Energy Efficient Program Control
6.893 Project Proposal

Ronny Krashinsky and Mike Sung
MIT Laboratory for Computer Science, Cambridge, MA 02139
{ronny|darkman}@mit.edu

9-26-2000

At an abstract level, programs consist of computation, data access, and control. The program control directs the flow of instructions during the execution of a program. Control flow instructions typically consist of conditional branches, jumps, and procedure calls and returns. Program control adds overhead to both the performance and energy usage of a program. In this project, we propose to investigate energy efficient program control mechanisms.

Initially, we will investigate program control in RISC processors. In a standard RISC processor, each instruction causes a flurry of micro-architectural operations. RISC ISAs have traditionally been optimized for performance; thus, they are designed to maximize the work performed by each instruction, as long as this work can be hidden in a pipelined implementation. This design goal is contrary to energy efficient implementations since in this case the work performed should be minimized.

Consider the following C code:

```
for(i = 0; i < x; i++)
  j += i <<1;
```

This is converted by the compiler to MIPS RISC code like:

```
L:  addu  $ri $ri 1
    addu  $rj $rj $rs
    slt   $rc $ri $rx
    bne   $rc $r0 L
    sll   $rs $ri 1
```

For the above code, we see that program control for the loop consists of a `slt` instruction to set the condition, followed by a `bne` instruction to branch to a program location based on that condition. The `slt` requires two 32-bit register file reads, a 32-bit ALU comparison, and a 32-bit register file write to set the comparison result. The `bne` requires two 32-bit register reads, another 32-bit ALU comparison, and a 32-bit branch target calculation.

As we can see, the program control for this sequence is extremely inefficient. 32-bit registers are used to store the true/false condition flag, and two 32-bit comparisons are performed. Many RISC architectures avoid this energy overhead by using condition codes to determine branches. Condition codes usually are set as a side-affect of arithmetic operations. Another alternative is to define a set of 1-bit condition registers which can be used as destinations for comparison instructions. In this case, boolean operations could also be performed using these registers. Another advantage of condition codes/registers is that they allow for very early branch resolution, potentially improving performance.

Another inefficiency in the above code is that the branch target address is calculated during every iteration of the loop. This could be avoided by including a branch target register (BTR) in the architectural state of the microprocessor. The compiler would set the BTR to the loop target before entering the loop (via a special load address instruction). Then the affect of the branch would be to set the program counter to the value of the BTR; there would be no need to recalculate the same branch target during every iteration of the loop. Additionally, the branch could be encoded using fewer bits, potentially leading to more energy savings.

With our proposed changes, the above code can be changed to:

```
    la    $btr L
L:  addu  $ri $ri 1
    addu  $rj $rj $rs
    sltb  $c1 $ri $rx
    bbz   $c1 (btr)
    sll   $rs $ri 1
```

This code is potentially much more energy efficient in its program control. Now the modified `slt` instruction (`sltb`) only writes a single bit for the comparison set. The modified branch instruction (`bbz`) only reads a single bit to determine the branch condition, and the branch target does not have to be calculated inside the loop.

In this project, we propose to evaluate the energy efficiency of alternative control mechanisms for RISC architectures. We plan to look into the mechanisms mentioned above, and we may also look into others such as auto-decrement instructions for loop control and predication to eliminate branches. If time permits we would also like to evaluate the energy efficiency of program control in other architectures such as vector, data flow, and VLIW.

Our plan is to:

- Profile benchmarks to determine commonly used control patterns (ISA simulator)
- Develop ISA extensions for alternative control mechanisms
- Modify compiler or assembly code to support ISA extensions
- Implement micro-architectural changes (SyCHOSys)
- Simulate energy usage (SyCHOSys)
- Analyze and iterate