help reduce power consumption. Since power consumption would definitely become a major issue in both the high performance servers and in the mobile notebooks, designers should begin to look for ways to improve efficiency and performance. One way that the designers would potentially look forward in future super-scalar processors is the concept of value prediction that is used to predict or speculate the value so that true data dependencies can also be conquered with a considerable success rate. This would further improve performance. With explicitly parallel processors like Itanium, the compilers would have increasing role to play in defining the performance of a processor.

6. References

1. The Intel Penitum-M Processor: Microarchitecture and Performance, from Intel Technology Journal, Volume 07, Issue 02, May 21, 2003
2. The micro-architecture of the Pentium 4 processor, Intel Technology Journal
3. Alternative Implementations of Two-Level Adaptive Branch Prediction, Tse-Yu Yeh and Yale N. Patt, ACM, 1992
4. IBM PowerPC 750FX RISC micro-processor
5. Power 4 micro-architecture from http://www.ibm.com
6. Modern Processor Design: Fundamentals of Superscalar processors by Shen and Lipasti
7. Advanced Computer Architecture bye Fountain and Kucksuk
8. Trimaran compiler from http://www.trimaran.org