Project 1 Part A: Design Your Own ISA

CSE 141, Fall 2008, Richard Strong

Due: October 15, 2008

You are encouraged to work on this project in teams of 2.

- October 9: e-mail a list of your team members to the TA (rstrong@cs.ucsd.edu). A hard copy is not needed. Make the subject of the email “cse141project1” so that the TA can filter your emails.

Overview

Project 1a and 1b will focus on the design of an ISA. This project will not require the use of the Xilinx tools. Your ISA should be general enough to run all the provided benchmark programs, while being simple enough to allow you to implement a reasonably fast processor in the allotted time frame for CSE 141L.

Team Policy

You can work in a team of two, and it is highly recommended. However, if your team mate drops this class, you must finish the project alone. So pick your partner wisely. Alternatively, you may work alone. Moderately higher standards are expected for teams of two over teams of one.

Goals

Ultimately your ISA and resulting implementation will be evaluated according to the following criteria:

- General purpose
  Is it able to execute the provided benchmark programs?
- Novelty
  How much is it different from existing ISAs such as MIPS or SPARC?
- Performance
  How long does it take to run benchmark programs on your processor?
• **Simple Hardware Design**
  How hard will it be to design this ISA in actual hardware? You will ultimately need to implement the ISA for your cse141L processor, so avoiding unnecessary evil is in your best interest.

• **Maximize Address Space**
  Make sure your ISA design can support an instruction and data address space of the full 34-bit range

**Requirements**

Your ISA should fulfill the following requirements.

• Instruction width: 17 bit
• Data and address widths(word size): 34 bit
• General purpose enough to be able to run provided benchmarks
• Include the following two instructions
  - `in [dest] [channel[3:0]]` : read a 34-bit data from the specified channel and save at [dest]
  - `out [src] [channel[3:0]]` : write a 34-bit data from [src] to the specified channel
• You must leave opcode space for expansion in case you leave out an instruction you need!
• The most significant bit of the instruction is reserved as a prediction bit (see lab2 part a for more details).

**I/O (Input/Output) Instructions**

Your ISA should support two I/O instructions - `in` and `out`. These instructions allow a processor to communicate with the external world via 16 channels. The width of a channel is 34-bit. The `in` instruction reads a 34bit data from a specified channel while the `out` instruction writes a 34-bit data to a specified channel.

I/O instructions have blocking semantics. If there is no data available to read by the `in` instruction, the processor simply waits until there is available data. Similarly, upon an `out` instruction, the processor must wait until the write operation completes; it might need to wait for available buffer space. You can freely use channels for various purposes such as debugging and multicore interface.

**Halt Instruction**

The halt instruction should stop execution of instructions.
Benchmarks

Three benchmarks are provided to help guide your ISA development. In Project 1b, you will write assembly programs based on your ISA for all three benchmarks. Assume that all the addresses (pointers) and data in the benchmark programs are 34 bit.

Fibonacci Number

// Recursive Fibonacci
// get 'n'th fibonacci number
// You should not alter the algorithm

int fib(int n)
{
    if (n < 0)
        return 0x3DEADBEEF;
    else if (n <= 2)
        return 1;
    else if (n == 29)
        return 514229;
    else if (n == 30)
        return 832030;
    else if (n == 48)
        return 4807526976;
    else if (n == 49)
        return 7778742049;
    else return fib(n-1) + fib(n-2);
}

Odd Job

Write a program that counts how many datawords (34bit) in an array have an even number of ones, and how many have an odd number of ones. For example, the binary number 11010001 has an even number of ones. The array is 64 words long, and starts at address 32. The number of entries that have an even number of ones should be written to address 24, and the number of odd to address 8.

Virtual Machine

// function virtualmachine:
//    emulates the execution of a very simple ISA
//    / 
//    // opcodes
//    // 0: subtract
//    // 1: right shift
//    // 2: nor
//    // 3: swap
struct inst {
  int op;
  int srcA;
  int srcB;
  int dest;
};

int VirtualMachine(int pc, int *mem) {
  while(1) {
    struct inst *instruction = mem[pc];
    int op = instruction->op;
    int srcA = instruction->srcA;
    int srcB = instruction->srcB;
    int dest = instruction->dest;
    pc = pc + 4;

    switch(op) {
      case 0: mem[dest] = mem[srcA] - mem[srcB]; break;
      case 1: mem[dest] = mem[srcA] >> 1; break;
      case 2: mem[dest] = ~(mem[srcA] | mem[srcB]); break;
      case 3: temp = mem[srcB]; mem[dest] = mem[mem[srcA]];
      case 4: in mem[dest], mem[srcA]; break; // in mem,
      case 5: out mem[srcA], mem[srcB]; break; // out data,
      case 6: mem[dest] = pc;
        if (mem[srcA] < 0) pc = mem[srcB];
        break;
      case 7: return pc;
    }
  }
}

Discussion Questions

1. In what ways did you optimize your dynamic instruction count?
2. How did you optimize your cycle time in the general case?
3. How would your design change if you had 7 fewer bits for each instruction (i.e. each instruction is 10 bits)? [Note: data memory remains 34 bits wide] How would your design change if you had 51 more bits for each instruction (68 bit instructions)? This would require that each instruction take up two data words? Alternatively, for a 68 bit instruction, would a larger data word (greater than 34 bits) be helpful? Why?
Project 1a Deliverables

Your team needs to submit a report, which should contain the following items:

- **Introduction:** The name of your ISA and detailed explanation (two pages or so) of your design decisions.
- **Instructions:** Instruction manual with instruction names, instruction formats, and RTL descriptions of functionality. For the manual style, follow the figure below, which is drawn from the MIPS R4400 manual. However, it does not need to be so ornately typeset.

### ADDIU

**Add Immediate Unsigned**

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDIU</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:**

ADDIU rt, rs, immediate

**Description:**

The 16-bit immediate is sign-extended and added to the contents of general register rs to form the result. The result is placed into general register rt. No integer overflow exception occurs under any circumstances. In 64-bit mode, the operand must be valid sign-extended, 32-bit values.

The only difference between this instruction and the ADDI instruction is that ADDIU never causes an overflow exception.

**Operation:**

\[
\begin{align*}
32 & : \quad \text{GPR}[rt] \leftarrow \text{GPR}[rs] + \text{immediate}_{15}^{16} || \text{immediate}_{15...0} \\
64 & : \quad \text{temp} \leftarrow \text{GPR}[rs] + \text{immediate}_{15}^{48} / / \text{immediate}_{15...0} \\
       & \quad \text{GPR}[rt] \leftarrow (\text{temp}_{31})^{32} || \text{temp}_{31...0}
\end{align*}
\]

- **Visible Architecture:** A description of all architectural (i.e. programmer visible) state (registers, memories, queues, global pointers, stack pointers, etc).
• Stack management and function call / parameter passing conventions
• Control Flow (branches): What types of branches are supported? How are target addresses calculated? What is the maximum branch distance?
• Addressing Modes: What memory and branch addressing modes are supported? How are target addresses calculated? What is the maximum branch distance?
• The three benchmark programs written in your ISA assembly language with heavy comments (one comment per-line is preferred).
• Discussion Questions

Points of consideration

• The grader will not be as familiar with your ISA as you. So be clear, concise and comment liberally (the three c’s for ISA success).
• Use the CSE141 WebBoard topic “Project 1” to ask questions or discuss the project. Students should feel encouraged to bounce ideas around.
• You are encouraged to pick the same partner you intend to work with in 141L.
• For lab 2 in CSE141L, the fetch unit uses the 5 least significant bits of the instruction as an offset. This consideration can be built into your ISA or you can modify the fetch unit later to match your ISA.

Useful Resources

• MIPS R4400 User's Manual