Reducing Energy Requirements for Instruction Issue and Dispatch in Superscalar Microprocessors*

Kanad Ghose
Dept. of Computer Science, State University of New York
Binghamton, NY 13902–6000. http://www.cs.binghamton.edu/~ghose

Abstract: Recent studies [MGK 98, Tiw 98] have confirmed that a significant amount of energy is dissipated in the process of instruction dispatching and issue in modern superscalar microprocessors. We propose a model for the energy dissipated by instruction dispatching and issuing logic in modern superscalar microprocessors and validate them through register level simulations and SPICE–measured dissipation coefficients from 0.5 micron CMOS layouts of relevant circuits. Alternative organizations are studied for instruction window buffers that result in energy savings of about 47% over traditional designs.

Keywords: power minimization, superscalar processor, instruction dispatching, instruction issue, window buffer

1. INTRODUCTION

Most modern microprocessors employ multiple instruction dispatching and multiple instruction issuing per cycle to achieve their performance goals. In a k–way superscalar processor, instructions are typically fetched for processing by the fetch stage in groups. A group of k logically consecutive instructions are then sent to the decode/rename1 stage (D/RN), where the instructions are decoded and physical registers for the architectural registers holding the results are assigned and inter–instruction dependencies are noted. Also in this stage, the most recent physical registers corresponding to the architectural registers needed as input by each of these instructions are looked up from a rename table. In the next stage, read PRF/dispatch, the input physical registers that contain valid data are read out and up to k instructions are moved or *dispatched* to a buffer called the instruction window buffer (IWB). In the process of dispatching, it is not necessary for the instructions to have all their input operands available; nor is it necessary for the execution units (execution logic) for these instructions to be available. Dispatched instructions wait in the IWB till their input operands and the required type of function units (FUs) are available, at which time they are ready for execution. Moving such ready instructions to the execution unit constitutes the *issue* step.

The IWB plays a critical role in instruction dispatching and issuing. Figure 1 (a) shows the logical structure of the IWB and the format of a IWB entry. The IWB is essentially a static multi–ported register file with some associative addressing and forwarding capabilities. Each entry has a bit to indicate if it is allocated or free; a field to hold the value of each of two input operands (operand 1 and operand 2, respectively) and whether these fields contain valid data (valid bits). At the time of dispatch, free IWB entries are located associatively for each dispatched instruction and an IWB entry is set up for each the dispatched instruction as follows:

* Supported in part by the National Science Foundation through award No. MIP–9504767.

ISLPED '00, Rapallo, Italy.

(a) The type of the function unit needed is noted in the entry.

(b) Literal operands, if any, are moved into the corresponding field ("operand 1 value" or "operand 2 value") and the corresponding valid bit ("operand 1 valid" or "operand 2 valid") in the entry is set.

(c) A register operand is read into the corresponding operand field of the entry and its associated valid bit in the entry was set if that physical register contained valid data. If an input physical register did not contain valid data (i.e., was waiting for a result to be written into it from a function unit), the "operand valid" bit of the corresponding IWB entry is cleared (to mark it as invalid) and the tag field associated with the IWB entry is set to the address of the corresponding input physical register.

A tag based forwarding logic is used to pull in (i.e., forward) the value of a result into IWB entries that are waiting for it. The tag field associated with an operand in an IWB entry indicates the address of the physical register whose contents, when generated, have to be used as the value of that operand. Results produced by the function units can be forwarded to waiting IWB entries from multiple buses running across all of the IWB entries. Two such result buses, Bus A and Bus B, are depicted in Figure 1 (a). There are two sets of lines for each result bus. One carries the actual result and the other carries the tag – the tag being the address of the physical register for which this result is eventually destined. For each forwarding bus, an IWB entry for each input operand (operand 1 or operand 2) has a comparator that compares the tag value stored in the entry against the tag values floated on the tag buses. If a tag match occurs, the value on the corresponding result bus is latched into the associated operand value field and the valid bit of the operand is set to indicate that the operand value is valid. This scheme allows the value for operand 1 and operand 2 to be latched in from any one of the result buses. When both operands of an IWB entry are valid, that entry is potentially ready for issue.

The dispatch of up to k instructions per cycle results in up to k simultaneous writes to the IWB, requiring k write ports into the IWB. Free IWB entries are located for this purpose by an associative searching on the allocated/free bit of the entries with static priorities assigned to the write ports for accessing free entries. As results are produced from the function units, they are forwarded to waiting IWB entries. For overall flow balance, paths must be provided for at least k simultaneous issues, requiring k read ports from the IWB. For the same reasons, at least k sets of forwarding buses must be provided. A typical timing diagram for the IWB is shown in Figure 1(b). The forwarding of results into the IWB and the dispatch of a new set of instructions are overlapped to minimize the cycle time: this forces the use of independent sets of bit lines for the forwarding and writes. The selection of ready IWB entries for issue and moving them to the required function units is generally a slow process [PJS 96] and takes almost a full cycle.

2. ENERGY DISSIPATIONS WITHIN THE IWB

The main sources of dynamic power dissipation in the process of instruction dispatch and issue are as follows:

I. Dissipations in dispatching instructions

The main components of power dissipations in dispatching instructions to the IWB are as follows:

(a) Power spent in prioritizing the associative addressing of free IWB entries through the write ports of the IWB following a predefined order. This prioritization is needed since the number of instructions dispatched (say Q) in a cycle can be fewer than k. In that case, the writes are directed through the first Q write ports of the IWB, while the remaining ports are unused.

(b) Power spent in driving the bit lines of the write ports to establish entries in the IWB: the number of transitions caused as a result of writing Q instruction entries into the IWB in a cycle is roughly proportional to W*k*Q*N, where W is the number of bits written for each entry and N is the number of entries in the IWB. The factor k is contributed by the diffusion capacitances of the k pass transistors that connect the bit lines of each port to a bitcell. The dependence on N comes from the fact that the bit line capacitances are directly proportional to N, the length (i.e., number of rows) of the array.

The energy spent in dispatching Q instructions to the IWB is thus:

$$C1.k^2 + C2.W.k.Q.N \quad (1)$$

where C1 and C2 are appropriate constants that are products of a numeric constant, capacitive coefficients, the supply voltage and the bit line voltage swing on writes.

II. Dissipation in selecting and issuing instructions

The main components of energy dissipation in the process of selecting and issuing instructions to the execution units are as follows:

(a) Power spent in selecting up to k ready instructions for issue to the execution units. These k ready entries can come from any of the N locations within the IWB. As the value of N is typically 20 to 64, typical implementations use a tree of simple arbiters to select a particular FU of the required type for issuing a ready instruction [PJS 96, Vas 96]. The total height of the arbiter tree is $\log_n N$. Assuming that M execution units of distinct types are available, M such arbiter trees are needed. The ready/waiting bit of each IWB entry drives a request input in only one of these arbiter tree; the tree chosen is based on the contents of the FU type field of the entry.

(b) Power spent in driving the contents of the selected rows of the IWB on the bit lines of the ports. The energy spent in this process, assuming R ready entries (R < or = k) are issued is roughly given by:

$$C3.N.b.W.k.R + C4.b.W.R \quad (2)$$

where C3 and C4 are constants (like C1 and C2), b is the fraction of bit line pairs that are partially discharged in the readout process. The first term accounts for the partial discharge of bit lines affected by the read. The k in the first term accounts for the diffusion capacitances of the k pass transistors that connect the bitcell to the k read ports. The second term accounts for dissipations in the sense amps and output drivers.

III. Dissipations in forwarding results to the IWB

The main components of energy dissipation in forwarding results to waiting IWB entries are:

(a) Energy spent in driving the results and tags over the result buses. If P results are driven in a cycle, the energy spent in this process is:

$$C5.f.B.(D+T).P.N.r \quad (3)$$

where D is the width of the data part of the result bus and T is the width of the tag part of the result bus. f is the fraction of the bus lines that make a transition when the results and tag are driven. C5 is a constant like the others used earlier. The factor N reflects the length of the buses across the rows of the IWB and B is the total number of result/tag buses. The factor r comes into play since each tag and result bus drives up to r input operand tag/data fields within an entry.

(b) Energy spent in the tag matching process. Assuming that fast, precharged comparators are used, where a precharged line is pulled down on a single mismatch in one of the g bits in the tag address (which is the number of address bits in a physical register address), the energy spent in delivering P results to Y operand slots on the average per cycle from each of the result data/tag buses is:

$$C6.P.g.(N.r - Y) \quad (4)$$

where the factor g represents the dependence on the length of the precharged line for each of the g–bit comparators. N.r is the total number of comparators associated with a single tag bus, that runs across r tag fields, one for each input operand.

3. REDUCING IWB ENERGY REQUIREMENTS

The foregoing analysis suggests several possibilities for reducing the power dissipation within the IWB. These are as follows:

Organization 1 – partitioned IWBs: In this case, instead of using a single unified IWB, we partition the IWB into multiple IWBs based on the type of instructions. One IWB could be devoted to LOAD/STORE instructions, another to integer instructions and another devoted to floating point operations. As a consequence of this partitioning, the number of read and write ports on each IWB can be reduced without appreciable loss in performance, directly reducing the energy spent in dispatching and issuing, effectively reducing k in equations 1 and 2. Further, as each IWB is smaller the arbiter trees for selecting instructions for issuing are smaller and lower dissipations results.

Organization 2 – partitioned IWBs with customized entry formats: As another side effect of partitioning IWBs by type, the format of entries in these IWBs can be different as the number and width of operands needed by the respective instructions can differ. For example, the IWB handling floating point instructions typically have longer operand fields (64 bits or higher), as well as 3 inputs (to handle "fused" instructions like multiply–and–add, typical in most high–end modern processors). This reduces dissipation in instruction issuing and result forwarding (by reducing the effective W and D in equations 2 and 3) and tag comparison (equation 4, by effectively reducing r).

Organization 3 – reduced transitions on bus and bit lines: A significant number of bit lines are driven in the process of dispatching instructions to the IWB, issuing ready instructions and forwarding results. If the number of significant bits in operands can be somehow be encoded, energy savings can be realized by driving only the lines that contain these significant bits. This capability can be implemented in two ways:

(a) By pre–configuring the data path: for example, when 32–bit operations are implemented on a 64–bit data path, an explicit instruction can be issued to allow only the lower 32 bits of the operand fields in the IWB entries to be active. No signals are driven on the higher order bits of the operands.

(b) By dynamically detecting the number of significant bits in the operands and storing the encoded length – in multiples of bytes, for simplicity – in an explicit field within the IWB entries. A single bit, for example, can indicate if both 32–bit fields in a 64–bit operand contain significant bits. The instruction decoder and the execution units can generate this bit by adding a simple logic to check if the higher order 32–bits are zero. As most literal operands are short, and not very many integer variables use more than 32 bits, this capability can be used to lower the effective W and D in equations (1), (2) and (3) to save power during instruction dispatching, issuing and forwarding.

4. RESULTS AND CONCLUSIONS

We used a detailed register–level simulator to glean transition counts, which accurately simulates at the cycle level a superscalar pipelined processor, based on the MIPS instruction set, as used in an earlier work

[GhKa 99]. For the results presented here, we used a 3 level cache hierarchy, with split L1 caches (clocked at 300 MHz.), an unified L2 cache (running at half the CPU clock rate) and an unified L3 cache. The L1 and L2 caches were assumed to be on–chip and the L3 cache was assumed to be off–chip. Our simulations were for a 4–way superscalar CPU with 2 LOAD units, 1 STORE unit, 6 integer units, 2 Integer multiply/divide (pipelined) units and 2 Floating point (pipelined) units. We simulated the execution of the SPECInt95 and few SPECFp95 (su2cor, mgrid, applu) benchmarks to get a good mix of CPU–intensive and memory–intensive load. For getting accurate measures of the dissipation in all major components of the IWB organizations studied, we laid out the major components of the IWB in a 4–metal layer, 0.5 micron technology and verified through SPICE simulations that it performed all major operations correctly and met the cycle time goal for a 300 MHz. clock. We also used SPICE to compute the energy dissipations in the IWB components on major transition events, using a supply voltage of 3.3 volts. These measures included not only dissipations due to capacitive loading, but also leakage and short circuit currents (the later being a particularly dominating component in the sense amps of the IWB).

The transition counts obtained from the simulations were fed into a power estimation program that looked up appropriate energy dissipations for each event as obtained from the SPICE simulations. The base case IWB was a 64–entry IWB with three operand fields in each entry. Each entry had three operand field, with 96–bit operand for handling extended precision floats. A value of 4 was assumed for k – this was also the number of read and write ports into the IWB, as well as the number of result buses. The arbiter modules had four inputs each. For the partitioned IWBs with customized entry formats and ports, we assumed: (a) an integer and a load/store IWB, each with the same number of ports and buses as the base IWB, but with 64–bit operand fields and 24 entries each; (b) a floating point IWB with 2 read, write ports and 2 forwarding buses, with 96–bit operands and 3 operands per entry.

Figure 2 depicts the power dissipated within the IPB in the base case and the power–efficient organizations discussed in Section 3. These dissipations are averaged over the simulated execution of the SPEC benchmarks as indicated earlier. In all of the organizations shown, the tag–matching based forwarding consumes the most power. This is expected, as the effective lengths of the tag and data buses that forward the result is a multiple of the bit line lengths. In addition, as only a few entries await a result on the average, the power dissipated by the majority of the pull–down comparators on tag mismatches is quite high. The partitioned organization, with the same number of ports and identical entry formats, the power saved over the base case comes from saving powers within the arbiters (which are smaller) and within the sense amps – the individual sense amps dissipate lower power and not all of these sense amps are activated simultaneously. The overall power savings compared to the base case is about 18%. Further power savings are realized by moving to organization 2, which has IWB entry formats as well as number of ports tailored to the IWBs by the type of instruction they handle. This is as expected and is in concurrence with the rationale presented earlier in Section 3. A power savings of 35% is realized with respect to the base case. When operand lengths are encoded to reduce unnecessary driving of signals on bit lines and forwarding bus data lines that contain no significant bits in the higher order position (Organization 3), a dramatic power savings of about 47% is realized with respect to the base case. Organization 3 thus seems to be the one of choice for IWBs in modern superscalar CPUs.

REFERENCES

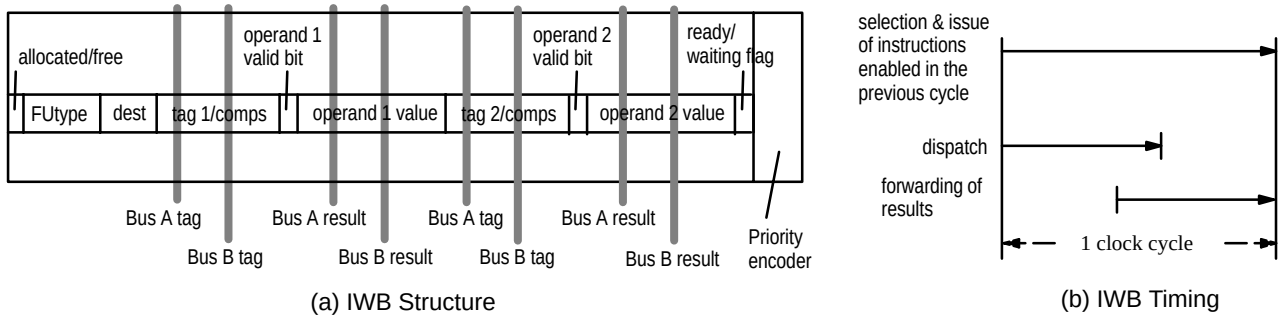
[GhKa 99] Ghose, K. and Kamble, M.B., "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit–line segmentation", in Proc. ISLPED '99, pp. 70–75.

[MGK 98] Manne, S., Grunwald, D., and Klauser, A. "Pipeline gating: speculation control for energy reduction", in Proc. 25–th Int'l. Sym. on Computer ARchitecture, pp. 132–141, 1998.

[PJS 96] Palacharla, S., Jouppi, N. P. and Smith, J.E., "Quantifying the complexity of superscalar processors", Technical report CS–TR–96–1308, Dept. of CS, Univ. of Wisconsin, 1996.

[Tiw 98], Tiwari V., et al., "Reducing power in high–performance microprocessors", in Proc. Design Automation Conf., 1198, pp. 732–737.

[Vas 96] Vasseghi, N. et al, "200 MHz. Superscalar RISC processor circuit design issues", Proc. ISSCC digest of technical papers, 1996, pp. 356–357.



(a) IWB Structure (b) IWB Timing

Figure 1. Typical structure of an instruction window buffer (IWB) and its timing. Read and write ports not shown.

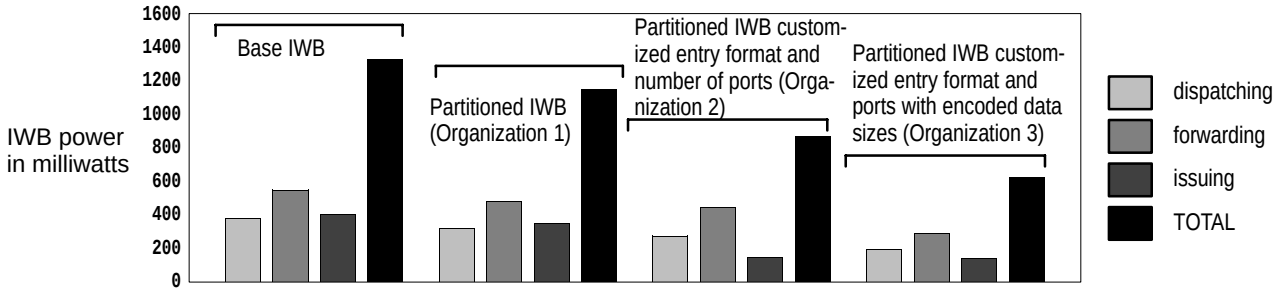


Figure 2. Power dissipations within the IWB in the base and the optimized organizations