ILP Execution

Lecture #11: Wednesday, 3 May 2000
Lecturer: Ben Serebrin
Scribe: Dean Liu

In this lecture, we discussed four papers:

1. D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo, "**The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling**," *IBM J. Research and Development 11:1*, January 1967.

2. S. Palacharla, N. P. Jouppi, and J. E. Smith, "**Complexity-Effective Superscalar Processors**," in *Proceedings of the 24th International Symposium on Computer Architecture*, June 1997.

3. J. E. Smith and A. R. Pleszkun, "**Implementation of Precise Interrupts in Pipelined Processors**," in *Proceedings of the 12th International Symposium on Computer Architecture*, June 1985.

4. L. Gwennap, "**VLIW: The Wave of the Future? Processor Design Style Could Be Faster, Cheaper Than RISC**," *Microprocessor Report, Vol. 8 No.2*, February 14, 1994.

1 IBM System/360 Model 91 [1]

The goal of the design was to achieve a 10x performance improvement over previous System/360 architecture. However, the improvement in circuit technology could only bring a four fold performance gain. So, they needed other advances in the microarchitecture to capture the rest of the gain needed to meet the performance goal.

The technology of choice for the 360/91 was ASLT which were discrete transistors. The design was probably on the order of 10,000-100,000 transistors using hybred integrated circuits with soldered logic module on the PCB.

The target clock frequency was 16 MHz. In comparison, the memory access latency was only a few cycles. As a result, there was no need to have caches. On a side note, the Model 85 was the first machine with a cache. It had a simpler pipeline and had the same performance as the Model 91.

With the aggressive microarchitecture, the IBM 360/91 achieved 0.5 IPC when contemporary machines of that time were about 0.1 IPC. Note the instruction words had

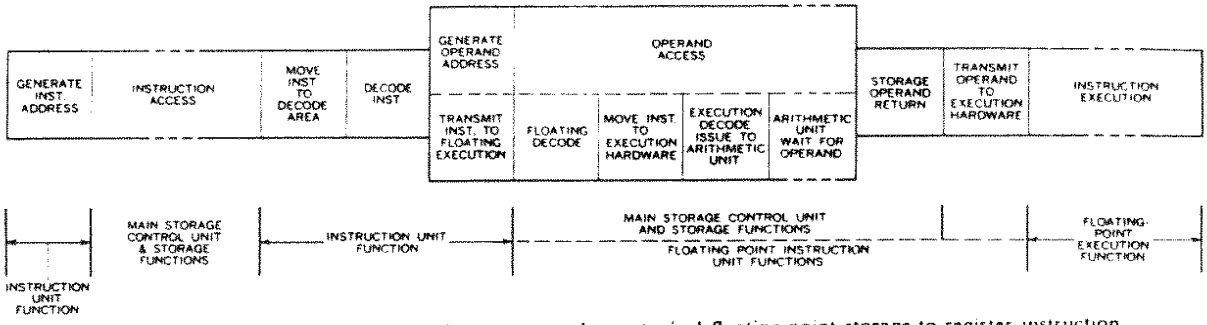

Figure 3 CPU "assembly-line stations required to accommodate a typical floating-point storage-to-register instruction.

Figure 1: Reproduction of Fig 3. in *The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling.*

variable lengths, so each machine instruction was equivalent to about three RISC instructions.

Figure 1 is a reproduction of the third figure in the paper which shows the pipeline diagram. First of all, the Model 91 was a memory-to-memory machine. We can also see that this machine had a long pipeline and experiences the standard pipeline problems such as control and data dependencies, and large branch penalty. Other microarchitecture features included 2-stage execute, some out of order execution, single instruction issue, and the inclusion of a loop buffer. There were forwarding paths to the reservation stations because out of order execution might cause registers to be reallocated. As a result, the instruction could not write back to the register. The 360 also only supported imprecise interrupts, except in some very limited cases.

The 360/91 employed prefetching fall through as the branching strategy. This reduced the penalty of a mispredict. In addtion, the machine used the loop buffer to store eight double word instructions of a short loop. The hareware checked whether there was a backward branch to within the past eight instructions. If so, the instruction unit entered into the "*loop mode*," and began to fetch instructions from the loop buffer.

Many of the common architectural features found in today's microprocessor can be seen in this machine. For example, today's data cache, reservation stations, and non-blocking store/load are respectively called "operand buffer," "operation buffer," and "address buffering" in the 360. Althogh in this paper they "operation buffer" the term reservation station comes from the same work.

With the advances in VLSI technology, microprocessors today incorporate many of the ideas that appeared in the mainframes in the '60s. The main differences between machines in the '60s and today are:

- Machines then did not worry about delay to main memory
- Machines then used core memory. In 1978, VAX was the first to use solid state memory. And solid state memory was not popular until the '80s.
- Machines today are very concerned about memory latency, so a lot of effort has been put into changing the memory architecture.

2 Complexity-Effective Superscaler Processors [2]

This paper defines the term *Complexity-Effectiveness* as "delay of the critical path through a piece of logic and the longest critical path through any of the pipeline stages determines the clock cycle." In layman's term, this means superscaler with faster clock cycle.

Although the introduction section of the paper emphasizes on how wire delay is becoming an important factor in path delay, wire delay is not explicitely taken into account.

Palacharla describes two ways to implement the renaming logic: RAM and CAM. The table size of RAM is equal to the number of logical registers. The RAM table mapping is done by indexing into the entry corresponding to the logical register to find the matching physical register. In contrast, the table size of CAM is equal to the number of physical registers. The CAM table mapping is done by broadcasting the logical registers and find out which physical register matches the logical register. In comparison, the RAM organization is more scalable since the number of architectrual registers is fixed for a given ISA. In this design, the delay is dominated by logic, so as the process scales the delay decreases linearly.

The delay of the wakeup logic is impacted by the issue width. Although there is some wire delay in the path, the quadratic term is quite small. Similarly, most of the delay of the select logic is also in the gates because the tree orientation of the logic structure.

In contrast to the other blocks, bypass logic has a lot of wires accross all functional units so the wire RC is the dominant factor in the overall delay. Assuming that the lengths of the functional units scale with the technology then the delay of the wires stays constant in seconds. This means that relative to gate delay, the wire delay is increasing. Bypass in an 8 issue machine in the 0.18um technology dominates cycle time. When you need it, you can't hide it.

From the analysis of the bypass logic, we see that the bottleneck in the overall performance is the communication between units. The solution is to reduce this global communication which leads to the alternative layout proposed in this paper. By clustering the functional units and registers and exploiting the dependencies between instructions, the data dependency between clusters is decreased resulting in reduced global communication. This is done by issuing dependent instructions to FIFO buffers associated with the different clusters. Thus, each cluster proceses a stream of instructions that pass values mostly upstream, utilizing the efficient intra-cluster communication and keeping inter-cluster communication to a minimum. The results from each cluster no long need to be broadcasted to all entries, but rather only to the head of these queues.

The key part of this clustering idea is the **steering logic** that steers the instructions into the different queues. Palacharla uses an heuristic to determine which FIFO receives the instructions. The heuristic, based on the availability of the operands to the instructions, is shown below:

1. If all the operands to instruction I are computed and are in the register file, steer I into a new FIFO.

2. I has one outstanding operand produced by I_{source} in FIFO F_a. If there is no instruction behind I_{source}, steer I into F_a, else steer it into a new FIFO.

3. I requires two outstanding operands producing from I_{left} and I_{right} from F_a and F_b, respectively. Apply rule 2 above to F_a. If the resulting FIFO is full or there is an instruction behind I_{left}, then apply rule 2 to F_b.

Using the dependence clustering degrades the overall performance by about 5-8%. Palacharla suggests that this lost in performance can be made up by increasing the clock frequency.

Besides the *Dispatch Driven* steering logic mentioned above, Palacharla discusses three other clustered microarchitectures:

1. *Single Window, Execution-Driven Steering*: This scheme uses one issue window and issues the instruction to the cluster which can provide the source value first.

2. *Two Windows, Dispatch-Driven Steering*: This scheme is similar to the heuristic discussed previously, except this scheme uses a flexible window instead of a FIFO.

3. *Two Windows, Random Steering*: This scheme uses the same structure as the *Two Windows, Dispatch-Driven Steering*, except that the steering of instructions into the cluster is done randomly.

Overall, all of these clustering techniques, except Random Steering, perform about the same with similar IPCs. Random Steering's worse performance can be attributed to the frequent communication between clusters.

The idea of clustering is being adopted by commercial microprocessors. The Alpha 21264 uses a similar clustering with *Single Window, Execution-Driven Steering* in its chip.

3 Precise Interrupts [3]

The difference between precise and imprecise interrupts is that with precise interrupts, the interrupts appear the same as in a single-issue, in-order machine. In comparison, the M-Machine has concurrent interrupts: the thread that causes the exception stores its state and other threads keeps going. The term "interrupt" and "exception" often have different meanings to different people; therefore, Prof. Dally suggets to use the term "event" instead.

Precise interrupts are important because you need to fix what you did. External interrupts are trivial because you can just stop sending the instruction to the execute unit. So the paper focuses on the internal interrupts.

The paper suggests a few ways to implement precise interrupts in an out of order machine.

1. Result Shift Register:
   - reserve the results bus / reservation vectors
   - force completion in order
   - all functional units have fixed latency
2. Reorder Buffer:
   - put out of order result into reorder buffer which is in program order
   - if "ready", then check whether an exception has occured. If okay, then commit
   - other instruction may wait for the result in the reorder buffer
3. History Buffer:
   - put old value into history buffer
   - if exception, place value in history buffer back to registers (in order)
   - the disadvantage is that there is only one result bus, so it takes one cycle per entry to restore from the history buffer. Thankfully, interrupts do not happen frequently
   - the register file has three ports
4. Future File:
   - two register files:
     (a) architecture file which is in sequence and is maintained by the reorder buffer
     (b) future file which is used by the execute unit
   - on exception, dump the content of the architecture file into the future file

Smith suggests that it is okay to pollute the cache as long as the memory is not touched. This is counter intuitive since a polluted cache cannot hide the memory latency which defeats the purpose of having a cache.

This paper is considered to be one of "The" paper on renaming because it is one of the first paper paper to talk about getting a new physical register for each write.

4 VLIW [4]

A short comparison of the three architectures are as follows:

- CISC – variable length instructions
- RISC – fixed length instructions, register-to-register instructions

- VLIW – puts many instructions into one instruction, exposes hardware to the compiler, and lets it do the scheduling

The VLIW exposes the hardware to the compiler and lets it statically schedule the instructions to exploit the ILP. Trace scheduling is a very powerful compiling technology and many companies license it from Muliflow because the same technology can be applied not only to schedule VLIW machines, but superscaler machines as well. Since the VLIW machines are heavily dependent on the quality of the compilers, it is necessary to overcome the complixity in the compiler in order to make VLIW viable.

The biggest problem with VLIW machines is binary compatibility. As technology advances, changes to the hardware such as increasing the functional units may result in the inability to run legacy code. One solution to the binary compatibility problem is to use a translator and to compile the code down to microcode and distribute the microcode only.

Another potential problem with VLIW architectures is running out of ILP in the executing thread. Although superscaler machines face similar problem, VLIW machines are mostly depending on the compilers to find the ILP in the static instructions and may miss some of the ILP found in the dynamic instructions.