# Register Hierarchy

Roshan Gummattira  Spyros Tsavachidis  Teresa Watkins

Department of Computer Science and Electrical Engineering
George R. Brown School of Engineering, Rice University
{roshankg, spy, watkinst}@rice.edu

## Abstract

*Attempts to exploit instruction level parallelism (ILP) by executing more instructions per cycle using parallel functional units increases the pressure on the register file and necessitates increasing the register file size, both through number of ports and number of registers. The size of conventional register files grow linearly with respect to the number of registers and proportionally to the square of the number of ports, which is costly in terms of size and latency. We believe a two-level, hierarchical register file structure implemented entirely in hardware can either increase the number of available registers for the same amount of space or decrease the size required for the same number of registers by splitting up register functionality into arithmetic and memory access sections, thereby decreasing the overall number of ports and therefore area and delay. Experiments were going to be run on Mediabench in RSIM to see if this could be implemented without introducing too much register overhead latency, but getting the simulator to work proved to be more challenging than expected and we ran out of time. Despite this, we feel that this technique is a scalable way to combat the negative effects of register pressure and increasing access latency.*

## **1.** Motivation and Hypothesis

In the competitive general-purpose microprocessor market, CPUs with higher clock speed and better performance have an advantage over their competitors. To increase speed and transistor density, developers using a smaller manufacturing process to scale down the feature size of the chips. The smaller the gate length of the transistor, the less time it takes for electrons to complete the connection between the collector and emitter, and the faster the signal can propagate through the connection.

To achieve better performance, microprocessor developers have steadily increased the issue width, or number of instructions executed each cycle, of their microprocessors. Current general-purpose microprocessors issue up to six instructions a cycle. Unfortunately, increasing speed and increasing issue width have negative side effects that combine to increase the latency of the register file. If these trends continue, the traditional monolithic register file will need to be remodeled to avoid becoming a bottleneck.

### **1.1** Clock Scaling vs Wire Delay

The problem with smaller feature size is that signals must traverse metal wires as well as transistors. The signal propagation time along a wire increases relative to the speed of switching logic as feature size decreases. Signal propagation time is based on the RC time constant of switching the wire between high and low voltages. As feature size decreases, the cross-sectional area of wire traces also decreases, which by the equation $R = \rho / (W \times H)$ (where $\rho$ is the resistivity of the conductive material, W is wire width, and H is wire height), means a higher resistance per unit length of wire.

The capacitance of the wire depends on the layout of the circuit, making it more difficult to derive a general estimate, but [1] gives some optimistic values that nevertheless increase with each decrement in feature size. This increase in the RC time constant means that less and less of the chip's area can be accessed in one cycle, as seen in Figure 1 from [1].

In Figure 1, $f_8$ and $f_{16}$ represent eight and sixteen *fanout-of-four* delays. FO4 is a commonly used delay metric that allows comparisons between different processes because it scales with the technology.

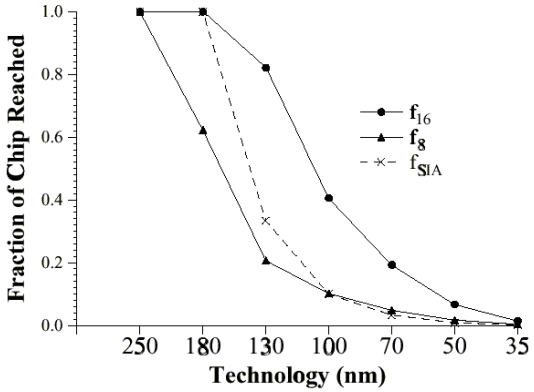



**Figure 1. Fraction of total chip area reachable in one cycle**

One FO4 is the time for an inverter to drive a capacitive load four times larger than its input capacitance. It takes at least 5.5 FO4s to compute the result of a highly optimized 64-bit adder [3], so 8 FO4s is the lower bound for the clock speed of a highly pipelined microprocessor and 16 FO4s estimates the clock speed of a microprocessor with a shorter pipeline. $f_{SIA}$ is the clock speed projected by SIA roadmap. This graph illustrates that with smaller processes, wire latency becomes increasingly significant and grows to be more of a factor than signal propagation through logic.

In summary, with decreasing wire size and increasing clock speed, less area can be accessed in a single clock cycle, limiting the size of a single cycle accessible register file.

## **1.2** ILP vs. Register File Size

A second major approach to increasing microprocessor computational power, extracting and exploiting increasing amounts of ILP, also has a side effects that inadvertently increases the latency of a register file access. Almost all mass produced modern processors try to take advantage of ILP by executing many instructions simultaneously on parallel functional units. But to make full use of these functional units, the register file must have enough read and write ports to supply the maximum possible number of operands required in a single cycle. Otherwise a functional unit will have to stall and wait for its operand value, and the register file will become a bottleneck. More functional units requires more read and write ports in the register file for a full crossbar. As we will explore in more detail in the next section, increasing the number of ports in each register of a register file greatly increases the size of the register file.

Besides more ports being required to communicate with the increasing number of functional units, more registers are required to provide space for the increasing number of operands needed to supply in-flight instructions. While most microprocessors spill and fill registers to memory at every context switch, the Itanium [4] tries to reduce the frequency of this occurrence by using register stacking. Register stacking allocates different sets of registers to different functions and only spills and to memory if it does not have a sufficient number of free registers to allocate to a function. To be effective this requires more than the minimum number of registers.

## **1.3** Register File Size Estimation

For our estimation of register file size, we borrow the model described by Rixner in [4]. In his model, the area of a register file is the product of the number of registers, the number of bits per register, and the size of a single-bit register cell. As seen in Figure 2 from [4], the size of each register cell is a function of the width of the register cell without ports ($w$), the height of the register cell without ports ($h$), and the number of ports ($p$) squared.

The area ($w \times h$) includes the storage space, power, and ground lines. Each read and write port requires at least one metal trace in each dimensions: one for the bit line to access the data and the other for word line to address it. This estimation for register cell size yields the quadratic: $p^2 + (w + h) \times p + w \times h$, which for large values of $p$ is about $p^2$. This is a safe assumption for our purposes because we are basing our model on the need for a large multi-ported register file. Combining both parts of the register file size estimation yields a total area of $Rp^2$, where R is number of registers times number of bits per register. This means increasing the number of ports into a register file gives a much greater increase in area, and therefore delay, than increasing the number of registers.

Because large multi-ported register files have long wires but little fan out, wire delay dominates over propagation delay. This delay is on the order of $pR^{1/2}$, showing more directly the negative timing impact of adding more registers and ports to the register file.

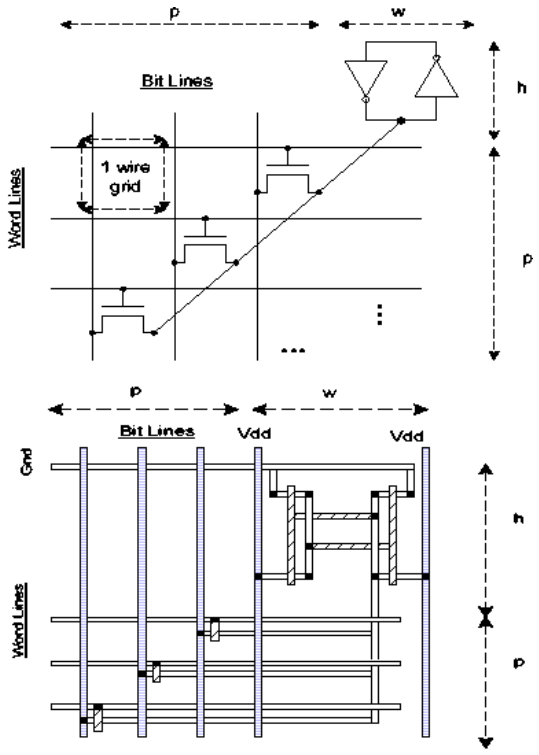



**Figure 2. Schematic and layout of a register cell**

For example, according to the model in [1], increasing the number of ports in a 64-entry register file from ten to thirty-two for a 35nm technology increases the access time from 172ps to 274ps (from 2.3 to 3.7 clock cycles at the SIA projected clock speed for that process).

## **1.4** Hypothesis

The number of ports and registers in a monolithic register file can be increased only so far with increasingly smaller processes until it is no longer possible to access all available registers in a single cycle. This increased access latency is what our modification is trying to combat. We feel that by breaking up the register file into two levels (a smaller higher level to interface with arithmetic units and a larger lower level which interfaces with the higher level and memory) we can either decrease the size (and therefore latency) of the register file or fit more registers into the same amount of space without introducing new instructions or slowing down the execution pipeline. We wanted to see at what point the area and delay improvement of the two-level structure makes it preferable to the single level structure, taking into account the additional overhead of register transfer and coherency between levels.

The next section of this paper examines the two-level register file structure in depth and models idealized gains over the single level structure. Section three looks at the experimental parameters and test setup for RSIM and section four covers the results of these simulations. Section five contains our conclusions and explores other techniques for reducing register file latency.

## **2.** Two-Level Register File

Our experimental two-level register file uses the lower level registers as a cache that has the ability to prefetch values from memory and hold register values not immediately in use. It accomplishes this by either taking less space for the same number of registers or placing more registers in the same amount of area occupied by the central register. Figure 3 shows the conceptual layout of our register file structure.

The main challenge of this register file structure was trying to create an effective, fast process for switching register values quickly and transparently between register levels. To be transparent, the move needed to take place between the time the instruction is decoded (i.e. the operands and type of instruction are known) and the time the instruction executes. Unfortunately, the implementation varies depending on the machine's pipeline. We tailored our algorithm for the RSIM pipeline, which has six stages: fetch, decode, issue, execute, complete, and graduate. This leaves one stage, the instruction issue stage, between decode and execution in which to position the registers.

To reduce the number of times we had to write values from the first level into the second level on a switch, we decided to make the first level registers a subset of the second level registers. That way, if the register value has not been modified in the first level, we can simply overwrite it on a register switch and avoid writing to the second level. More detail on this is given later.

For the area and delay comparisons, we chose to make the number of ports required for the interconnect (I) equal to half of the number of registers in the top level, with a read port to write port ratio of 3:1.

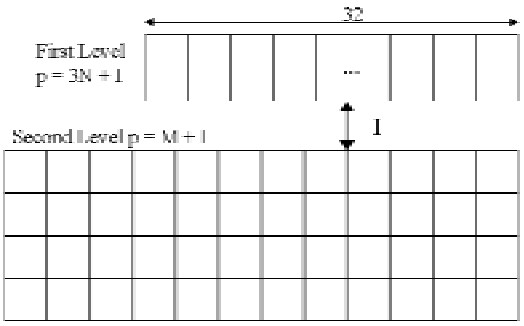


Figure 3. Two-level Register File

This means it can take up to three cycles to swap out every value in the first level register file, which will happen on a context switch, but we assume that in general, new register values will be not be needed for all upper level registers every cycle.

To determine the required number of ports in each level of our two-level register file, we took the estimation for number of ports in a central register file organization [4]: (M+3)N where M is the number of ports to memory, and N the number or arithmetic functional units. Then we and modified the equation to suit our two-level architecture. While the central register file requires enough ports in each register to supply all the arithmetic functional units and cover memory access latency, in the two-level register file, only the second level requires ports to access memory and only the smaller first level requires ports to access other functional units. Both levels require ports to interface between the two levels. Combining these factors, we decided that the first level requires (3N + I) ports per register, where I is the number of ports required to interface between the two levels. The second level requires only I + M ports per register.

In the two-level structure, we assume that the first level has 32 registers for up to 10 arithmetic functional units and 64 registers for 10-20 arithmetic units. There are two approaches to deciding on the number of second level registers. One approach is to use the same number of registers as our base case, 128, and see how that affects area and delay. The other is to use the same amount of area and see how that affects number of registers and delay.

## **2.1** Area and Delay Approach

Because we are not actually building this structure, we will model the decrease in and increase in area using the estimations from [4], where area is approximated by $Rp^2$. Assuming M=4, I=16 for 32 first level registers and I=32 for 64 first level registers, we get the graph below for area vs. number of arithmetic units

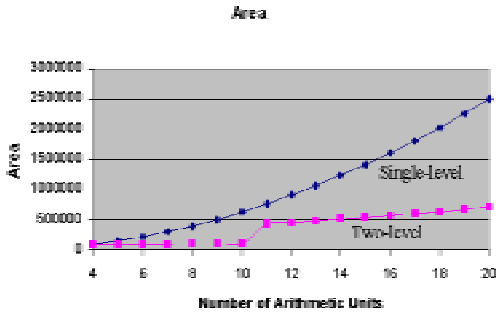



**Figure 4. Area Comparison of Register File Organizations**

The reduction in area becomes more pronounced as the number or functional units increases. Because area is directly related to power, by decreasing the area of the register file the power required by the register file is decreased. This graph makes it easy to see how much better the two-level structure scales with increasing numbers of functional units. Changing the parameter M while keeping the number of functional units constant has a similar but slightly less dramatic difference.

In translating register file area to delay, delay from the upper level (or single level) register file to the functional units is proportional to the number of registers R in the single register file for N < 10 and proportional to $pR^{1/2}$ for N > 10. Delay for memory operations is ignored because it is assumed to be much less than the latency of memory accesses. In the two-level register file, delay is proportional to R for N < 18 (because N =18 has same area as p = 10 in single level structure) and $pR^{1/2}$ for N > 18. These results are modeled in figure 4.

For N < 10 the two-level register file has a slight advantage because it has fewer registers in the structure that interfaces with the arithmetic units. There is a significant difference in delay for N > 10, and overall the delay of the two-level structure increases much more slowly than the delay of the single-level register file.

Unfortunately, this result is overly optimistic because it completely ignores increased latency from register file management overhead. It remains to be demonstrated that this difference in delay is greater than the increase in overhead for a two-level structure.

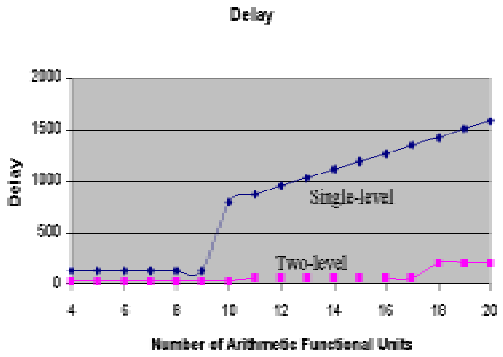



**Figure 5. Delay Comparison of Register File Structures**

## **2.3** Delay and Number of Registers Approach

In this approach we use the area estimations from before, but this time we try to see how many registers we can fit into the same amount of area. The size of the first level does not change, so we still have the same delay benefits for functional units accessing the first level. Now the limiting factor for delay is how much of the second level is accessible from the first level in a single cycle, but this scales much better than the monolithic register file. This is because the number of ports (and therefore area) in the second level increases more slowly in the two-level structure than in the monolithic structure. Table 1 describes the increase in number of registers in the second level as the number of functional units increase (again M = 4 and I = half the number of first level registers).

**Table 1. Number of Registers for given area**

| *N* | *First Level* | *Second Level* |
|---|---|---|
| 4 | 32 | 128 |
| 5, 6 | 32 | 256 |
| 7, 8 | 32 | 512 |
| 9, 10 | 32 | 1024 |
| 11,12 | 64 | 256 |
| 13 - 16 | 64 | 512 |
| 17 - 21 | 64 | 1024 |

We chose the nearest power of two that took up less space than the monolithic register file to allow room for the increased logic needed to manage the two-level structure. Despite this, there is a significant increase in the number of available second level registers. As wire latency has not quite reached the point of multi-cycle register file access, this result shows that there is still some benefit to this modification even if by taking the same amount of area it does not decrease delay, because of the increase in register availability.

## **2.4** Register File Management

The register swapping logic is simple enough that is should be able to execute alongside the pipeline, moving the operand to the appropriate level before the instruction reaches the execute stage. The logic required to oversee the administration is added to the register rename logic already present in most ILP extracting architectures (except Itanium, in which case it would be added to the register remapping logic). The first-level registers are a subset of lower level registers to simplify register coherency. A few extra bits need to be added to the register rename logic: a first level and second level dirty bit, the location of the register in the first level, the location of the operand in the second level, and a simple LRU counter for each first level register for the replacement scheme.

After the decode stage, if the instruction is a memory store operation, it will check to make sure that there is not a modified (dirty) value of the operand in the first level. If there is, it will update the second level with this value. If the instruction is a load, no such check is necessary. Next any first level entry will be removed by modifying the rename logic to indicate that the operand is not present in the first level and changing the LRU value to indicate that the first level register is empty. This should be done before the execution stage, at which time the instruction will proceed normal using the second level register. If the instruction changes the operand value, the second level dirty bit must be changed to reflect this.

For an arithmetic operation, if every operand is not present in the first level, they must be placed in empty or least recently used first level registers. In the later case, if the previous operand was dirty, it must be written back to the second level). The rename logic must be modified to indicate the register's new position, and the LRU counter must be changed to indicate that the register was very recently used. If one of the operands is modified by the

arithmetic operation, this must be indicated by changing the first level dirty bit.

A possible optimization of this algorithm is to modify the LRU replacement technique to evict older unmodified values before modified values to reduce the number of writes back to the second level. More simulations would need to be run to see if how this would affect switching time between different levels of the hierarchy.

## **3.** Experimental Set-up

Our original plan of testing the modified register file structure on Simplerscalar using Spec2000 benchmarks had to be modified when it was discovered that Simplescalar had a hard-coded register file and no register renaming. Instead the modification was tested by running Mediabench on RSIM, a simulator by Vijay Pai at Rice University.

RSIM is execution driven models current ILP processors, featuring out-of-order scheduling, static and dynamic branch prediction, and multiple instruction issue per cycle. It follows the architecture of the MIPS R10000 processor. It supports register renaming and has a configurable number of registers via multiple register windows (16 registers/window).

The most difficult part of altering the simulator to implement the 2-level register file was slowing down the instructions at the execution stage according to the increased access latencies of the associated registers. RSIM does not have configurable register file access latencies, so we would have had to keep track of all the instructions and slow them down. To help simulate a 2-cycle monolithic register file we would have added an extra latency cycle in all instructions in the execution stage.

The implementation of the 2-level register hierarchy involved the following steps:

- Keep track of all instructions coming from the instruction issue stage. These instructions have their dependencies solved and are ready to execute.
- For each of these instructions, we find its operands. These refer to physical registers since the register renaming happened in the instruction decode stage.
- For each of the above registers, we check where in the 2-level register structure they are found. If we have everything we need in the upper level, then there is a single cycle access penalty. If we have to move data between the two levels there is a 2-cycle penalty. We also update the mapping between the 2 levels and the dirty bits used for coherency.
- We update everything again, when the output of the ALUs is ready.

It is noted that there is no extra penalty when the interface between the 2 register levels is congested. This was due to implementation difficulties and would have been omitted. We believe that this should not be important since we already assumed that I (number of intrer-level ports) is equal to R1/2 (number of level 1 registers) which presumably provides enough bandwidth.

It is noted that this implementation is ‘crude’ but we believe that this configuration can give valuable evidence about the applicability of the register hierarchy idea.

### **3.1** RSIM parameters

The RSIM parameters we would have used for the simulation were chosen to model a processor with multiple functional units, many registers and high clock speed. This is reflected by the L1, L2 and memory latencies that are estimates based primarily on [1].

**Table 2. RSIM Parameters**

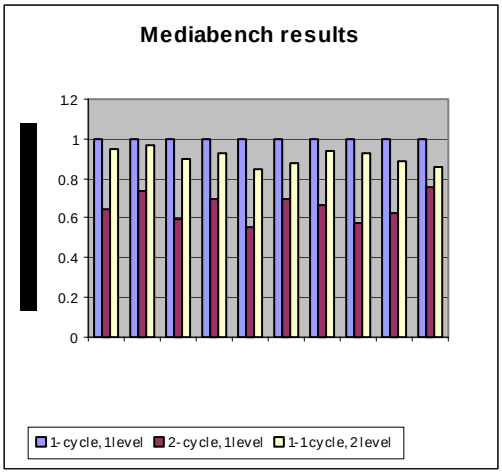
| Parameter | Single-level | Two-level |
|---|---|---|
| ALUs | 8 | |
| FPUs | 2 | |
| Registers | 128 | 32, 128 |
| L1 cache | 16K 1-way | |
| L2 cache | 64K 4-way | |
| L1 access latency | 3 cycles | |
| L2 access latency | 8 cycles | |
| Instruction window | 64 | |
| Branch prediction | 2-bit history predictors | |
| Memory Latency | 36 cycles | |

In RSIM, we were going to run Mediabench using the parameters in Table 2. We compared a modified register file with 32 first level and 128 second level registers to a 128-entry single level register file with either one or two cycle access latency. The bandwidth between the two levels of the register file (I) was assumed to be infinite.

This means, as explained earlier, that it takes one cycle for the functional units to access first level registers and two cycles to access second level registers. This experimental setup follows the approach 'keep the number of registers the same, but reduce latency and area' as described in section 2.3.

## **4.** Simulation Results

We would have used several benchmarks from Mediabench to test our approach. The key feature of these programs is the intense use of integer operations. A thorough analysis of their behavior can be found in [6]. The following graph is made up based on what we think should have happened. It is based on [2] and does not have any real simulation data to support it. But we expect out register file structure to outperform a single monolithic file of 2-cycle access latency and perform worse than a single cycle latency register file..




## **5.** Conclusions

Our modeling results give evidence that our hypothesis is correct, but we have no real simulation data. We conclude that this is a good idea to test, but that most microprocessor simulators are not built in such a way that allows easy modification and testing of different register file implementations. This is understandable since up till now it has not been a limiting factor to performance, but we feel that it may become so in the future. Realizing any gain from our modification depends mostly on the ability to fit the necessary logic into the pipeline without slowing it down, but with the simplified register-switching algorithm this will hopefully not be too difficult.

Further research into this might first test our configuration and expand it further to compare the two-level register hierarchy to a pipelined register file or one of the other implementations discussed in the following sections.

The final section, 5.6, looks at a real-world register file and compares it to our model. The following discussion shows that our model may be a little harsh. There are many other improvements currently being used that seem to be effective at reducing the size of the register file without changing the basic single-level structure.

## **5.1.** Similar Work

There have been many innovative ideas for creating a more efficient register file, and most modern microprocessors use their own optimizations to reduce the number of ports required. In order to be effective through multiple generations, the solution must be scalable. All of the ideas presented in this section involve providing less than a full crossbar between all register and all arithmetic functional units, but the choice of partitioning scheme and method of implementation vary dramatically.

Rixner suggested several partitioning methods in [4], including a distributed register file organized around SIMD clusters and a hierarchical register file separating memory registers and arithmetic registers. Our approach is a closest to the hierarchical organization, but meant more for general-purpose microprocessors than media processors. Zalamea suggested a two-level hierarchical register file for VLIW processors, but transfers between levels were handled by explicit instructions instead of implicitly by hardware. Cruz proposes a multiple-banked register file that is similar to but different from our two-level approach.

## **5.2** Distributed Register File

The distributed register file involved clustering registers around each arithmetic functional unit to allow fast accesses to registers local to a particular functional unit, but slower accesses to

registers elsewhere through a fast switched interconnect. This requires only two ports for the local arithmetic functional unit, but because the size of the full crossbar interconnect between all N arithmetic functional units grows as $N^2$, this is most beneficial for when interconnection is required only within smaller data independent SIMD clusters. This approach is more scalable than a single central register file and works very well for media processing, but is less effective for general-purpose processors because of increased dependence between instructions and the size and latency overhead of the switched interconnects.

## **5.3** Hierarchical Register File

This hierarchical register organization, like the one we are proposing, has a smaller first level with ports that communicate with the arithmetic functional units and a larger second level with ports only to the first level and memory. The implementation was not discussed in great detail, but the motivation was. Rixner proposes replacing the data cache in media processors with the memory-interfacing second level of the register file because data in media applications displays little locality, making data caches less efficient. Although it is true that this organization is particularly useful for media processors because of the large number of parallel functional units they possess, we feel that this organization will also work well with (rather than in place of) the caches in general-purpose processors, especially as latency to the first level cache increases (another side effect of increasing wire delay) and number of functional units increase.

## **5.4** Compiler Controlled Register File

Zalamea [7] simulated a compiler controlled two-level register hierarchy. This makes the hardware implementation simpler (we are still not sure whether even the simple amount of logic introduced for the two-level hierarchy management in section 2.4 will fit in a highly optimized pipeline), but it introduces additional instruction overhead to perform transfers between register levels and cannot hide register switching as we hope to. Their results were not as promising as the ones we measured.

## **5.5** Multiple-Banked Register File

Cruz suggested dividing the register file into several banks of varying number of registers and ports. Two examples of this organization are given in Figure 6 from [2]. Cruz noticed that while a large number of registers are needed to hold all of the operand values, only a few of those values are actually needed at a given time. Because of the similarity of this approach to memory hierarchy, he called his structure a register file cache.

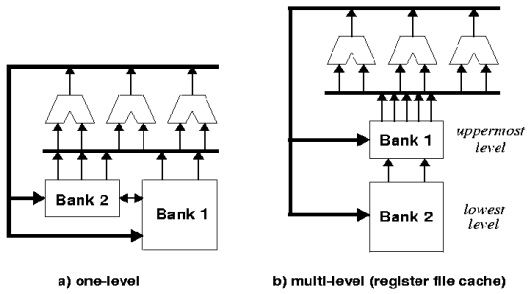



**Figure 6. Multiple-Banked Register File**

The main difference between this approach and the one we implemented is that in the register file cache, both memory and arithmetic functional units access the highest level and values are always written to the lowest level to maintain register coherency. This paper also focuses attention on different levels of register bypassing and prefetching schemes for the register cache.

## **5.6** Itanium Example

The problem of register file latency has already begun to affect current microprocessors. Intel's Itanium microprocessor has two large 128-entry register files, but they use methods other than multi-level register files to reduce the number of ports. Our model assumed that there would be MN ports devoted to memory accesses, but in reality there are many less than that. For example, by the model the Itanium would require 40 read and write ports, but in reality it gets by with only 14. For one thing, the Itanium uses multiple memory banks to allow the same write port to do twice the work and modify two memory locations, one in each memory bank.

Instructions templates give the Itanium another advantage. By assigning particular instructions to certain locations within the template, they can restrict the interconnections in

a way that allows operands to reach the destinations they need while at the same time reducing the routing complexity. The main lesson to learn from this is that designers sometimes find it more comfortable to stretch existing structures to the limit rather than trying something new and different.

## **6**. References


[1] V. Agarwal, M. Hrishikesh, S. W. Keckler, and D. Burger. *Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures*. In Proceedings of the 27th International Symposium on Computer Architecture, pages 248-259, June 2000

[2] J.L. Cruz, A. Gonzalez, M. Valero, and N. Topham, "*Multiple-Banked Register File Architectures*".

[3] S. Naffziger. A subnanosecond 0.5μm 64b adder design. In *Digest of Technical Papers, International Solid-State Circuits Conference*, pages 362-363, February 1996.

[4] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, and J.Owens, "*Register organization for media processing*", in proceedings of the 26th International Symposium on High Performance Computer Architecture.

[5] H. Sharangpani and K. Arora, Intel *Itanium Processor Microarchitecture overview*. Technical report, Intel Corporation, Santa Clara, CA, USA, 2000.

[6] Smith William, Chunho Lee, MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems.

[7] J. Zalamea, J. Llosa, E. Ayguade, and M. Valero, "*Two-Level Hierarchical Register File Organization for VLIW Processors*".