Journal of Circuits, Systems, and Computers
Vol. 18, No. 6 (2009) 1093–1117

World Scientific
www.worldscientific.com

MODIFYING THE DATA-HOLDING COMPONENTS OF THE MICROPROCESSORS FOR ENERGY EFFICIENCY

YUSUF OSMANLIOĞLU, Y. SİNAN HANAY and OĞUZ ERGİN*
TOBB University of Economics and Technology,
Söğütözü Cad. No:43 Söğütözü, Ankara, Turkey
**oergin@etu.edu.tr*

Revised 12 May 2009

Storage components are a major source of energy dissipation in modern superscalar microprocessors. As the instruction windows get larger with each generation of processors, register files and other structures that hold intermediate data become larger and dissipate more power. Therefore it is important to find new ways to reduce the energy dissipation of data holding components.

Many values written to and read from these storage components are known to require fewer bits than is provided by the storage space. Upper order bits of these values are unnecessary and energy savings can be achieved by not writing and reading these bits. Floating value data has different characteristics and offer special energy savings opportunities. In this paper we analyze the bit-level statistics of data values and propose simple modifications to data holding components that save significant energy inside the processors. Our results show that by using simple modifications on the storage components it is possible to achieve 40% reduction in the integer register file energy dissipation and up to 60% reduction in the immediate field of the issue queue. We also show that energy dissipation can be reduced by half for some floating point benchmarks by identifying values that indicate a zero operand.

Keywords: Computer architecture; microprocessors; registers; energy management; integrated circuit design.

1. Introduction

Modern superscalar microprocessors use aggressive techniques like out-of-order execution and multithreading in order to boost performance. These aggressive techniques result in larger instruction windows with each new generation of microprocessors. With the use of register renaming, as the number of instructions inside a processor increases, more registers are needed to hold temporary results. Larger register files dissipate more energy due to increased length of bit lines and word lines and are a major hot spot inside contemporary microprocessors. Recently the datapath width of processors is increased to 64 bits which mandated the use of larger storage elements inside the processors.

Integer and floating point register files and immediate field inside the issue queue are the major place where the actual data is stored and read. In this paper we

propose some simple modifications in these data holding components in a processor to reduce energy dissipation. These modifications are done by observing the average operand values in common workloads and arranging the width of the storage space accordingly. In addition to the regular integer data, we analyze the patterns of floating point data that is stored inside the processors. This, to the best of our knowledge, is the first work that analyzes the data patterns of floating point values to achieve energy efficiency. The proposed solution would be especially beneficial for processors in which the data storage components dissipate a significant percentage of the total chip power. As an example, the register file (embedded in the reorder buffer) can dissipate more than 10% of the total chip power in the Intel Pentium Pro processor.[38]

2. Related Work

Performance benefits of working with smaller values were realized in Intel's Pentium 4 in the context of double pumped functional units.[15] These pipelined functional units operate with 16 bits of the value at a cycle and each operation lasts for 3 cycles. Function units of the 32-bit datapath work on the lower order 16 bits of the value in the first cycle and the rest of the value in the second cycle. The third cycle is reserved for handling overflow and carry-out. This way it is possible to push the frequency of the function units. Similar approach was proposed for a system level energy savings in Ref. 21.

Many researchers observed the presence of narrow values in some way, especially in the context of writing and reading of zero bytes.[28,33] Encoding bytes that contain zero was previously proposed for energy reduction. However inserting an additional bit for each byte is not a very efficient solution when value widths are large.

Significance compression was also proposed for energy efficiency by employing both hardware and compiler techniques.[7,8] In significance compression, size of the value is propagated throughout the pipeline in byte granularity. In contrast, our partitioning approach depends on defining the size of the narrow values for each data holding component separately at design time at whole data width granularity.

The narrowness of the values inside the processor was also used for improving performance. Multiple narrow operands were packed into wide function units in Ref. 4 and wide registers in Ref. 11. In Ref. 22, values that are narrow enough to fit inside a rename table entry were placed inside the rename table instead of the register identifier. In Ref. 23, narrowness of the values were predicted and depending on the prediction information multiple instructions were executed using a single function unit. Similar approach was proposed for VLIW machines in Ref. 26. Value prediction tables were partitioned according to operand width in Refs. 24 and 29 in order to reduce the complexity. Many previous works also showed that value widths are highly predictable.[11,23,24]

Reducing register file energy dissipation is a highly researched topic. There are different techniques to reduce the energy dissipation of data storage elements such

as partitioning the register file,[40] register caching and removing ports. Aggarwal and Franklin proposed reducing the ports from the upper order bits of the registers since those bits are usually not necessary.[1] Register caching was proposed to keep the usually accessed values inside a smaller structure[6] and dividing the register file into multiple banks (or clusters) was also proposed to keep the structure small and reduce its energy dissipation.[3, 10, 34] Reducing the ports by delaying the writing of values or multiplexing the read ports were also proposed in Refs. 19 and 27 for register file energy efficiency. Multibanking with a two-level register file organization was proposed in Ref. 2 in order to save energy and reduce access delays. Our proposed solution can be used together with these schemes for achieving further energy savings.

Kondo used the narrowness of the values for energy efficiency and performance by dividing 64-bit registers into 32-bit parts and allocating registers more efficiently in Ref. 20. Each and every instruction allocates 32-bit partitions but dependents may not use both partitions when they seek the value. In some aspects our work is similar to Kondo's but the fact that this technique requires modification on other processor components such as the issue queue, makes the technique more complex. Dependents have to remember two register tags instead of one in Ref. 20 which increases the energy dissipation of components other than the register file.

In Ref. 12, Gonzalez *et al.* proposed a content aware register file architecture. Register file is divided into multiple banks which include 3 different register files. One of these banks is a register file that holds the narrow values. If the register file needs to hold a wide value, pointers are stored in a separate bank to point to the location of lower order bits of a value. This architecture saves energy when a narrow value is accessed but takes a performance hit when a wide value needs to be accessed.

Wang *et al.* proposed a banked register file design where upper order bits of the registers are removed in some of register banks to achieve energy efficiency.[37] The width of the result value of all the instructions is predicted at the rename stage and a register is allocated from an appropriate bank. Although such a design offers better energy reduction opportunities, there is a performance penalty due to result value mispredictions.

Dividing the datapath into two asymmetric clusters based on operand widths was proposed for improving performance and reducing energy dissipation in Refs. 14 and 32. Instructions with narrow operands are steered into a cluster where only narrow function units and narrow registers are present. This partitioning creates a load imbalance because of the fact that many of the instructions do not have all of their operands narrow. Although many of the produced values are narrow, the percentage of instructions that can be steered to the narrow cluster is limited which limits the performance and energy reduction benefits.

Narrow values were also exploited to reduce the number of soft errors inside processors.[12, 16] Just identifying the narrow operands, as it is proposed in this work, and later sign extending the narrow part reduces the vulnerability of stored value

to particle strikes.[12] In order to protect the lower order bits of the narrow value, significant portion of the value can also be replicated inside the storage space for error detection and protection.[12,16] Parity generator circuit design that produce the parity signal faster for narrow values was also recently proposed.[41]

System level optimizations and compiler level techniques were proposed to reduce the energy dissipation of datapaths.[9,31]

3. Structure of the Storage Elements

Registers and immediate values are two major data types that are used in regular programs that are run on contemporary microprocessors. Registers hold both integer and floating point data types while immediate values can only be of integer types and are held inside the instruction queue of a processor.

Data holding components inside a processor are constructed using SRAM bitcells of Fig. 1. In a 64-bit processor, if there are 128 registers, the register file is composed of a 128×64 array of these SRAM bitcells. Likewise, each entry in the partition of the issue queue that holds the 64-bit immediate values include 64 of these bitcells.

Each port of the SRAM bitcell in Fig. 1 introduces two pass transistors to the cell. For a superscalar processor with P4 architecture that is capable of issuing 4 instructions per cycle, every bitcell needs to have 12 ports (4 write 8 read). Consequently, bitcells of the register files of a machine with a large instruction window occupies more area due to increased number of pass transistors. Large number of ports also increases the static and dynamic power dissipation of the bitcells.

Superscalar microprocessors employ register renaming in order to exploit instruction level parallelism and remove false data dependencies. In register

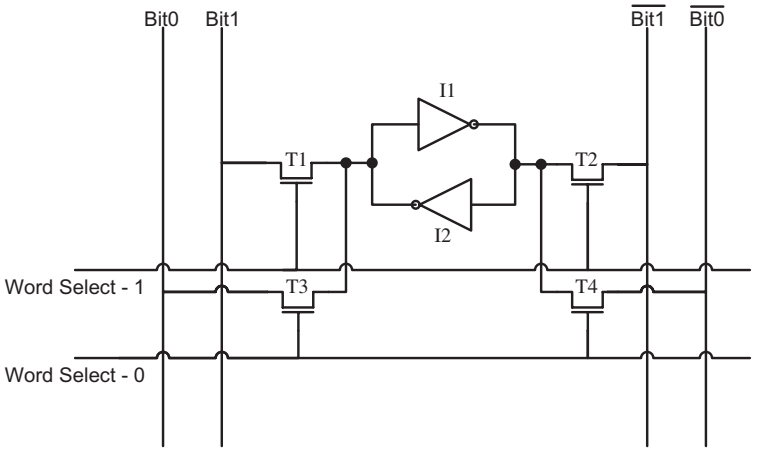


Fig. 1. Two-ported SRAM bitcell which consists of two inverters that are connected to form a positive feedback. Also there are two pass transistors per read/write port.

renaming, a new physical register is assigned to each and every result-producing instruction. After the physical register is assigned, architectural register that is used by the instruction as destination is mapped to this physical register until the same architectural register is overwritten by another instruction.

Register renaming is implemented in different ways in different microarchitectures. In P6 architecture of Intel, reorder buffer entries also serve as physical registers and there is a separate architectural register file that holds the architectural state.[17] In other architectures such as Intel's Pentium 4,[14] MIPS R10000[35] and Alpha 21264,[16] architectural register file and physical register file are combined into one structure and separated from the reorder buffer. In P6 architecture a physical register (reorder buffer entry) can be released at the time of instruction commitment, whereas in architectures with unified register files, registers cannot be released until the corresponding architectural register is overwritten by another instruction.[25] Because of the presence of register release constraints in architectures with unified register files, more registers are required resulting in larger register files.

Register file structure has some components other than SRAM bitcells. For faster operation prechargers are used to precharge the bit lines to V_{dd} before the registers are accessed. Register access starts with decoding the register identifier with decoders. When the decoder produces a match signal, word select driver is employed to select the entire entry that consists of 64-bits. After waiting for sufficient time for bitcell to create enough voltage difference between the bit and ∼bit lines, sense amplifiers are employed for rapidly sensing the value and reading it out to the latches. For a write operation, instead of prechargers and sense amplifiers, write drivers are used to charge the value of bit and ∼bit lines with strong signals and eventually force the contents of the bitcell to change to the new inputs. Figure 2 depicts the described storage structure.

All of the circuit components dissipate energy on a register file access. However, the power dissipation of word select drivers (which drive the gate capacitance of 128 NMOS transistors), write drivers (which drive the diffusion capacitance of transistors whose number is equal to the number of entries) and prechargers are more dominant as they have to pull large capacitive loads to V_{dd} level.

Register file energy dissipation can be divided into two components: static and dynamic. As the size of the register file increases, both components of the energy dissipation increases. Both static and dynamic energy dissipation of the register file can be reduced by exploiting narrow values. In this article we exploit narrow values for reducing dynamic energy dissipation.

3.1. *Storing floating point values*

Structure of the floating point values is different from the integer values inside the processors. Generally a floating-point value is represented with the following formula:

$$(-1)^S \times M \times 2^E \tag{1}$$

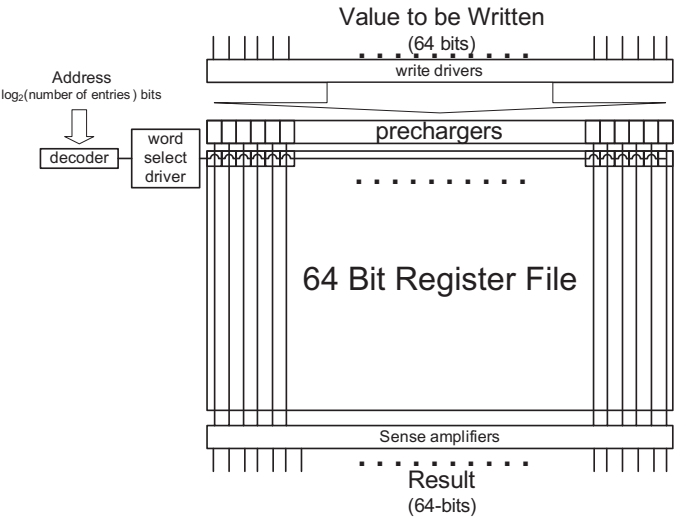


Fig. 2. Structure of a register file that is capable of storing 64-bit values. Register file access includes precharging bit lines, address decoding, driving the word select line, bitcell delay and sensing for a read operation.

where S denotes sign, M denotes the mantissa (fractional part) and E denotes the exponent. The most widely-used standard for floating-point computation is IEEE 754. This standard introduces methods for representing floating-point numbers and some special cases such as ±infinity and NaN (not a number). In 64-bit IEEE 754 standard, the exponent part of a value (E) is biased by 1023 bits. Although the same kind of circuitry that is used to store integer values is used to implement floating point storage structures, the 64 SRAM bitcells are interpreted differently as they hold information about three fields of a floating point value.

Figure 3 shows the description of the IEEE 754 floating point standard's data format for 64-bit floating-point numbers. The sign bit is positioned as the most significant bit, 11-bit long exponent part stores biased exponent, and the remaining 52 bits are used to represent the significant part of the floating point value without its most significant bit. For example, the number $(123456789)_{10}$ is represented as below:

$(123456789)_{10} = 11101011011110011010001010 1_2 = (-1)^0 \times 1.11010110111100110100010101_2 \times 2^{26}$
[From Equation (1)]:
Sign = 0
E = 26
Bias = 1023
$\text{Exp} = 1023 + 26 = 1049_{10} = 10000011001_2$
$\text{M} = 1.\underline{11010110111100110100010101}_2$
Fraction = 11010110111100110100010101000000000000000000000000000000

Special cases are represented by using the exponent and fraction parts of the storage area together. All of the bits of the exponent field are set to 1 to indicate

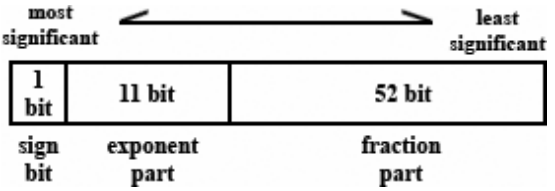


Fig. 3. The three fields in a 64-bit IEEE 754 floating point value.

infinity or NaN result. In order to indicate a zero value, all of the exponent and fraction bits have to be reset to zero. The table below shows the special cases and corresponding contents of exponent and fraction areas.

Type	Sign	Exponent	Fraction	Value
±Zero	*	00...00	00...00	0.0
−Infinity	1	11...11	00...00	$-\infty$
+Infinity	0	11...11	00...00	$+\infty$
Not a number	*	11...11	Non zero	NaN

*Sign bit can be either 0 or 1.

Floating point values are stored just as the integer values; 64 SRAM bitcells are used to hold a 64-bit IEEE 754 floating point value. However, since each floating point value has different fields, floating point data cannot be treated like a regular integer data.

4. Exploiting Narrow Values for Energy Efficiency

Narrow values can be used to reduce the energy dissipation of the integer value holding storage elements. As these values hold a lot of upper order repeating sign bits, they create inefficiency while being written to and read from the register file. By identifying the narrow values and not writing the upper order bits into the register file it is possible to reduce energy dissipation of the register file significantly. Also instead of reading the upper order portions, simple sign extension can be employed and the reading of unnecessary bits can be avoided.

4.1. *Definition of narrow values*

Many of the produced and consumed values in a processor datapath usually do not require the full datapath width. These values use the upper order bits of the storage elements unnecessarily and are called "narrow" values. The examples below show some 32-bit narrow value examples, their short form and their widths:

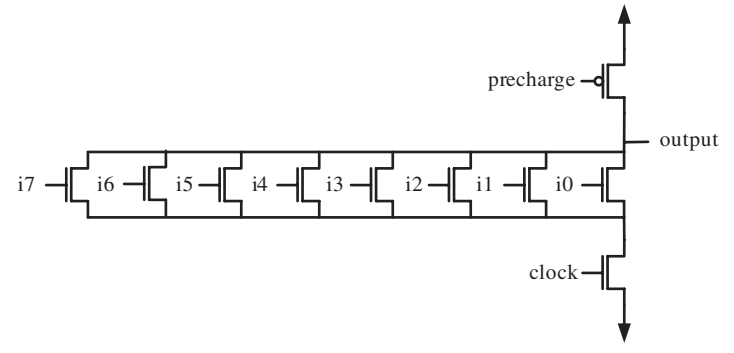
00000000000000000000000000000000	→ 0	(1 bit)
00000000000000000000000000000001	→ 01	(2 bits)
11111111111111111111111111111111	→ 1	(1 bit)
11111111111111111111111110001010	→ 10001010	(8 bits)

Note that the sign part of a narrow value is just compressed and it is expressed with a single bit. Repeating bits in the higher order part of the value can be removed by representing sign part with only one bit. In architectures whose datapath width is 64 bits, narrow values waste more space since the values that can be represented with a couple of bits will have to be stored using the full 64-bit storage space.

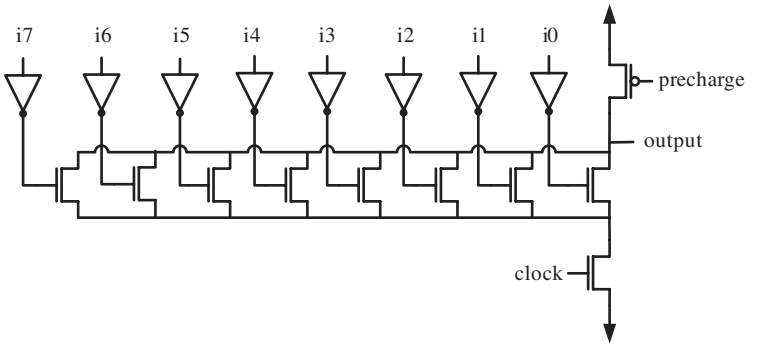
4.2. *Detecting narrow values*

Narrow values can be easily detected and exploited inside the datapath. In order to detect narrow values, leading zero and leading one detectors are used. These detectors can be implemented by using a simple NAND gate. However, because of fan-in and delay considerations, using dynamic logic is a more elegant solution.

Figure 4 shows the circuit diagram of consecutive zero and one detection circuits that are designed using dynamic logic. These circuits are in fact dynamic NAND gates. Circuit operation starts with the precharging phase at which the precharge signal goes low and the output node is precharged to V_{dd}. For consecutive zero



(a) 8-bit Consecutive Zero Detection Circuit



(b) 8-bit Consecutive One Detection Circuit

Fig. 4. (a) Consecutive zero and (b) one detection circuits. These detectors employ dynamic logic for fast operation.

detection circuit, at the evaluation phase, if any of the inputs is high, output is discharged with the clock signal and the circuit indicates that there are no consecutive zeros in the incoming value. If all of the inputs are zero, output is not discharged indicating that there are a series of zeros in the input.

Note that n-type transistors are used in both circuits since the output node has to be pulled down to ground and n-type transistors are better at passing ground signals. Therefore for consecutive one detector, instead of using p-type transistors, inputs are inverted.

When detecting narrow values using consecutive zero and one detectors, the size of the circuit grows as the size of the narrow value gets smaller. This is because of the fact that a narrower value has more unnecessary upper order bits.

Figure 5 shows the energy and delay characteristics of the consecutive zero detector circuit of Fig. 4(a). Detector circuits are optimized for best performance and the delay of detector circuits is measured for the worst case which occurs when the output node discharges only through a single transistor. Both the precharge and evaluation delays increase with increasing number of inputs since the total capacitance of the output node that needs to be charged and discharged increases.

There are two y-axis of the graph; the one on the left belongs to the energy dissipation data which is shown with a bar chart and the y-axis on the right side shows the scale for the delay data which are shown with line graphs. As the graph reveals, delay and energy dissipation of the detector circuits increase with increasing number of inputs.

For the circuit in Fig. 4(b) delay and energy figures are similar with the addition of a single inverter delay and possibly energy dissipation of multiple inverters. Including the inverters in the detector to detect leading ones does not effect precharging delay but increases evaluation time around 35 ps and energy dissipation by around 60 fJ for every possible input. If the inverted inputs are already available the inverters in Fig. 4 are unnecessary and do not increase the delay or energy dissipation of the detector circuit.

These two detectors have to be employed in parallel to generate the narrow value identification signal as shown in Fig. 6. After connecting leading zero and leading one detectors to the upper order 65-N bits of a value in a 64-bit datapath that defines a narrow value as a value that can be represented with N bits, outputs of the two detector circuits are ORed to generate the narrow value identification signal. This signal is 1 when the incoming value is narrow and it is 0 when the input value is larger than N bits.

Techniques proposed in this work are based on design time decisions. Therefore it is not necessary to count the number of zeros and ones that are unnecessary inside each and every value by using complex leading zero anticipators and leading zero detectors discussed in Ref. 30. Our technique relies on the common behavior of benchmarks. System only checks if the generated value is above or below the predefined narrow value width limits.

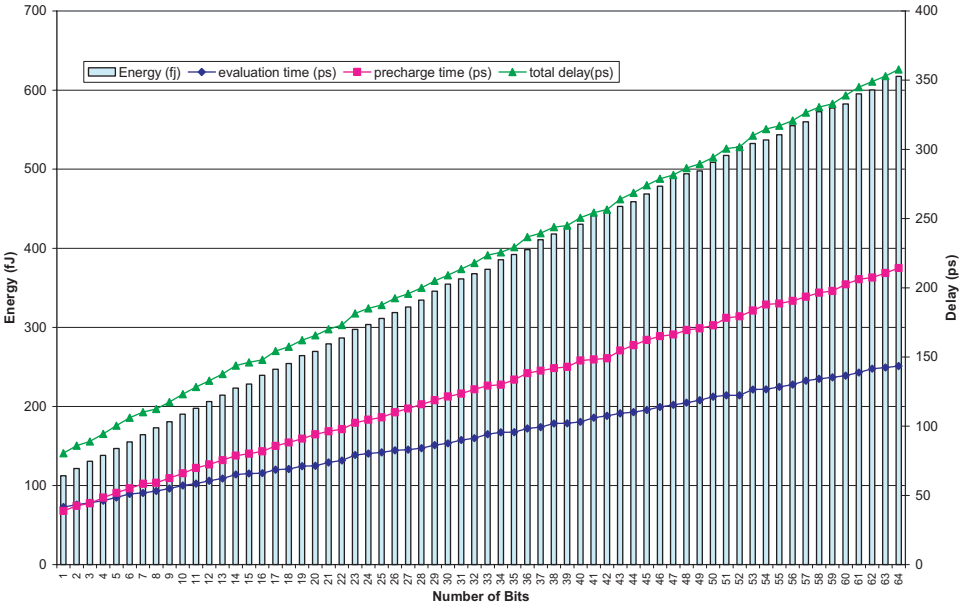


Fig. 5. Energy dissipation, evaluation delay, precharge delay and total delay (evaluation + precharge) of the consecutive zero detector circuit for different number of bits.

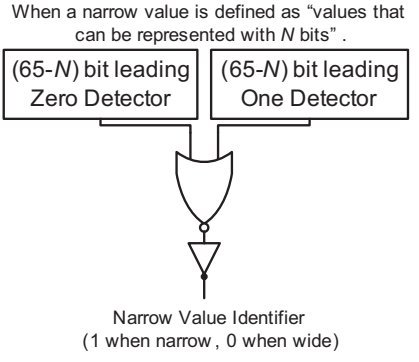


Fig. 6. Narrow value identification circuit. Signal is generated by ORing the output of two consecutive bit detectors.

4.3. *Partitioning the data structures for energy efficiency*

In this paper, we propose the partitioning of the integer data structures into two parts and only writing the significant part of narrow values inside the small partition. For this purpose we augment the data structures by 1 bit for each value and store the narrowness information inside this extra storage. If a value that needs to be written into the register file is detected to be narrow, the narrow identifier bit for the value is set and the upper order bits of the value is not written inside the

larger partition. Upon a read operation, narrow value identifier bit is checked and if a narrow value is indicated, upper order bits of the value are not read. Instead, a multiplexer is used in order to extend the sign bit of significant portion of the narrow value. If a wide value is stored inside the register, the narrow value identifier bit is reset to indicate that upon a read, instead of the sign of the lower order part of the stored value, upper order bits will be accessed.

Figure 7 shows the proposed storage structure for integer values. Storage space is divided into two halves consisting of the lower order bits and the upper order bits. In this work we assumed a 64-bit datapath since many processors today have a similar datapath width. If a narrow value is defined to hold n bits, the lower order part holds $n + 1$ bits including the "narrow value identifier" bit and the upper order part holds 64-n bits of the stored values. The partition that holds the lower order n bits is always used for reading and writing the significant portion of the value. The partition that holds the upper order 64-n bits is only used if the incoming value that needs to be written is wide. This way, by not reading

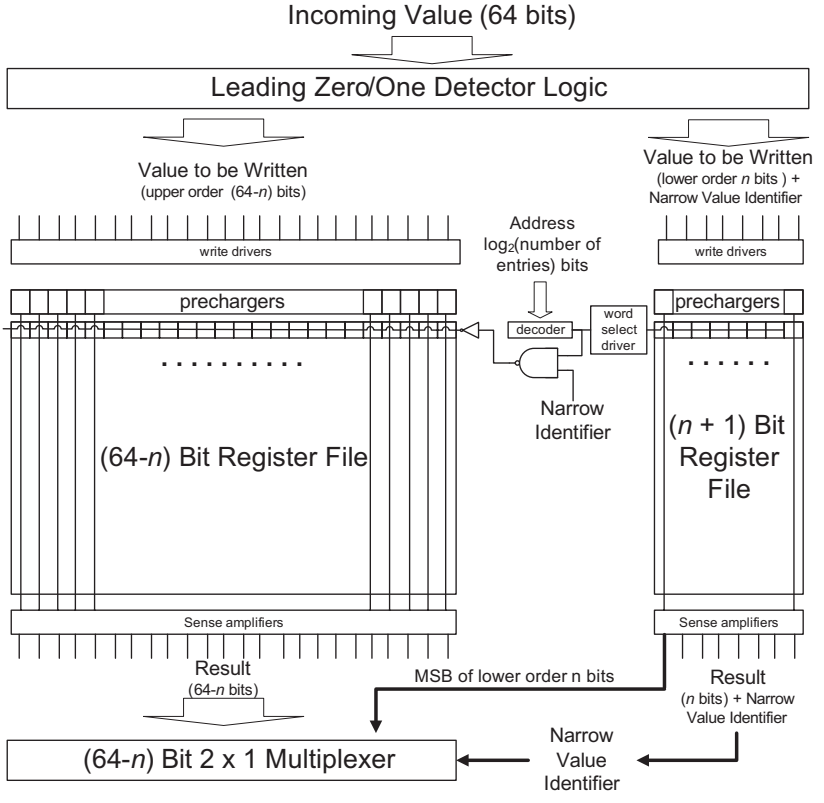


Fig. 7. Operand-width aware data storage structure. A Register file is divided into two partitions; a $(n + 1)$ bits wide partition for the significant portion of narrow value and a partition that holds the upper order 64-n bits of a value when necessary.

and writing the upper order parts of the narrow values energy dissipations can be achieved. High percentage of narrow values is needed to achieve high energy savings and justify the added power dissipation and complexity of the associated circuits.

Although the register file shown in Fig. 7 is divided into two parts, there is still only one decoder logic for each entry. When the decoder logic indicates a match, upper order bits of the register are activated only if the incoming value is not narrow (i.e., narrow identifier input is not zero). This narrow identifier signal is generated from two sources depending on the type of access. If the access type is a write, narrow identifier signal comes from the consecutive zero and consecutive one detectors. If the access is for a read operation, narrow identifier signal comes from the stored "narrow value identifier bit" (NVIB) of the entry. Narrow identifier input of the NAND gate does not require multiplexing since the write ports and the read ports of the register file are usually separated. For word select lines devoted for read operation the input of the NAND gate comes from the stored bit and for word select lines reserved for write operation, narrow identifier signal comes from the narrow value detector.

Energy savings are accomplished on a write operation by disabling the write driver circuits (that needs to drive the bit lines of the upper order bits of the stored value) and by disabling the local word select line of the partition that holds the upper order bits. When a value is read from the register file, energy is saved by not activating the local word select line of the partition that holds the upper order bits and by disabling the sense amplifiers that monitor the corresponding bit lines.

Accessing a narrow value mandates some changes on the register file. One (64-n) bit 2×1 multiplexer is needed now to choose between the stored upper order part of the value and the sign bit of the lower order part. If a stored value is narrow, the value is reconstructed by extending the sign bit (most significant bit) of the lower order part by using the multiplexer.

Figure 8 shows the 1-bit multiplexer circuit which is used to reconstruct the narrow values or read out the upper order bits of a wide value. (64-n) of these 1-bit multiplexer circuits are used in the 2×1 multiplexer block to construct the

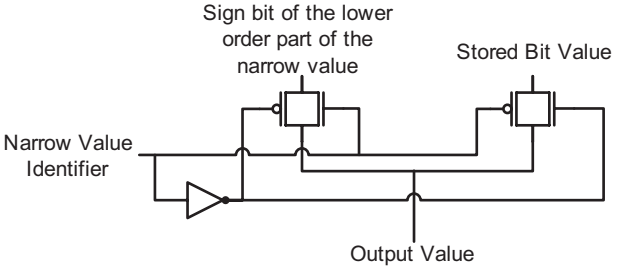


Fig. 8. 1-bit multiplexer circuit that chooses between the stored value and the sign bit of the lower order part. Circuit is implemented by using transmission gates for reducing delay overhead.

64-bit final value. CMOS transmission gates are used in the circuit to perform fast multiplexing.

There are obvious architectural trade-offs when defining the length of the narrow values. For maximum energy efficiency the size of the narrow values has to be smaller since with narrower values, read and write operations on larger portions of the value can be avoided. On the flip side, defining a narrow value to have too few bits, results in a low percentage of narrow values inside the program and lower energy savings. If most of the values are identified as wide values, partitioning of the register file will not save energy and may even result in increased energy dissipation. Therefore there is an optimum point for the definition of narrow values where the partition that holds the lower order part of the values is not too large and the percentage of narrow values is significant for best energy efficiency.

On the other hand, there are also circuit level trade-offs regarding the length of the narrow value. Delay overhead of the consecutive zero and one detectors are dependent on the size of the narrow values as shown in Fig. 5. As the size of the narrow value decreases, number of inputs for consecutive zero and one detectors increase. Therefore delay and power dissipation of detector circuits increase as the narrow values are defined to include fewer bits. This circuit level property is another side effect of defining narrow values that include too few bits.

5. Simulation Methodology

In order to get an accurate idea of the percentage of narrow values inside current ×86 processors, we used the PTLsim simulator[36] which is capable of executing 64-bit ×86 instructions. We executed all of the programs in SPEC 2000 benchmark suite and 12 programs in SPEC 2006 benchmark suite (we were not able to execute the rest of the programs in SPEC 2006) on the simulator to observe the percentage of the narrow values in different workloads. All benchmarks were compiled using gcc 3.4.3 with maximum optimizations to target the 64-bit ×86-64 instruction set. Results were gathered by running the first two billion instructions in each benchmark. Table 1 shows the configuration of the simulated architecture.

For estimating energy dissipation of the data storage structures, event counts from the simulator were combined with the energy dissipations measured from actual full custom CMOS layouts of the register file. For SPICE simulations we used BSIM model 4.5.0.[5]

6. Results and Discussions

Percentage of narrow values is crucial for energy savings in partitioning the register file according to operand width. If the percentage of narrow values is very low, energy dissipation of the register file can even be increased due to the energy dissipation of the added circuitry. However, operand widths in common executed workloads are generally small as observed and reported by many researchers.[11, 23]

Table 1. Configuration of the simulated processor.

Parameter	Configuration
Machine width	4-wide fetch, 4-wide issue, 4 wide commit
Window size	64 entry issue queue, 64 entry load/store queue, 128-entry ROB, 128 integer registers (unified reg file), 128 floating point registers
Function units and latencies (total/issue)	4 Int Add (1/1), 1 Int Mult (3/1)/Div (20/19), 2 Load/Store (2/1), 2 FP Add (2), 1FP Mult (4/1)/Div (12/12)/Sqrt (24/24)
L1 I-cache	32 KB, 2-way set-associative, 64 byte line, 1 cycles hit time
L1 D-cache	64 KB, 4-way set-associative, 64 byte line, 2 cycles hit time
L2 Cache unified	2 MB , 8-way set-associative, 128 byte line, 6 cycles hit time
BTB	4 K entry, 2-way set-associative
Branch Predictor	Combined with 1 K entry Gshare, 8-bit global history, 4 K entry bimodal, 1 K entry selector
Memory	256-bit wide, 300 cycles first part, 1 cycle interpart
TLB	32 entry (I) — 2-way set-associative, 128 entry (D) — 16-way set associative, 12 cycles miss latency

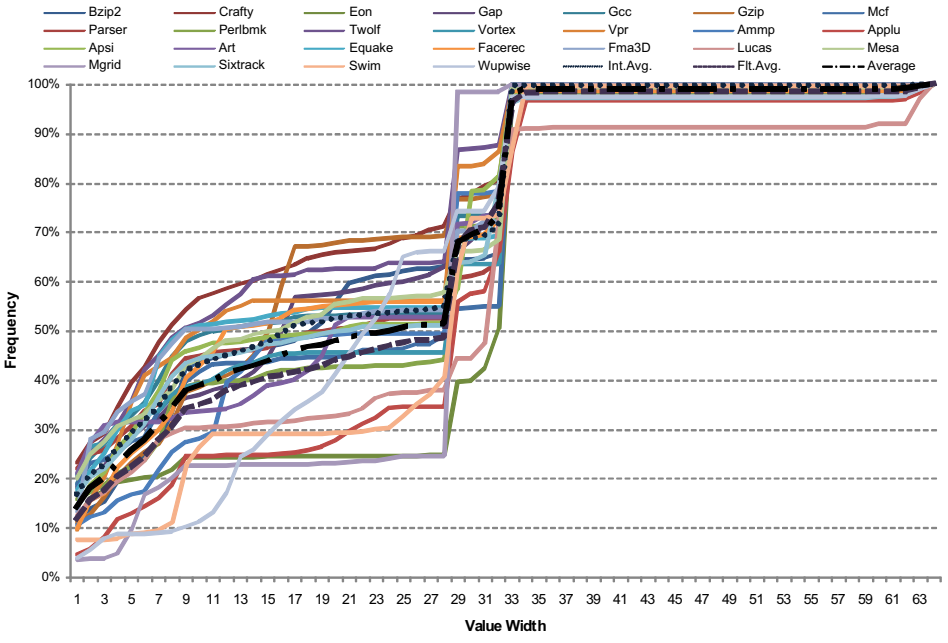
6.1. *Integer register file*

Integer register file is the major integer data holding structure inside a processor. The unified register file holds both the architectural state and the temporary values generated by the function units.

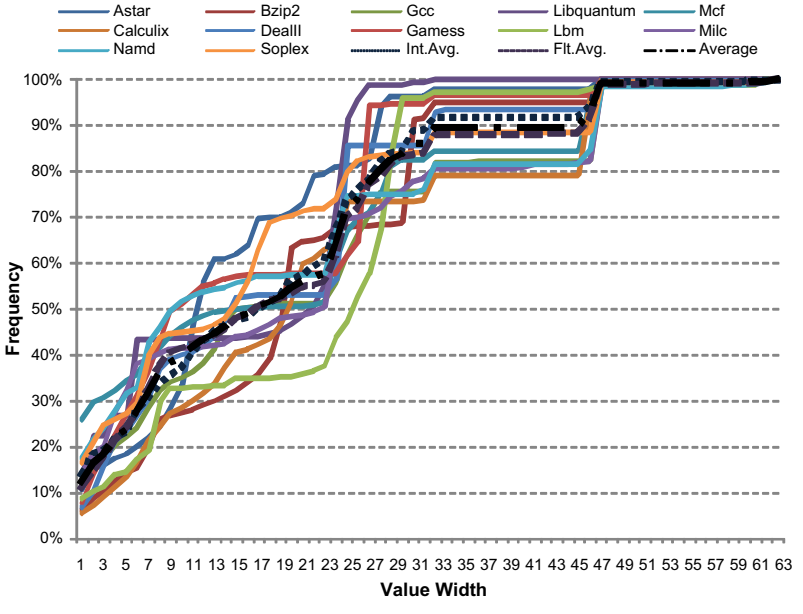
Figure 9 shows the frequency of the width of the values that are written back to the register file for spec 2000 and spec 2006 benchmarks. Each line belongs to a separate benchmark and for every value width the y-axis shows the percentage of generated values smaller or equal to that value width. As the graphs reveal, the percentage of narrow values in these workloads is quite high. Many of the spec 2000 benchmarks defining the width of the narrow values to include more than 34 bits usually cover most of the generated values. The graph is shown in Fig. 9(b) shows that spec 2006 benchmarks have larger value widths than spec 2000 benchmarks on average. Although 34 bits still cover a large percentage of values, 48 bits seems to be covering almost all the values generated in spec 2006 benchmarks.

In order to calculate the energy savings obtained by using an operand-width aware register file, we combined the energy dissipation figures obtained from SPICE simulations with the event counts gleaned from our microarchitectural simulator. Energy dissipation of the register file components as well as the energy dissipation of all the additional circuitry was included in the calculation to get an accurate estimate of the energy savings.

Figure 10 shows the energy reduction in the register file by partitioning the structure. Each line shows the energy savings figure for a separate benchmark and the x-axis shows the width of the partition that holds the lower order bits of the stored value. Depending on the value width characteristics of the benchmarks, different energy savings are achieved for different narrow value definitions. Benchmarks which have a lot of very small narrow values (such as mgrid with 50–75% energy reduction when narrow value is defined to include 29 bits) show highest energy savings

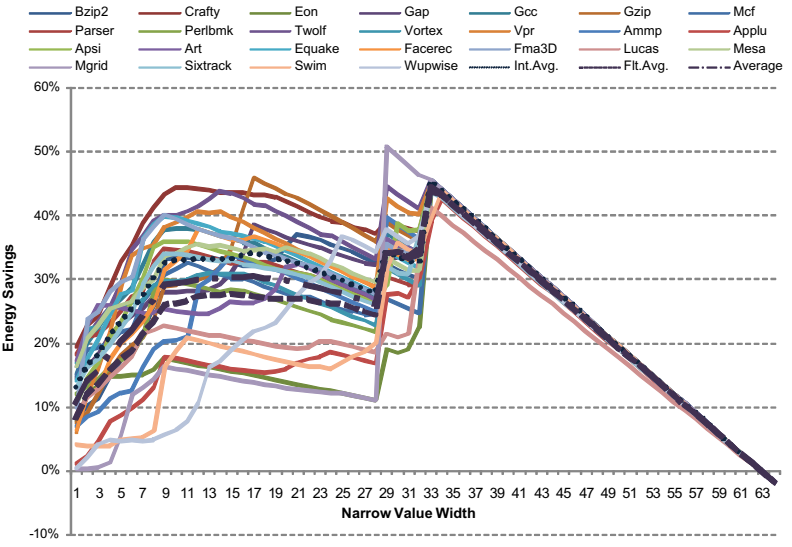


(a) SPEC 2000 Benchmarks

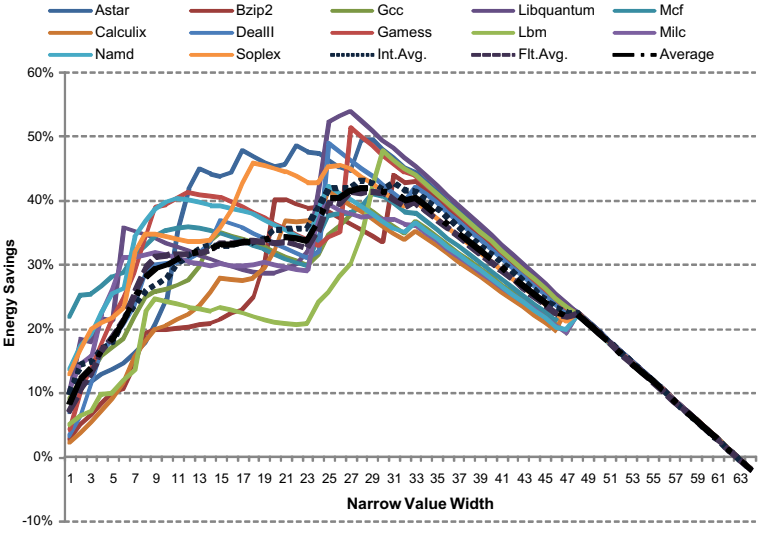


(b) SPEC 2006 Benchmarks

Fig. 9. Width histogram of the values written to the integer register file for SPEC 2000 benchmarks. For each benchmark, a separate line shows what percentage of the operands have smaller width than that is indicated by the number on the x-axis.



(a) SPEC 2000 Benchmarks



(b) SPEC 2006 Benchmarks

Fig. 10. Energy savings in the integer register file when the structure is partitioned to process narrow values more efficiently. Results for each benchmark are shown with a separate line graph and the average is indicated with a bold dashed line.

numbers. Whereas for some other benchmarks, in which medium size values dominate, defining narrow values larger yields the best energy efficiency. Highest energy savings (44.5%) with the proposed partitioning technique is achieved by defining narrow values with 33 bits on the average across all spec 2000 benchmarks.

Energy savings achieved for each benchmark varies. There are multiple peaks and jumps in the graph due to the cumulative increase of the frequency of narrow values inside the program with increasing narrow value widths. For example if the percentage of 33-bit values is very high, defining the narrow values with 32 bits miss the opportunity of including many of the values inside the narrow class by just one bit. On the right side and the middle of the left side of the graph there are linearly decreasing slopes. The reason for these slopes can be understood by examining Fig. 9. As Fig. 9 indicates, this behavior of the graph in Fig. 10 is because of unchanging percentage of narrow values for corresponding narrow value widths. For most of the benchmarks, there are not many values that can be expressed with narrow value widths between 34 and 62.

Individual benchmarks have different energy savings results. Some benchmarks show multiple peaks and maximum energy reduction points in the graph. For example, gzip has two peak values on the chart. Energy dissipation of the register file is reduced by 45.92% by defining the narrow value width as 17 when running gzip. Also 45.45% energy reduction in the register file is achieved by defining narrow value width as 33. In this kind of behavior it always makes sense to define the narrow values larger (33 for gzip) since the delay of the detector circuits are reduced because of the aforementioned reasons described in Sec. 3.2.

Figure 10(b) shows the energy reduction results with a partitioned register file when spec 2006 benchmarks are run on the simulator. Unlike the spec 2000 benchmark average, defining narrow values as the values that can be represented with 29 bits achieves the highest energy savings for spec 2006 programs. Different benchmarks show different energy savings behavior. For example, astar has four peak values at 13, 17, 22 and 29-bit narrow values. As stated previously, selecting the wider narrow value definition is more beneficial at the circuit level if these choices provide similar architectural energy effiency benefits. On the average across all spec 2006 benchmarks, 42% energy reduction is achieved in the integer register file when the narrow values are defined to include 29 bits.

Our studies show that, when programs like spec 2000 benchmarks are run on a processor, defining the narrow value width as 33 is the optimal solution for the integer register file, whereas for spec 2006 benchmarks 29 bits seems to be a better choice. This is generally because of the fact that large number of values can be represented with 29 or 33 bits. The jumps in the graph on these bits are usually caused by stack and frame pointers inside the processors. Each benchmark has its own behavior and its own optimum point. Designers should do the same analysis on the workload of a processor before defining the width of narrow values and the size of the partitions.

Energy savings lines in the charts for each benchmark get a negative value when narrow values are defined with more than 62 bits. This is because of the energy overhead of the additional logic and the stored narrow identifier bit. 64-bit narrow value actually means a wide value that is not compressed at all but the additional narrow identifier bit is always written.

6.2. *Immediate field of the issue queue*

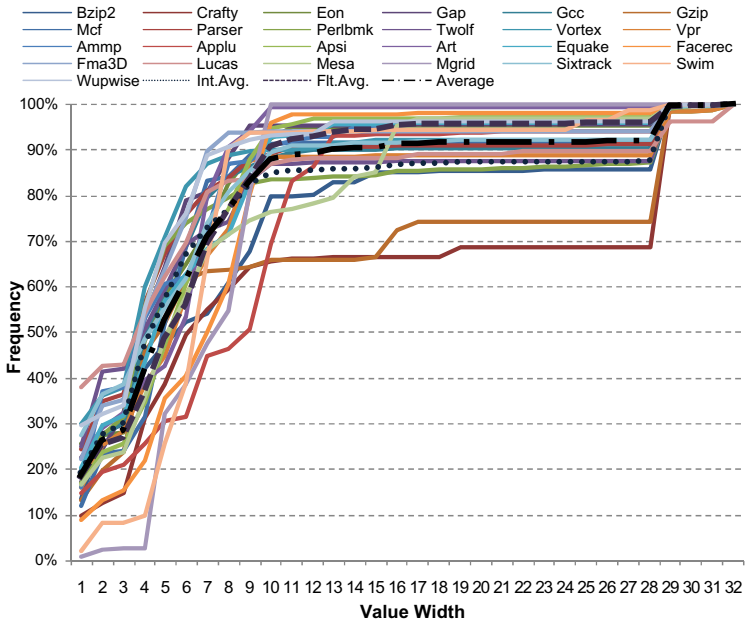
64-bit × 86 architecture allows up to 32-bit wide immediate values which mandates the use of a 32-bit wide storage space for each instruction that is stored inside the issue queue.[39] Figure 11 shows the width histogram of the immediate values written to the issue queue for spec 2000 benchmarks. As the graph reveals, programs very rarely use the full width of the immediate storage space. This results in a steep increase in the graph as the value width approaches to 12 bits. Similar behavior is depicted for spec 2006 benchmarks where immediate fields are slightly more utilized with wider operands. Nevertheless 25 bits seems to be covering almost all of the immediate values on the average across all spec 2006 benchmarks.

The large percentage of narrow values depicted in Fig. 11 results in a larger energy reduction in the immediate field of the issue queue when compared to the achieved energy reduction in the integer register file. Figure 12 shows the energy reduction achieved for spec 2000 and spec 2006 benchmarks in the immediate field of the issue queue by using operand width partitioning. On the average across all spec 2000 benchmarks, 56.52% energy reduction is achieved when narrow values are defined to include 10 bits inside the immediate field. Specific benchmarks show even larger energy savings; for example 64.49% energy reduction is achieved inside the immediate field for mgrid. For spec 2006 benchmarks, results are slightly different from that of spec 2000 benchmarks as the maximum achieved energy reduction on the average is 53.74% and it is achieved at 8 bits.

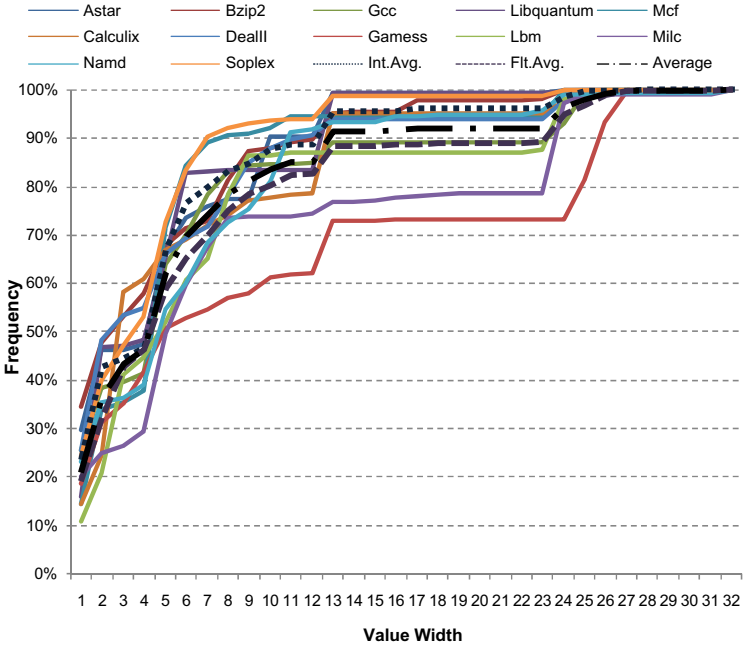
6.3. *Floating point register file*

The format of the values stored inside the floating point register file is different from that of the integer storing components since the stored values have 3 different components: sign, exponent and the fraction. Because of the format of floating point values simple narrow value identification results in very wide narrow value definitions and eventually smaller energy efficiency. Therefore before we applied any partitioning on the floating point register file we analyzed the frequency of the special cases such as “zero”, “not a number (NaN)” and ±infinity.

Figure 13 shows the frequency of different value types written to the floating point register file. 13 bars on the left belong to the floating point benchmarks of spec 2000 and the 7 bars in the middle belong to the floating point benchmarks of spec 2006. Since the integer benchmarks do not contain many floating point values we did not consider them for this experiment. As the chart reveals, around 50% of the produced floating point values in spec 2006 benchmarks and 40% of the produced floating point values in spec 2000 benchmarks indicate a zero value. Except for some benchmarks (such as facerec, lucas and mesa) NaN and infinity results are not very dominant in the produced results. It is clear from Fig. 13 that by just adding a bit for each floating point register to indicate that the stored value is zero leads to an almost 50% reduction in register file energy dissipation for spec 2006 benchmarks if the energy dissipation of the extra bit and the logic is neglected.

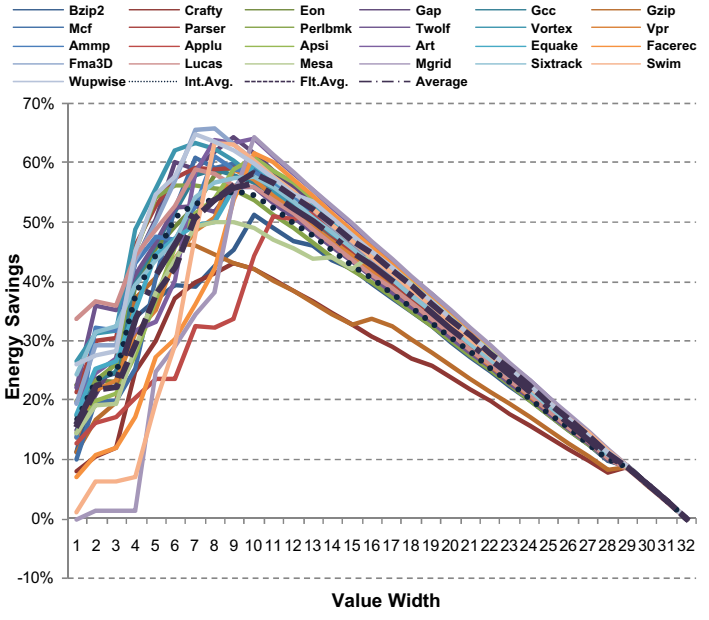


(a) SPEC 2000 Benchmarks

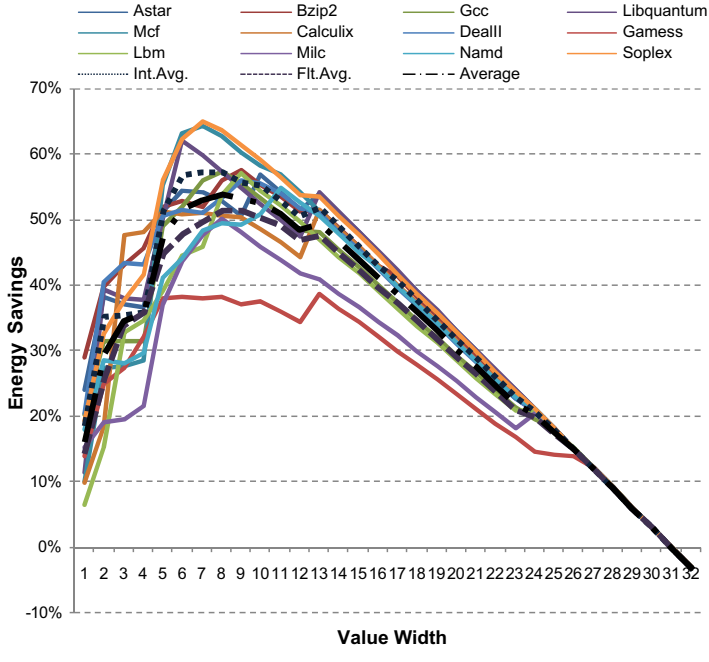


(b) SPEC 2006 Benchmarks

Fig. 11. Width histogram of the immediate values written to the issue queue for SPEC 2000 and SPEC 2006 benchmarks.



(a) SPEC 2000 Benchmarks



(b) SPEC 2006 Benchmarks

Fig. 12. Energy reduction achieved in the immediate field of the issue queue by operand width aware partitioning for SPEC 2000 and SPEC 2006 benchmarks.

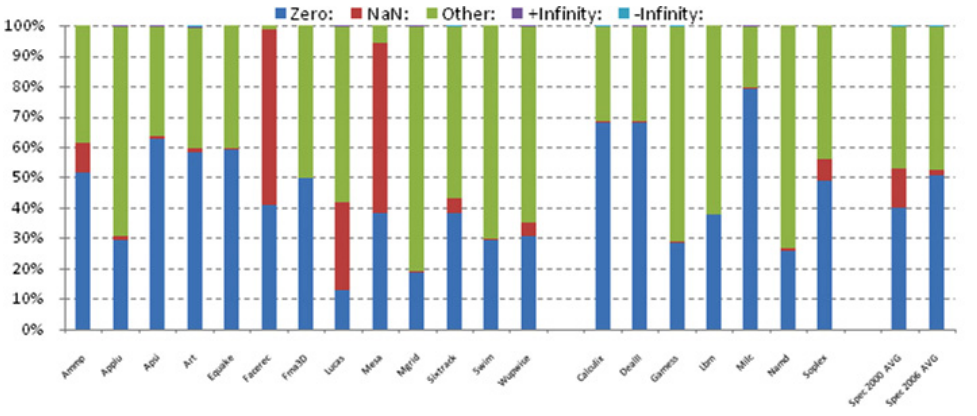


Fig. 13. Frequency of the special cases for the value written to the floating point register file.

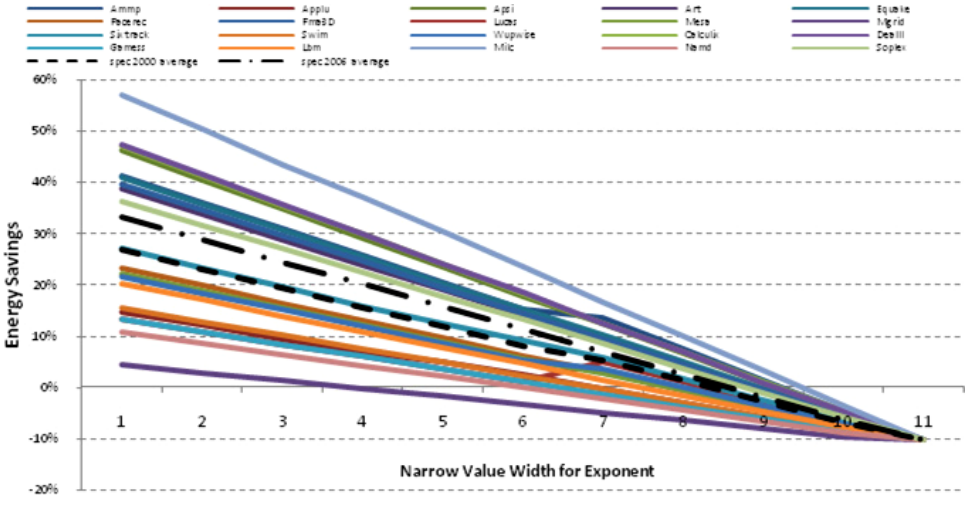


Fig. 14. Energy reduction achieved in the exponent field of the floating point register file by operand width aware partitioning for spec 2000 and spec 2006 floating point benchmarks.

Instead of looking at the floating point values as a whole we analyzed the data patterns for fraction and exponent field separately and calculated the energy savings that can be achieved by separately partitioning these fields. Figures 14 and 15 show the energy reduction by operand width aware partitioning of the exponent and the fraction fields respectively. Both charts show that almost for all benchmarks, most of the contents can be expressed with only a single bit. This result is in line with Fig. 14 since almost half of the written floating point values are zero. Therefore instead of putting a narrow value identifier bit for floating point values, it makes more sense to add a bit to indicate that the incoming value is a zero.

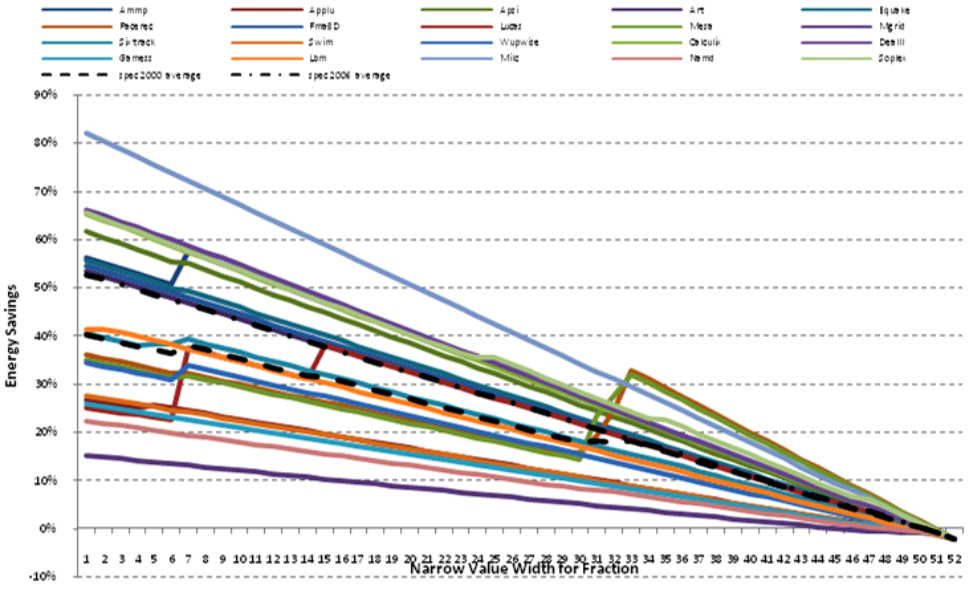


Fig. 15. Energy reduction achieved in the fraction field of the floating point register file by operand width aware partitioning for spec 2000 and spec 2006 floating point benchmarks.

Some of the spec 2000 benchmarks show high percentage of NaN values which may be exploited for more energy reduction by including another bit to identify NaN values. However spec 2006 benchmarks do not show this behavior. Therefore adding a NaN value identifier bit is not generally beneficial for all benchmarks and the designer has to examine the workloads that will be run on the processor in order to make a decision to add such a bit or not.

7. Conclusions and Future Work

Most of the stored values in a processor requires less than full width provided by the datapath. Upper order bits of these values are unnecessary and are written to and read from the storage components of the processor unnecessarily. In this work we proposed to determine the optimum narrow value width and partition the data storage structures into two partitions. Each narrow value is written to the partition that holds the lower order bits and the sign bit of this lower order part of the value is later extended to reconstruct the value when it is sought.

Operand-width aware partitioning of the integer register file reduces the energy dissipation by around 44.5% on average across all spec 2000 benchmarks on an $\times$86 machine when the narrow values are defined as values that can be represented with 33 bits. Energy savings of individual benchmarks vary; it is possible to achieve energy savings of as high as 50.75% with some benchmarks. Similarly for spec 2006 benchmarks 42% energy reduction is achieved in the integer register file with a

29-bit narrow value definition although the Libquantum benchmark achieved 53.97% energy reduction with 27 bits.

Besides the register file we also analyzed the immediate field of the issue queue where up to 32-bit-wide immediate operands are stored. We showed that most of the immediate operands that come from the programs use a very small part of the 32-bit immediate field, and by using operand width aware partitioning it is possible to reduce the energy dissipation of the immediate field by 56% for spec 2000 benchmarks for 10-bit narrow values and 54% for spec 2006 benchmarks for 8-bit narrow values on the average. Specific benchmarks show a possible energy reduction of as high as 64% in the immediate field of issue queue.

Floating point values are stored differently and follow different patterns. We showed that most of the values that are written to the floating point register file are zero. We also showed that for some benchmarks NaN operands are also significant. From our findings we concluded that dividing the storage structure into partitions according to operand width is not a suitable solution for floating point register file but instead zero values can be identified with a single bit to achieve almost 50% energy reduction in the floating point register file.

Apart from the architectural trade-offs of identifying narrow values we also showed that there are some circuit level trade-offs in the associated circuitry. We showed that when two different narrow value width definitions result in similar energy reductions, it is always better to choose the wider narrow value definition to reduce the delay of the detector circuits. The operand width aware structure partitioning technique described in this paper can be applied to other data holding components of a processor such as the data cache or the writeback latches.

In this study we offered a method to reduce the dynamic energy dissipation of the data holding components. However, these components also dissipate energy even when there is no activity on them due to the leakage currents which holds an important part of the total energy dissipation of the contemporary microprocessors. Exploiting narrow values can also be used to reduce the static energy of the register file by using sleep transistors on the upper order bits of the register file and turning them off when the stored value is narrow. Exploiting narrow values for static energy reduction through reducing the leakage current is left for future work.

Acknowledgments

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) through research grant number 107E043.

References

1. A. Aggarwal and M. Franklin, Energy efficient asymmetrically ported register files, *Proc. Int. Conf. Computer Design (ICCD)* (2003).
2. R. Balasubramonian, S. Dwarkadas and D. Albonesi, Reducing the complexity of the register file in dynamic superscalar processor, *Proc. Int. Symp. Microarchitecture (MICRO-34)* (2001).

3. E. Borch, E. Tune, S. Manne and J. Emer, Loose loops sink chips, *Proc. Int. Conf. High Performance Computer Architecture (HPCA-8)* (2002).
4. D. Brooks and M. Martonosi, Dynamically exploiting narrow width operands to improve processor power and performance, *Proc. HPCA* (1999).
5. BSIM 4.5.0 Manual, http://www-device.eecs.berkeley.edu/∼bsim3/bsim4.html.
6. A. Butts and G. Sohi, Use-based register caching with decoupled indexing, *Proc. Int. Symp. Computer Architecture* (2004).
7. R. Canal, A. Gonzales and J. Smith, Very low power pipelines using significance compression, *Proc. Int. Symp. Microarchitecture* (2000).
8. R. Canal, A. Gonzalez and J. Smith, Software-controlled operand gating, *Proc. Int. Symp. Code Generation and Optimization* (2004).
9. Y. Cao and H. Yasuura, A system-level energy minimization approach using datapath width optimization, *ISLPED* (2001).
10. J.-L. Cruz *et al.*, Multiple-banked register file architecture, *Proc. Int. Symp. Computer Architecture (ISCA-27)* (2000), pp. 316–325.
11. O. Ergin *et al.*, Register packing: Exploiting narrow-width operands for reducing register file pressure, *MICRO* (2004).
12. O. Ergin, O. Unsal, X. Vera and A. González, Exploiting narrow values for soft error tolerance, *IEEE Comput. Archit. Lett.* **5** (2006).
13. R. Gonzalez *et al.*, A content aware register file organization, *ISCA* (2004).
14. R. Gonzalez *et al.*, An asymmetric clustered processor based on value content, *ICS* (2005).
15. G. Hinton *et al.*, The microarchitecture of the Pentium 4 Processor, *Intel Technol. J.* **Q1** (2001).
16. J. Hu *et al.*, In-register duplication: Exploiting narrow-width value for improving register file reliability, *DSN* (2006).
17. J. Keshava and V. Pentkovski, Pentium® III processor implementation tradeoffs, *Intel Tech. J.* **3** (1999).
18. R. E. Kessler, The Alpha 21264 microprocessor, *IEEE Micro.* **19** (1999) 24–36.
19. N. Kim and T. Mudge, Reducing register ports using delayed write-back queues and operand pre-fetch, *Proc. Int. Conf. Supercomputing (ICS-17)* (2003).
20. M. Kondo and H. Nakamura, A small, fast and low-power register file by bit-partitioning, *HPCA* (2005).
21. S. Kumar, P. Pujara and A. Aggarwal, Bit-sliced datapath for energy-efficient high performance microprocessors, *Proc. 4th Workshop on Power Aware Computer Systems (PACS'04)*, Portland, Oregon, USA, December 2004.
22. M. Lipasti, B. R. Mestan and E. Gunadi, Physical register inlining, *ISCA* (2004).
23. G. Loh, Exploiting data-width locality to increase superscalar execution bandwidth, *Proc. Int. Symp. Microarchitecture* (2002).
24. G. Loh, Width prediction for reducing value predictor size and power, in *First Value Prediction Workshop*, *ISCA* (2003).
25. M. Moudgill, K. Pingali and S. Vassiliadis, Register renaming and dynamic speculation: An alternative approach, *Proc. Int. Symp. Microarchitecture (MICRO-26)* (1993), pp. 202–213.
26. T. Nakra *et al.*, Width sensitive scheduling for resource constrained VLIW processors, *Workshop on Feedback Directed and Dynamic Optimizations* (2001).
27. I. Park, M. Powell and T. Vijaykumar, Reducing register ports for higher speed and lower energy, *Proc. Int. Symp. Microarchitecture (MICRO-35)* (2002).
28. D. Ponomarev, G. Küçük, O. Ergin, K. Ghose and P. M. Kogge, Energy efficient issue queue design, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **11** (2003) 789–800.

29. T. Sato and I. Arita, Table size reduction for data value predictors by exploiting narrow width values, *Proc. Int. Conf. Supercomputing* (2000).
30. M. S. Schmookler and K. J. Nowka, Leading zero anticipation and detection — A comparison of methods, *15th IEEE Symp. Computer Arithmetic (ARITH-15 '01)* (2001), p. 0007.
31. M. Stephenson *et al.*, Bitwidth analysis with application to silicon compilation, *Proc. PLDI* (2001).
32. O. Unsal, O. Ergin, X. Vera and A. Gonzalez, Empowering a helper cluster through data width aware instruction steering policies, *Proc. 20th Int. Parallel and Distributed Processing Symposium (IPDPS-20)*, Rhodes, Greece, April 2006.
33. L. Villa, M. Zhang and K. Asanovic, Dynamic zero compression for cache energy reduction, *Micro-33*, December 2000.
34. S. Wallase and N. Bagherzadeh, A scalable register file architecture for dynamically scheduled processors, *Proc. Int. Conf. Parallel Architectures and Compilation Techniques (PACT-5)* (1996).
35. K. Yeager, The MIPS R10000 superscalar microprocessor, *IEEE Micro*, Vol. 16, April 1996.
36. M. T. Yourst, PTLsim user's guide and reference: The anatomy of an x86-64 out of order microprocessor, Technical report, www.ptlsim.org.
37. S. Wang, H. Yang, J. Hu and S. G. Ziavras, Asymmetrically banked value-aware register files for low energy and high performance, *Microprocessors and Microsystems* **32** (2008) 171–182.
38. D. Brooks, V. Tiwari and M. Martonosi, Wattch: A framework for architectural-level power analysis and optimizations, *ACM SIGARCH Computer Architecture News*, Vol. 28, May 2000, pp. 83–94.
39. Intel® 64 and IA-32 Architectures Software Developer's Manual, http://developer.intel.com/products/processor/manuals/index.htm.
40. O. Ergin, Exploiting narrow values for energy efficiency in the register files of superscalar microprocessors, *16th Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'06)*, Springer-Verlag, September 2006, pp. 477–485.
41. Y. Osmanlioglu, Y. O. Kocberber and O. Ergin, Reducing parity generation latency through input value aware circuits, *Proc. Great Lakes VLSI Symp. (GLSVLSI'09)*, 2009.