Appears in the 8th International Symposium on High-Performance Computer Architecture (HPCA-8), Cambridge, MA, Feb. 2–6, 2001

Using Internal Redundant Representations and Limited Bypass to Support Pipelined Adders and Register Files

Mary D. Brown   Yale N. Patt

Electrical and Computer Engineering
The University of Texas at Austin
{mbrown,patt}@ece.utexas.edu

Abstract

This paper evaluates the use of redundant binary and pipelined 2's complement adders in out-of-order execution cores. Redundant binary adders reduce the ADD latency to less than half that of traditional 2's complement adders, allowing higher core clock frequencies and greater execution bandwidth (in instructions per second). Pipelined 2's complement adders allow a higher clock frequency, but do not reduce the ADD latency. Machines with redundant binary adders are compared to machines with 2's complement adders and the same execution bandwidth and bypass network complexity. Results show that on the SPECint95 benchmarks, the average IPC of an 8-wide machine with 1-cycle redundant binary adders is 9% higher than a machine using 2-cycle pipelined adders.

Pipelined functional units and multi-cycle register files may require multi-level bypass networks to guarantee that an instruction's result is available any cycle after it is produced. Multi-level bypass networks require large fan-in input muxes that increase cycle time. This paper shows that one level of bypass paths in a multi-level bypass network can be removed while still achieving within 3% to 1% of the IPC of a machine with a full bypass network.

1. Introduction

Future microprocessors require greater execution bandwidth for higher performance. The first step towards increasing the bandwidth is to reduce the ALU latency—along with the things that feed the ALUs, such as the scheduling logic and bypass networks—so that the execution core can be clocked at higher frequencies. Taking this first step increases execution bandwidth and reduces the latency for executing chains of dependent instructions. In the Intel Pentium 4, the core clock frequency was set by the ALU and bypass network latency [10]. Other parts of the chip, such as the fetch engine, could provide the required execution bandwidth at lower clock frequencies.

To further increase the execution bandwidth, it is necessary to pipeline the functional units or increase the number of functional units. The three execution core configurations shown in Figure 1 all provide an execution bandwidth of 2 instructions per cycle. Configuration A shows 2 ALUs, each with 1-cycle latency. Configuration B shows 2 ALUs, each pipelined over 2 cycles. Dependent instructions cannot execute in back-to-back cycles in this configuration. Configuration C also shows 2 ALUs pipelined over 2 cycles, but it allows intermediate results to be forwarded from the first stage of the ALU. This allows a dependent chain of instructions to execute in consecutive cycles.

In each configuration, the bypass mux for one input of one ALU is shown. The ALU inputs for Configurations A and B can receive data from the register file or the outputs of either ALU. The ALUs in Configuration C can receive the data from the register file or either stage of either ALU. The outputs of Stages 1 and 2 are used as inputs to Stage 1, but only the outputs of Stage 2 are written back to the register file. For one-cycle operations, forwarding from Stage 1 is necessary so that dependent operations can execute in consecutive cycles. For multi-cycle operations, the output from Stage 1 is an intermediate result. Because results are available from multiple stages but only written to the register file in Stage 2, a multi-level bypass network is used.

Programs with a large amount of exposed ILP can exploit high execution bandwidth. Programs with little exposed ILP benefit from low execution latency. Given a constant cycle time, all three configurations provide the same bandwidth. Configuration A is the best as it provides low latency. Configuration B has a long latency. Configuration C has long latency for final results, but also provides low-latency intermediate results.

Our results show that for machines with an execution bandwidth of 8 instructions per cycle, an ideal machine using 1-cycle adds will have an average IPC 8% higher than

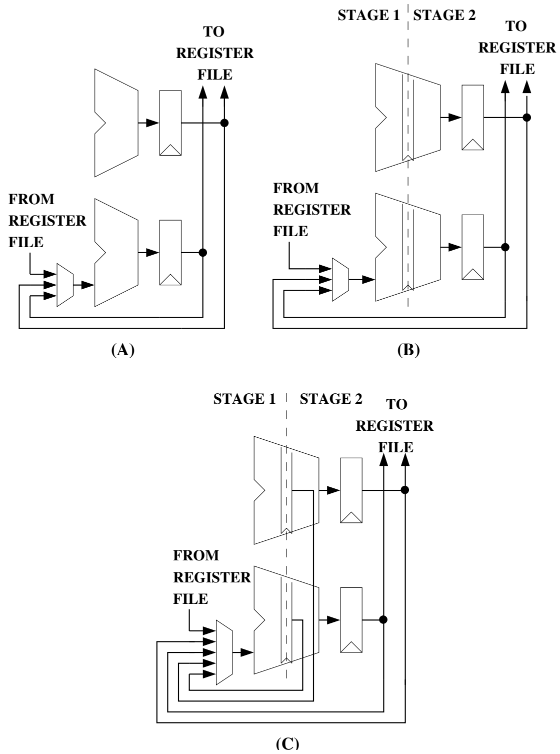

Figure 1. Three ALU configurations.

a machine with 2-cycle adds on the SPECint2000 benchmarks, and 11% higher on the SPECint95 benchmarks. A machine with redundant binary adders and the same bypass network complexity can be used to reach within 1% of the IPC of the ideal machine. Because the multi-level bypass networks for pipelined adders and register files may increase the cycle time, this paper also investigates limited bypass networks and the schedulers to support them. One level in a multi-level bypass network can be removed while still achieving within 3% to 1% of the IPC of a machine with a full bypass network.

Section 2 discusses some background and related work. Section 3 reviews redundant binary arithmetic and discusses which operations can be performed with redundant binary inputs. Section 4 discusses execution cores with redundant binary adders and limited bypass networks. Section 5 shows experimental results, and Section 6 concludes.

2. Background and Related Work

One technique for pipelining adders is to use pipelined digit-serial adders [6]. An example of this concept, called *staggered adds*, was implemented in the Intel Pentium 4 [10]. When staggering a 32-bit add over two cycles, the carry-out of the 16th bit and the lower half of the result are produced in the first cycle, and the upper half of the result is produced in the second cycle. Back-to-back dependence execution is possible by forwarding the lower half of the result and its carry-out in the first stage, and then forwarding the second half of the result in the second stage. Conventional 2's complement carry-lookahead adders have a critical path that grows logarithmically with respect to the number of bits in the data. Using a 2-stage, staggered adder is unlikely to cut the effective add latency in half unless the latency is set by control signal propagation.

Redundant binary arithmetic can be used to further reduce the effective ALU latency. When integers are in a redundant binary representation, they can be added without a carry propagation through the entire length of the data operands. Hence redundant binary adders have a latency that is independent of the data size, and a much shorter critical path than 2's complement adders. Data can be forwarded between dependent ADDs in redundant binary representation. A redundant binary number must be converted back to 2's complement before it can be stored in memory or used by some types of instructions.

Redundant binary arithmetic has been a steady area of computer arithmetic research since the 1950's [3]. Although the ILLIAC III used a redundant binary adder-subtractor for fast multiplication and division [2], redundant binary arithmetic has mainly been used in adders that are internal to hardware multipliers and dividers [12]. The reason for its limited use is that results of redundant binary operations must be converted back to 2's complement using a conventional (slow) adder with a full carry-propagation. Hence redundant binary adders only show a performance improvement relative to 2's complement adders when these format conversions can be avoided or moved off the critical path of program execution. Conversions can be avoided when executing a chain of dependent redundant binary operations and forwarding the intermediate results in redundant binary representation [7].

Multi-cycle register files and pipelined functional units that produce intermediate results require extra levels of bypass buses for data to be available any cycle after it is computed [5, 17]. Larger, multi-level bypass networks will have longer forwarding delays. There are several techniques to reduce forwarding delays. One technique is to interleave the bit slices of several functional units within a cluster. The functional units, input multiplexors, and latches within a cluster are stacked and aligned such that for each bit slice, the data wires of all functional units are adjacent.

Another technique to reduce forwarding delays is to remove selected bypass paths altogether. Ahuja et. al. [1] demonstrated that certain bypass paths were rarely used. By removing these buses from an in-order pipeline and scheduling code to avoid pipeline stalls, performance was close to that of a pipeline with a complete bypass network. On the VIPER VLIW microprocessor [8], each functional unit had a bypass path to only itself and its closest functional unit in order to reduce the bypass network complexity. Nagarajan et. al. [13] use an array of ALUs with bypass paths only to nearby ALUs. Instructions are statically assigned to ALUs such that forwarding delays are reduced.

Cruz et. al. [5] examine removing the first level of a 2-level bypass network used with a 2-cycle register file. IPC was degraded because dependent instructions could no longer execute in back-to-back cycles. They point out that if the second (rather than the first) level of the bypass network were removed, there would be a 'hole' in data availability—that is, results are available the first cycle after they are produced, then are not available in the following cycle, and then are later available from the register file. This paper examines limited multi-level bypass networks that cause holes in data availability. Section 4.3 will explain how to schedule instructions around these holes.

3. Redundant Binary Arithmetic

This section presents an overview of the redundant binary number system that we utilize to make fast adders. Section 3.1 describes the redundant binary representation. Section 3.2 describes how to convert between redundant binary and 2's complement representation. Section 3.3 describes how addition works in the redundant binary number system and implementation of a redundant binary adder. Section 3.4 discusses the delays of redundant binary adders relative to carry-lookahead adders. Section 3.5 describes overflow in the redundant binary system and detection of 2's complement overflow. These sections are primarily a review of information presented in previous work [2, 3, 9, 11, 16]. Section 3.6 describes how other operations used in a modern instruction set, specifically the Alpha ISA, may be handled in a redundant binary system.

3.1. Overview

Most current ISAs use 2's complement representation for integers. Two's complement representation has several attractive properties: (1) addition and subtraction operations work the same way regardless of whether the numbers are positive or negative, (2) there are approximately the same number of positive and negative values represented with a given number of bits, and (3) N bits can be used to represent 2^N possible values—that is, each distinct pattern of 0s and 1s represents only one value, and each distinct value has only one representation.

Redundant number systems can have more than one representation for a given value, so they require more bits for representing the same range of integers as 2's complement representation. However, many redundant number systems do have an advantage over 2's complement, which is that addition can be performed in constant time regardless of operand size. There are many redundant number representations; we will limit our discussion to one such representation called *signed digit* representation.

Signed-Digit number representations were first described by Avizienis [3]. Each digit of a number in signed-digit representation may be either positive or negative. In this paper, we will only discuss a specific signed-digit representation commonly called *redundant-binary* representation where each digit can take on any value from the set {-1, 0, 1}. Because there are three possible values of a digit, two bits are required to encode each digit.

In conventional unsigned binary format, the i^{th} digit represents 2^i multiplied by 0 or 1. In redundant binary representation, the i^{th} digit represents 2^i multiplied by -1, 0, or 1. An n-digit redundant binary number X = $x_{n-1}, x_{n-2}, ..., x_0$, where $x_i \in$ {-1, 0, 1} represents the value $\sum_{i=0}^{n-1} x_i 2^i$. For example, the 4-digit number $\langle 0, 1, 0, -1 \rangle$ represents $2^2 - 2^0 = 3$. Three could also be represented by $\langle 0, 0, 1, 1 \rangle$ in redundant binary.

3.2. Conversion To/From 2's Complement

A redundant binary number is converted back to 2's complement by means of a subtraction with carry propagation. Suppose the redundant binary number X is represented by two sets of bits, X^+ and X^-, representing the positive and negative powers of 2, respectively, that are added to compute the value. For example, if $X = \langle 0, 1, 0, -1 \rangle$, then $X^+ = \langle 0, 1, 0, 0 \rangle$ and $X^- = \langle 0, 0, 0, 1 \rangle$. The 2's complement representation of X can be computed by subtracting X^- from X^+ in the 2's complement number system.

The conversion from 2's complement to redundant binary is straightforward. All bits except the most significant bit of the 2's complement number are assigned to the positive component, X^+. The most significant bit of the 2's complement number is assigned to the most significant digit of X^- so that the number will retain the correct sign.

If the values 1, 0, and -1 are encoded as $\langle 0, 1 \rangle$, $\langle 0, 0 \rangle$, and $\langle 1, 0 \rangle$, respectively (i.e. one bit indicates the digit is negative, the other indicates it is positive), then converting from 2's complement to redundant binary requires no logic; the path can be hardwired. The conversion from redundant binary back to 2's complement requires a 2's complement subtraction circuit with full carry propagation.

3.3. Implementation

Redundant binary addition limits carry propagation to at most two digits. The computation of the ith digit of the sum is a function of digits i, $i-1$, and $i-2$ of both inputs.

Several possible logic diagrams for one digit-slice of a redundant binary adder were shown in previous works [12, 16]. One logic diagram is shown in Figure 2. The intermediate output h_i is a function of digit i. The output f_i is a function of digit i and h_{i-1}. The sum for digit i is a function of digit i, h_{i-1}, and f_{i-1}.

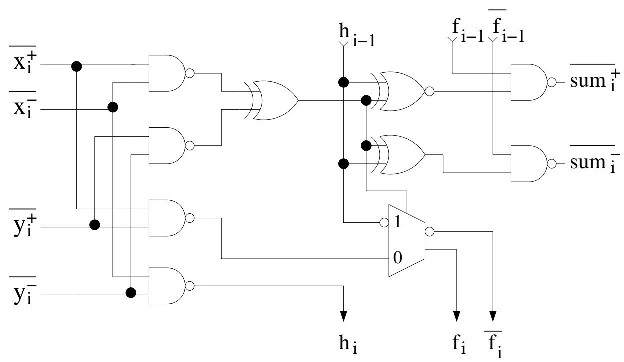

Figure 2. One Digit-Slice

3.4. Delays of Redundant Binary Adders

The critical path through one bit slice of a redundant binary adder, which is also the critical path through the whole adder, consists of seven transistors with fan-outs less than or equal to 4.

Several researchers have compared the delays of redundant binary adders to 2's complement carry-lookahead (CLA) and carry-select adders. Makino et. al. [12] fabricated a multiplier that used redundant binary adders. They simulated several redundant binary adders and CLAs using SPICE2 with a 0.5-μm CMOS process technology. They found that a redundant binary adder was 2.7 times faster than a redundant binary to 2's complement converter, and 3 times faster than a conventional 64-bit CLA.

Nagendra et. al. [15] compared the delays of 32-bit CLA and signed-digit adders optimized for performance. They found that their signed-digit adder, which used a radix-4 signed-digit representation, was 2.6 times as fast as the CLA. The same authors found a carry-save adder, which uses a redundant representation similar to the redundant binary representation described in this paper, to be twice as fast as their signed-digit adder [14].

3.5. Overflow

Integer overflow occurs when the result of a computation is too large to fit within a fixed number of bits. With 2's complement, this is easily detected by examining the signs of the sources and destination.

With a chain of redundant binary additions, carry-outs can quickly propagate towards the most significant digit of a number. For example, when the value 1 is repeatedly incremented in redundant binary using an adder built from the circuit in Figure 2, the representations of the resulting values will be $\langle 0,0,0,1\rangle$, $\langle 0,0,1,0\rangle$, $\langle 0,1,0,-1\rangle$, $\langle 1,-1,0,0\rangle$, $\langle 1,-1,1,-1\rangle$, and so on. Non-zero digits propagate left faster in redundant binary than in 2's complement. It is possible for there to be a carry-out of the most significant digit, while the value of the number may still be representable with fewer digits. When this scenario, called *bogus overflow*, occurs, either the carry-out is 1 and the most significant digit is -1, or vice-versa. This can be easily avoided by using one of several techniques [2]. One technique exploits the fact that the representations $\langle 1,-1\rangle$ and $\langle -1,1\rangle$ can be converted to $\langle 0,1\rangle$ and $\langle 0,-1\rangle$, respectively. When bogus overflow occurs, the sign of the most significant digit is complemented.

In addition to correcting for bogus overflow, 2's complement overflow must still be detected. Two's complement overflow will occur if any of the following events occur:

- The carry-out is still -1 or 1 *after* correcting for bogus overflow.
- The most significant digit is -1 and the rest of the result is negative. In order for the result to have the same value as if it were computed in 2's complement, the most significant digit should be set to 1.
- The most significant digit is 1 and the rest of the result is not negative. In order for the result to have the same value as if it were computed in 2's complement, the most significant digit should be set to -1.

3.6. Other Operations

Not all operations in modern instruction sets can be computed in the redundant binary system. This section discusses which integer instructions in the Alpha ISA can execute in redundant binary format.

Arithmetic Operations. Addition, subtraction, and multiplication using redundant number systems has been demonstrated in previous work [2]. The Alpha ISA also has three additional arithmetic instructions, CTLZ (count the leading zeros of an operand), CTTZ (count the trailing zeros of an operand), and CTPOP (count the number of set bits in an operand). CTLZ and CTPOP require the operand to be in a unique representation, which means they should be executed in 2's complement format. CTTZ may be executed in redundant binary by counting the trailing non-zero digits.

Byte Manipulation and Logical Operations. The byte manipulation instructions extract data from one or more bytes of their operands. If individual bytes are extracted from a redundant binary number, the result, when converted back to 2's complement, may be incorrect. Hence byte operations are performed in 2's complement. Bitwise logical operations (e.g. XOR) also require the inputs to be in 2's complement to produce the correct result. One exception is when the two source register operands of a logical operation are the same register. This is the standard way to implement the MOVE operation in the Alpha ISA.

Conditional Operations. Conditional moves and branches test for values greater than, equal to, or less than zero; or they check to see if the least significant bit of a value is set. Both number systems require an OR circuit to test for zero. Testing for positive or negative values in 2's complement is straightforward: the value is negative if the most significant bit is set. In redundant binary, the sign of a number is determined by the most significant digit with a value other than zero. If this digit is -1, the number is negative; otherwise it is positive. Testing the least significant bit of a number in redundant binary format requires a 2-input OR of the two bits comprising the least significant digit of the number. In summary, all conditional move and branch instructions may be executed in redundant binary format, although an additional circuit is needed to test for positive or negative values.

Shifts and Scaled Adds. The Scaled Add instruction of the Alpha ISA shifts one of the inputs to the left by two or three bits before adding it to an immediate value. Scaled Adds and Left shifts will work in the redundant binary system by shifting digits rather than bits. If the most significant bit of the result is 1, it should be changed to -1 because the number would be negative in 2's complement representation. For example, the number $\langle -1, 1, 0, 1\rangle$ (-3 in decimal representation) would become $\langle -1, 0, 1, 0\rangle$ (-6) when shifted left by one digit. A right shift may not produce the correct result in redundant binary. Right shifts are performed with 2's complement inputs.

Memory Access Instructions. Loads and Stores can compute memory addresses in redundant binary format. The cache index is formed from a subset of the bits of the 2's complement representation of the memory address, and is not easily formed from the redundant binary representation. Sum Addressed Memory [9] (SAM) can be used to avoid converting the address to 2's complement.

In conventional data caches, the cache index is an input to a decoder. The output of this decoder is a one-hot vector that asserts one word line for a row of the cache. SAM uses a different type of decoder from conventional caches. A SAM decoder accepts two numbers, a base and a displacement, as input, and produces a one-hot vector of word lines. The one-hot vector is produced using a separate equality test for each word line rather than a full carry-propagating addition. The Sun UltraSPARC III uses Sum Addressed Memory to avoid the base + displacement calculation normally needed for address computation [11]. SAM can be used to index the data cache with a single redundant binary number by treating the positive and negative components of the number as the two SAM inputs.

It is also possible to use a modification of SAM to eliminate the base + displacement calculation when the base register is in redundant binary format. This modified SAM has three inputs: the positive and negative components of a redundant binary number and a 2's complement displacement. The modified SAM consists of a conventional SAM preceded by a circuit similar to a carry-save adder. The critical path through the modifed SAM is, at worst, the critical path through the conventional SAM preceded by a 3-input XOR gate. In our experiments, we assume that all machines utilize SAM to avoid the base plus displacement calculation.

Data loaded from memory is already in 2's complement format because data is stored in the caches and main memory in 2's complement representation.

Quadword to Longword Forwarding. Alpha supports both quadword (64-bit) and longword (32-bit) data types. A quadword instruction may forward its result to a longword instruction. The lower 32 bits of the quadword are extracted to form the longword input. In order to extract the lower half of a quadword in redundant binary format, the same mechanism used for correcting bogus overflow and conditions for testing 2's complement overflow must be used at the 32nd digit in addition to the 64th digit. When 2's complement numbers are converted to redundant binary, the 32nd bit should be hardwired to the negative portion of the 32nd digit of the redundant binary representation so that longwords will retain the correct sign.

Summary of Instruction Classifications. Table 1 classifies the fixed-point instructions of the Alpha ISA according to their input and output formats. If the format is listed as RB, operands can be in either redundant binary or 2's complement format. If the format is TC, operands must be in 2's complement format. The fourth column of the table indicates the fraction of dynamic instructions belonging to each class. On average, 33% of the instructions with register destinations produce results in redundant binary format, and about 25% of the instructions require at least one input in 2's complement format.

4. The Execution Core

This section describes how redundant binary adders and limited bypass networks can be incorporated in an execution core. Section 4.1 discusses two possible execution core configurations. Section 4.2 discusses limited bypass networks. Section 4.3 discusses solutions for two scheduling problems: scheduling in a machine with redundant binary adders (and hence multiple data formats) and scheduling in

Instruction	Input Formats(s)	Output Format	Fraction of Inst. Stream
ADD, SUB, MUL, LDA, LDAH, CMOVLBx, SxADD, SxSUB, SLL	RB	RB	18.0%
CMOVLT, CMOVGE, CMOVLE, CMOVGT †	RB	RB	0.4%
CMOVEQ, CMOVNE ‡	RB	RB	0.5%
Memory Access	RB	TC	36.6%
CMPEQ ‡	RB	TC	0.5%
CMPLT, CMPLE, CMPULT, CMPULE †	RB	TC	3.9%
conditional branches † ‡	RB	—	14.4%
Other	TC	TC	25.7%

†test for positive/negative values requires extra logic tree or wired-OR
‡test requires subtraction for comparison

Table 1. Instruction Classifications

a machine with limited bypass networks.

4.1. Core Configurations

This section discusses two possible execution core configurations. The first uses a physical register file that stores data in 2's complement representation. The second uses two or more copies of the physical register file: some using 2's complement representation and some using redundant binary representation. Although each entry in a redundant binary register file requires twice as many bits of state as an entry in a 2's complement register file, fewer bypass paths are needed when redundant binary register files are used.

Only TC Register Files. If register files only store data in 2's complement, the output of the ALUs producing redundant binary results (i.e. the *RB-output* ALUs) must be converted back to 2's complement before it can be stored. Figure 3 shows an example of an RB-output ALU, an ALU that accepts only 2's complement inputs (i.e. a *TC-input* ALU), and part of the bypass network.[1] Three bypass levels are required for any RB-output ALU. The paths in bold hold data in redundant binary format. The first two bypass levels (BYP-1 and BYP-2) of the RB-output ALU can be used by any RB-input functional unit, but they cannot be used by TC-input functional units.

Consider the dependency graph and pipeline diagram in Figures 4 and 5. The ADD gets the result of the SLL (Shift Left Logical) from the first bypass path (BYP-1). The AND gets the result of the SLL in 2's complement format from BYP-3. The SUB gets the ADD's result from BYP-1 and

[1]This example and all further examples in this section assume the redundant binary addition and logical operations take one cycle, the format conversion takes two cycles, and the register file access takes one cycle. Bypass paths needed to bypass values from the register file write stage to the register file read stage are not shown in the figures.

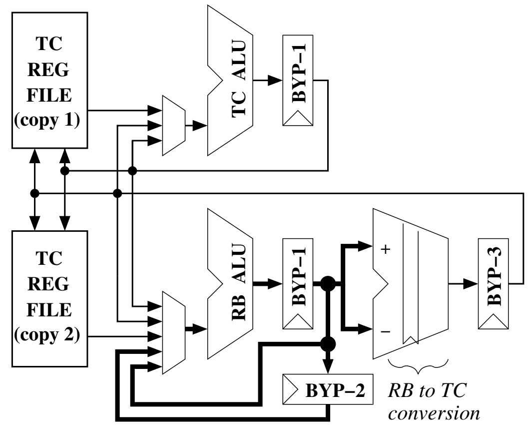

Figure 3. ALUs with TC Register Files

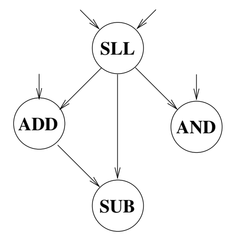

Figure 4. Dependency Graph

	Cycle:	1	2	3	4	5	6	7
(RB ALU)	SLL	RF	EXE	CV1	CV2	WB		
(TC ALU)	AND				RF	EXE	WB	
(RB ALU)	ADD		RF	EXE	CV1	CV2	WB	
(RB ALU)	SUB			RF	EXE	CV1	CV2	WB

Figure 5. Pipeline Diagram.

the SLL's result from BYP-2.

TC and RB Register Files. In the second configuration, the inputs to TC-input functional units come from 2's complement register files and 2's complement bypass paths. The inputs to other functional units come from redundant binary register files and bypass paths in either representation. Figure 6 shows an example with one TC-input ALU and one RB-output (and TC or RB-input) ALU. This configuration requires the same number of bypass paths as a machine with only TC ALUs. There is no second-level bypass, and BYP-3 is only used by the TC-input ALU and TC register file. The timing of operations is the same as when using all TC register files.

4.2. Limited Bypass Networks

The number of bypass levels, and hence the number of bypass paths, required in a full bypass network increases linearly with the number of cycles between execution and

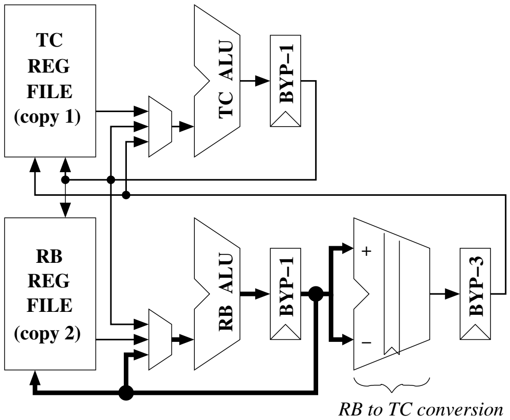

Figure 6. ALUs with TC and RB Register Files

the last stage of register file write-back. This section explains how bypass levels can be removed.

Most instructions execute as soon as their last (or only) source operand is available. When an instruction executes immediately when its last source operand becomes available, the input comes from a first-level bypass path (i.e. BYP-1). Hence the first-level bypass paths are used more often than any others. An instruction would only need a second-level bypass in the following cases:

- it had a second source that became available the cycle before the last available source
- it was stalled for one cycle because another instruction was granted execution
- it was just recently placed in the scheduling window and was scheduled at its earliest opportunity.

None of these situations occur very frequently.

The machine with only TC register files was modeled with a limited bypass network: the second bypass level (BYP-2) was removed, and the output of BYP-3 was not used as a bypass for the RB-input ALUs. For RB-input instructions, the result of an RB-output instruction is available in redundant binary format immediately after it is produced, and then there is a 2-cycle hole in data availability. After that, the result is available from the register file. For TC-input instructions, the result is available from BYP-3, and then from the register file.

For the dependency graph shown in Figure 4, this limited bypass network would result in the pipeline schedule shown in Figure 7. The AND gets the result of the SLL from BYP-3. The SUB is delayed by three cycles and retrieves both source operands from the register file (or a bypass path within the register file, depending on the register file design). Performance results of this machine and a conventional 2's complement machine with a limited bypass network are discussed in Section 5.2.

	Cycle:	1	2	3	4	5	6	7	8	9	10
(RB ALU)	SLL	RF	EXE	CV1	CV2	WB					
(TC ALU)	AND				RF	EXE	WB				
(RB ALU)	ADD		RF	EXE	CV1	CV2	WB				
(RB ALU)	SUB						RF	EXE	CV1	CV2	WB

Figure 7. Pipeline Diagram.

This paper only investigates the removal of entire levels of bypass paths. Further restrictions in bypass networks may be made with little loss in IPC with the help of instruction steering. This topic remains an area of future work.

4.3. Scheduling

The use of multiple data formats can present scheduling problems due to variable times for result availability. For example, a SUB dependent on an ADD may be scheduled one cycle after the ADD, but a logical instruction dependent on an ADD must wait 3 cycles (1 for the addition, 2 for the format conversion) before it can be scheduled.

There are many scheduling techniques to handle this problem. One technique is to use separate schedulers for the different classes of operations identified in Table 1. The result tag broadcasts, or wakeup signals, from a scheduler for RB-output instructions to a scheduler for TC-input instructions can be latched for 2 cycles to account for the format conversion. The use of separate schedulers is warranted since these two classes of instructions execute on different functional units. Another technique is to associate the result of a redundant binary operation with two resources (i.e. physical register numbers or tags): one resource indicates that the result is available in redundant binary format; the other indicates the result is available in 2's complement.

The limited bypass network described in Section 4.2 will create holes in data availability. Wakeup array-style scheduling logic [4] can be used to schedule around these holes. Figure 8 (a) shows a block diagram of this scheduling logic. The wakeup logic contains information about each instruction that specifies which resources (i.e. source operands or functional units) are needed. One input to the wakeup logic is a wire for each resource, called the RESOURCE AVAILABLE line, which is asserted if that resource is available. The wakeup logic for an instruction monitors the RESOURCE AVAILABLE lines for the required resources. When all required RESOURCE AVAILABLE lines are asserted, the instruction requests execution. When it is granted execution by the select logic, a shift register that acts as a countdown timer to count the instruction's latency is enabled. The output of the shift register is the RESOURCE AVAILABLE line for that instruction, and it indicates when dependent instructions may be scheduled. For example, the timer in Figure 8 (b) would be used for a 2-cycle instruction (assuming the scheduling operation takes 1 cycle). To handle holes in data availability, the initial value in the shift register would interleave 0s and 1s according to which levels of the bypass network were missing.

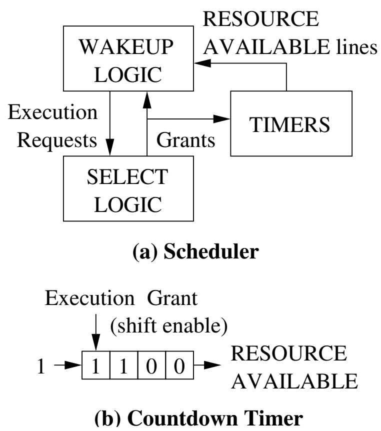

Figure 8. Scheduling logic

5. Experiments

5.1. Machine Model

The SPECint95 and SPECint2000 benchmarks were simulated using an execution-driven simulator for the Alpha ISA. All benchmarks were run to completion using modified input sets to reduce simulation time. Some characteristics of the machine model are shown in Table 2. The pipeline latency was a minimum of 13 cycles: 6 for fetch and decode, 2 for rename, 1 for schedule, 2 for register file read, a minimum of 1 for execution, and 1 for retirement.

Branch Predictor	48KB hybrid gshare/PaS, 4096-entry BTB 2 basic blocks per cycle fetched
Decode, Rename, and Issue Width	8 instructions
Instruction Cache	64KB 4-way set associative (pipelined) 2-cycle directory and data store access
Instruction Window	128 Reservation Station Entries
Execution Width	4 or 8 functional units
Data Cache	8KB 2-way set associative (pipelined)
Unified L2 Cache	1MB, 8-way, 8-cycle access contention for 2 banks is modeled
Memory	100-cycle access contention for 32 banks modeled

Table 2. Machine Configuration

Machines with 4 and 8 functional units were studied because the effects of execution latency depend on the execution bandwidth. As execution bandwidth increases, performance is more dependent on the latencies of instructions on the critical path. All other machine parameters remained constant so that the amount of exposed ILP changed as little as possible. All functional units were homogeneous. The use of special-purpose functional units would have made it difficult to make a fair comparison between redundant binary and conventional execution cores because the classes of operations that would execute on each type of functional unit depend on the data representation.

All machines had a 128-entry instruction window and select-2 schedulers (i.e. schedulers that pick 2 instructions per cycle for execution on 2 functional units). The 4-wide machine had two schedulers, each holding 64 instructions. The 8-wide machine had 4 schedulers, each with 32 instructions. Groups of two consecutive instructions were steered to each scheduler in a round robin manner. The 8-wide machine was partitioned into two clusters, each with 4 functional units. If a result produced on a functional unit in one cluster had to be forwarded to a functional unit in the other cluster, there was a 1-cycle propagation delay.

For both execution widths, four machines were modeled: a **Baseline** machine, the RB (redundant binary) machine with TC register files and a limited bypass network (**RB-limited**), the RB machine with both TC and RB register files (**RB-full**), and the **Ideal** machine. The **Baseline** machine used 2-cycle, pipelined 2's complement ALUs. The **RB** machines used 1-cycle redundant binary adders with 2-cycle format converters. The **Ideal** machine used 1-cycle 2's complement arithmetic units. The execution latencies for the machines are given in Table 3. The **RB-limited** machine used the bypass network described in Section 4.2. All machines had the same number of bypass paths.

Instruction Class	Base	RB (TC result)	Ideal
integer arithmetic	2	1 (3)	1
integer logical	1	1	1
integer shift left	3	3 (5)	3
integer shift right	3	3	3
integer compare	2	1 (3)	1
byte manipulation	2	1 (3)	1
integer multiply	10	10	10
fp arithmetic	8	8	8
fp divide	32	32	32
loads, stores (SAM decoder)	1	1 (3 for stores)	1
dcache latency	2	2	2

Table 3. Instruction Class Latencies

5.2. Results

Figure 9 shows the IPC of the 8-wide machines on the SPECint2000 benchmarks. For each benchmark, the first bar shows the IPC of the **Baseline** machine; the next two bars represent the **RB** machines, and the last bar represents the **Ideal** machine. The **RB-full** machine had an IPC 7% higher than the **Baseline** machine, and within 1.1% of the **Ideal** machine. Figure 10 shows the results on the SPECint95 benchmarks. The **RB** machine had an IPC 9% higher than the Baseline machine, and within 2% of the Ideal machine. Overall, the **RB-limited** machine performed within 2% of the **RB-full** machine.

The results for 4-wide machines are shown in Figures 11 and 12. Fast functional units have less of an advantage on the 4-wide machines because the execution bandwidth is a bottleneck for the amount of exposed ILP in these benchmarks. On the SPECint2000 benchmarks, the **RB-full** machine has an IPC 5% above the **Baseline** machine, and within 0.5% of the **Ideal** machine. On the SPECint95 benchmarks, the **RB-full** machine has an IPC 6% above the **Baseline** machine, and within 1.3% of the **Ideal** machine. Overall, the **RB-limited** machine performed within 2.3% of the **RB-full** machine.

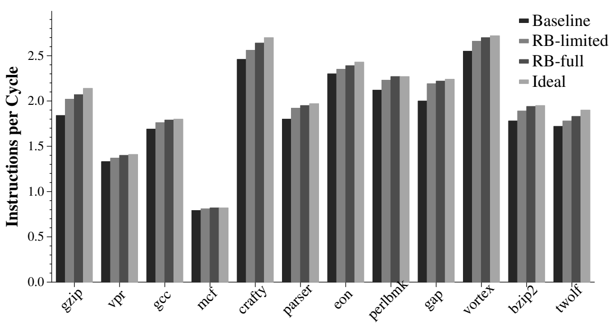

Figure 9. 8-wide machines, SPECint2000.

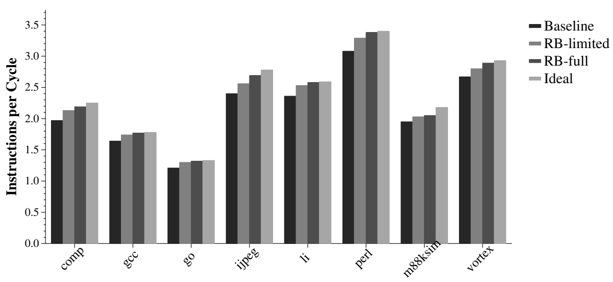

Figure 10. 8-wide machines, SPECint95.

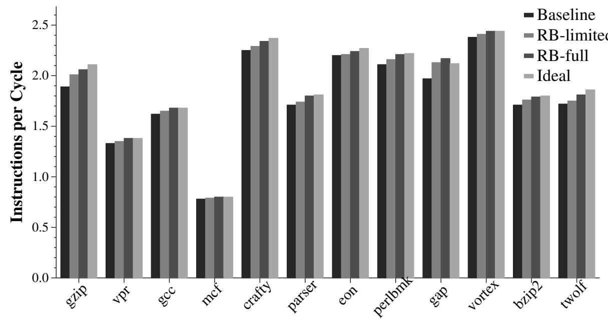

Figure 11. 4-wide machines, SPECint2000.

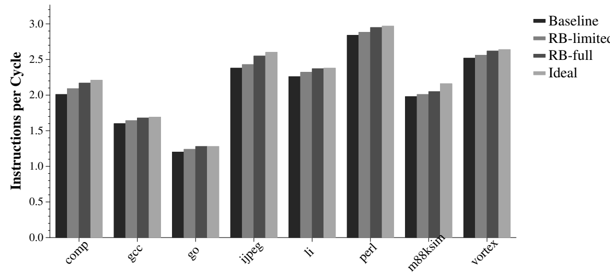

Figure 12. 4-wide machines, SPECint95.

Format Conversions. The IPC of the **RB** machines is lower than that of the **Ideal** machine because of the format conversions that were on the critical path of execution. There are four cases of data bypasses: (1) a 2's complement result is forwarded to a 2's complement operation, (2) a 2's complement result is forwarded to a redundant binary operation, (3) a redundant binary result is forwarded to a redundant binary operation, and (4) a redundant binary result is forwarded to a 2's complement operation. Only the fourth case requires a format conversion. Figure 13 shows a distribution of the four cases for the 8-wide **RB-full** machine on the SPECint2000 benchmarks. Only last-arriving bypassed source operands (i.e. source operands that delay an instruction's execution) are included in the distribution. The number at the top of each bar indicates the fraction of all dynamic instructions that had at least one bypassed source operand. The numbers at the bottom indicate the fraction of the data bypasses that required format conversion (RB to TC). For example, on the bzip2 benchmark, 2.4% of 69% of all dynamic instructions were delayed because of a format conversion. Because a majority of the last-arriving sources are from memory loads, which produce 2's complement results, few last-arriving sources required format conversions.

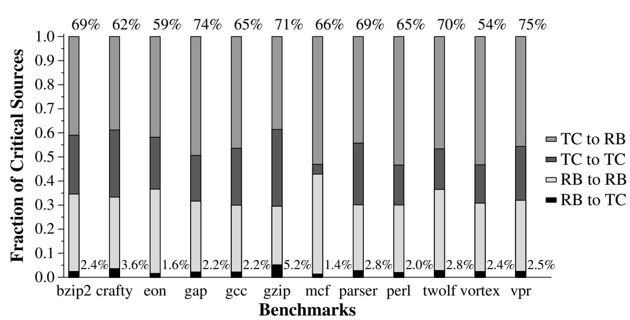

Figure 13. Potentially critical bypass cases.

Limited Bypass Networks. To evaluate the potential of using a scheduler that can support holes in data availability, the **Ideal** machine was modeled with limited bypass networks. Because it had a 2-cycle register file, three levels of bypass paths were required for a full bypass network. Five limited bypass configurations were modeled: **No-1** had no first-level bypass paths, **No-2** had no second-level bypass paths, **No-1,2** had no first or second-level bypass paths, and so on. The difference between the **Ideal** machine and the **No-1** machine is the effect of increasing all execution latencies by one cycle. In the **Ideal** machine, 21% to 38% of the instructions did not receive any sources off of the bypass network, 51% to 70% retrieved a source operand from the first-level bypass bus, and 5% to 14% of the instructions received a source operand from another bypass path. Because the first-level bypass paths are heavily utilized, those machines that do not remove the first-level bypass paths perform the best. The harmonic means of the IPC over all 20 benchmarks for each machine are shown in Figure 14. The 4-wide **No-1,2** machine outperformed the 8-wide **No-1,2** machine because the 8-wide machines are clustered, and both machines have ample execution bandwidth for the long execution latencies.

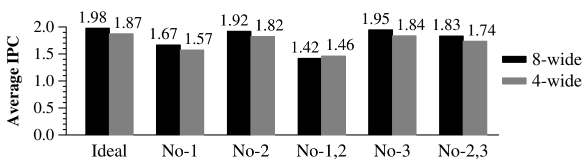

Figure 14. IPC with Limited Bypass Networks.

6. Conclusions

Redundant binary adders have about half the latency of 2's complement adders. As a result, an execution core built from redundant binary adders can be clocked at a higher frequency, resulting in greater execution bandwidth and lower execution latencies. Our results show that redundant binary adders provide a significant increase in performance for latency-critical applications. Two's complement, pipelined adders are sufficient for throughput-intensive applications. To further reduce execution and forwarding latency, limited bypass networks may be used with little loss in IPC.

Acknowledgements

We would like to thank Jared Stark and the anonymous referees for their comments on earlier drafts of this work. We would also like to thank Andy Glew, Shih-Lien Lu, and Chris Wilkerson for their valuable discussions. This work was supported in part by Intel and IBM. Mary Brown is supported by an IBM Cooperative Graduate Fellowship.

References

[1] P. S. Ahuja, D. W. Clark, and A. Rogers. The performance impact of incomplete bypassing in processor pipelines. In *Proc. of MICRO-28*, pages 36–45, 1995.

[2] D. E. Atkins. Design of the arithmetic units of ILLIAC III: Use of redundancy and higher radix methods. *IEEE Trans. on Computers*, C-19:720–732, Aug. 1970.

[3] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, EC-10(9):389–400, Sept. 1961.

[4] M. D. Brown, J. Stark, and Y. N. Patt. Select-free scheduling logic. In *Proc. of MICRO-34*, 2001.

[5] J.-L. Cruz, A. González, M. Valero, and N. P. Topham. Multiple-banked register file architectures. In *Proc. of ISCA-27*, pages 316–324, 2000.

[6] M. D. Ercegovac. On-line arithmetic: An overview. *SPIE Real-Time Signal Processing VII*, 495:86–93, Aug. 1984.

[7] A. Glew. Processor with architecture for improved pipelining of arithmetic instructions by forwarding redundant intermediate data forms. U.S. Patent Number 5,619,664, 1997.

[8] J. Gray, A. Naylor, A. Abnous, and N. Bagherzadeh. VIPER: A VLIW integer microprocessor. *IEEE Journal of Solid-State Circuits*, 28(12):1377–1383, Dec. 1993.

[9] R. Heald, K. Shin, V. Reddy, I.-F. Kao, M. Khan, W. L. Lynch, G. Lauterbach, and J. Petolino. 64-KByte sum-addressed-memory cache with 1.6-ns cycle and 2.6-ns latency. *IEEE Journal of Solid-State Circuits*, 33(11):1682–1689, 1998.

[10] Intel Corporation. *IA-32 Intel Architecture Software Developer's Manual With Preliminary Willamette Architecture Information Volume 1: Basic Architecture*, 2000.

[11] W. L. Lynch, G. Lauterbach, and J. I. Chamdani. Low load latency through sum-addressed memory (SAM). In *Proc. of ISCA-25*, pages 369 – 379, 1998.

[12] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko. An 8.8-ns 54 x 54-bit multiplier with high speed redundant binary architecture. *IEEE Journal of Solid-State Circuits*, 31(4):773–783, Apr. 1996.

[13] R. Nagarajan, K. Sankaralingam, D. Burger, and S. W. Keckler. A design space evaluation of grid processor architectures. In *Proc. of MICRO-34*, 2001.

[14] C. Nagendra, M. J. Irwin, and R. M. Owens. Area-time-power tradeoffs in parallel adders. *IEEE Transactions on Circuits and Systems*, 43(10):689–702, Oct. 1996.

[15] C. Nagendra, R. M. Owens, and M. J. Irwin. Power-delay characteristics of CMOS adders. *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, 2(3):377–381, Sept. 1994.

[16] N. Takagi, H. Yasuura, and S. Yajima. High-speed vlsi multiplication algorithm with a redundant binary addition tree. *IEEE Trans. on Computers*, C-34(9):789–796, Sept. 1985.

[17] D. M. Tullsen, S. J. Eggers, J. S. Emer, and H. M. Levy. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proc. of ISCA-23*, pages 191–202, 1996.