A Reversible Computer Architecture

Carlin J. Vieri*
cvieri@ai.mit.edu
MIT Artificial Intelligence Laboratory

Introduction

Power in modern microprocessors is rapidly becoming a primary design concern. Microprocessors containing a few million transistors and dissipating tens of watts are becoming commonplace, making heat removal in dense structures difficult and limiting their usefulness in portable applications. Increasing device density and an expanding market for portable devices will continue to push traditional low power design techniques to their limits.

These techniques have a limit imposed by the second law of thermodynamics, which requires a minimum increase in a physical system's entropy when information is discarded. Early physicists were confused on this point, suggesting that a minimum energy dissipation of $kT \ln 2$ was associated with each boolean logic operation performed. Landauer [Lan86] forcefully pointed out that this result was incorrect and that this minimum dissipation limit applied only to each bit *erased*. A processor which erases very few bits during computation may therefore dissipate very little energy. To perform useful computation and not destroy information requires that the computation be *reversible*. Ressler [Res81] first considered a reversible architecture, and Hall [Hal94] discussed a reversible processor based on a PDP-10 instruction set. A number of reversible, energy recovering circuit techniques have been proposed, SCRL [YK94] being the most refined. This paper presents Pendulum, a reversible processor architecture.

Reversible Architecture Design

A reversible computer may not erase bits. All information necessary to reverse the computation must be retained. In designing a reversible processor, all operations must have some inverse. Any information not actually needed by the program must be kept in a "garbage stack" for retrieval later. Control flow information poses a particularly tricky problem because it differs greatly from standard irreversible processor operation. Jumps and branches must be undone when running in reverse, so extra program trace information must be retained.

A reversible processor needs to keep track of two types of program information: operands used in irreversible operations (stored on the program garbage stack), and program flow information resulting from jumps and branches (stored on the PC stack). The information in these stacks may be compressed, manipulated, or moved to a different location for disposal. Dissipating energy by destroying bits in a remote location is not useful for reducing energy consumption of a processor, but it is important if heat removal is a limiting factor in the packing density of computing elements.

Memory Access

Since information in a reversible machine may not be destroyed, any type of memory access which overwrites the previously stored value is unacceptable. Traditional load/store memory accesses assume that copying and overwriting information are free operations. If we combine them into a single operation, *exchange*, the information is merely moved from one place to another; nothing is destroyed. The programmer has explicit control over where intermediate results are placed in memory, and when the program execution is reversed the values are returned to the register file and uncomputed. Exchange, which is both symmetric and reversible, should be used to access all memory elements (registers and main memory) in a reversible processor.

Control Flow

Traditional control flow operations, such as jumps and branches, allow program execution paths to coalesce in a manner which has no closed form inverse. In the forward direction a branch conditional is evaluated, but in the reverse direction we may need to evaluate a different function of the data variables to correctly reverse our execution steps. In the simplest case, two or more unconditional branches to the same location results in ambiguity about which path to follow in reverse. A more complex issue, diagramed in Figure 1 arises if some number of conditional branch statements all have as their destination a particular instruction address. They may have identical conditions for branching, and there is no way to compute which branch was taken (if any) to bring the program into its current state. Unless information about which branch was taken is stored, the processor cannot execute the branch in reverse. To solve this problem, Pendulum treats control flow instructions as subroutine calls, and pushes the current program counter (PC) value onto the PC stack

*Carlin Vieri is advised by Tom Knight, Jr. and is supported in part by ARPA contract DABT63-92-C-0039. Address: Room 703, 545 Technology Square, Cambridge MA 02139. Web: http://www.ai.mit.edu/people/cvieri/cvieri.html

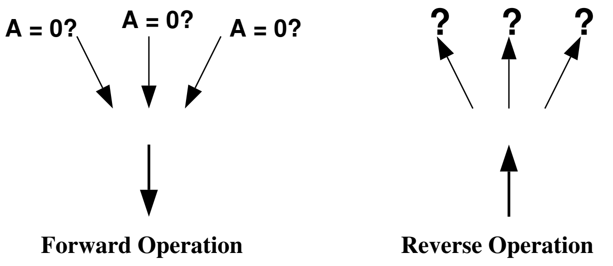

Figure 1: Demonstration of Control Flow Confluence

whenever the PC changes non-sequentially. Running in reverse, a single "come-from" instruction pops the PC value off the stack to undo execution. This PC value is decremented before it is loaded into the PC so that the branch is not executed in reverse.

Processor Configuration

We have designed and simulated in the Verilog HDL a processor based on the MIPS R2000 RISC processor. The simplicity of the MIPS design allows a greater emphasis to be put on the unique features of a reversible processor. The current design executes instructions in five cycles; each cycle resembles an appropriate pipeline stage even though the architecture is not pipelined. Pendulum follows the RISC philosophy of simple instructions and memory accesses, but takes the load/store architecture one step further and uses `exchange` as the fundamental memory access instruction.

In the interest of simplicity the datapath is configured under a "General Purpose Accumulator" scheme. This design maintains usefulness (at the price of a performance hit) if only accumulator style instructions are supported. The architecture requires two register addresses to be specified: a source/destination and a source, supporting operations of the form $A \leftarrow A \star B$ where $\star$ is some general operation. This design avoids the garbage creation that would occur with a GPR machine, since in the case of $A \leftarrow B \star C$, the value of A would have to be pushed onto the garbage stack before being overwritten.

Pendulum supports a full set of arithmetic and logical operations, shifts and rotates, both on register values and instruction immediates. Control flow is achieved through a number of unconditional and conditional jumps and branches, including linking jumps and jump to register values. Memory access is handled with the single exchange instruction.

Program Example

As a small example of how one of the standard idioms compiles on the Pendulum processor, consider an if-then-else structure. Control flow instructions target the instruction immediately following a come-from (cf) so that when running in reverse, the cf pops the PC value of the instruction immediately preceding the jump or branch. So the following pseudo-code

```
if (a > 0) a = a-10; else a = a-5;
```

compiles to:

```
top:  bgtz a,mid
      subi a,5
jad:  j end
      cf
mid:  subi a,10
      cf
end:  ...
```

Forward execution proceeds as follows: First, the comparison is made. If the branch is not taken, five is subtracted from `a` and control jumps (pushing the jump address `jad` onto the PC stack) to `end` and execution continues normally. If the branch is taken, `top` is pushed onto the PC stack, the PC is loaded with `mid`, ten is subtracted from `a`, `mid+1` is pushed onto the stack by the `cf` since control is falling into a section of code that is also a jump/branch destination, and execution continues from `end`.

Assuming the PC is beyond `end` and Pendulum is running in reverse, execution reverses as follows: The `cf` above `end` pops a PC value off the stack. This value will be either `mid+1` or `jad`. If it is `mid+1`, the PC is loaded with `mid`, ten will be added to `a`, the next `cf` will pop off another PC value, this time `top`, and the PC is loaded with `top-1`. Program reversal will continue above `top`.

If the first PC value popped off was `jad`, the PC is loaded with `jad-1`, five will be added to `a`, the `bgtz` is a NOP, and execution continues normally above `top`.

Conclusion

To reduce the power consumption of dense computation devices, bit erasure must be avoided. The Pendulum reversible computer architecture transfers entropy reversibly rather than increasing entropy by erasing bits, and may therefore dissipate very little energy.

References

[Hal94] J. Storrs Hall. A reversible instruction set architecture and algorithms. In *Physics and Computation*, pages 128–134, November 1994.

[Lan86] R. Landauer. Computation: A fundamental physical view. *Found. Physics*, 16:260–266, 1986.

[Res81] A. L. Ressler. The design of a conservative logic computer and a graphical editor simulator. Master's thesis, MIT Artificial Intelligence Laboratory, 1981.

[YK94] S. G. Younis and T. F. Knight, Jr. Asymptotically zero energy split-level charge recovery logic. In *International Workshop on Low Power Design*, pages 177–182, 1994.