EE 4720 **Homework 2** Solution **Due: 9 October 2002**

ISA manuals are needed for some problems below. Links to the ISA manuals can be found on the new references Web page: `http://www.ece.lsu.edu/ee4720/reference.html`

Problem 1: Consider the following SPARC instructions:

```
sub %g3, %g2, %g1   ! g1 = g3 - g2;
and %g1, 0xf, %g1   ! g1 = g1 & 0xf
```

Wouldn't it be nice to have a `sub.and` instruction that would do both:

```
sub.and %g3, %g2, 0xf, %g1   ! g1 = ( g3 - g2 ) & 0xf
```

(*a*) Could the SPARC V9 ISA easily be extended to support such *double-op* instructions? If yes, explain how they would be coded.

Discussion: In the problem "easily be extended" means extending the ISA without adding entirely new instruction formats. The instructions above take two register source operands and an immediate source operand. Ordinary two-source register, one destination register instructions are coded using Format 3 with the `i` (immediate) bit set to zero. That format has an unused field, `asi`, which can be used for the immediate in the double-op instructions. The `asi` field is only eight bits, but that's enough for the immediates in the examples above.

Solution: Yes. Code the double-op instructions using SPARC Format 3 (`i=0`) and placing the immediate value in the `asi` field.

(*b*) Estimate how useful double-op instructions would be, using the data below. Usefulness here is conveniently defined as the dynamic instruction count. Consider a large class of double-op instructions that operate on two source registers and an immediate. For example, `add.add`, `sll.add`, and `and.or`. The data below does not provide important statistics needed to estimate the usefulness. Describe what statistics are needed and make up numbers. The made up numbers can be totally arbitrary (as long as they are possible).

The data below show instruction category and immediate sizes running the gcc compiler (cc1). Assume that this is a representative program and so the results apply to others. The data show the total number of instructions, and the breakdown by category, including ALU instructions that use an immediate, ALU instructions that use two source registers, etc. Following that histograms of the immediate sizes are shown for four instruction categories. This is very similar to the data shown in class. The percentage at each size and a cumulative percentage are shown. For example, 11.12% of ALU immediate instructions use two bits and 55.40% use two bits or fewer.

Quantifying usefulness means determining how many pairs of dynamic instructions can be combined into a double-op instruction. They can be combined if the first of the pair writes a register that is only used by the second of the pair (otherwise the first instruction could not be eliminated) and if the immediate will fit in the `asi` field (see the solution to the previous part).

The data below can be used to determine how many immediates would fit in the `asi` field, which is eight bits. The "ALU Immediate Size Distribution" table indicates that 95.13% of ALU instruction immediates are eight bits or less, which is good for the double-op instructions.

Using the data below one can only get a rough estimate of how many combinable pairs there are. One of each pair will be a two-source register ALU instruction, which represents 15.3% of all instructions. Assuming that each of these can be one of a pair (a bad assumption, but perhaps the best we can do with the data other than guessing) and assuming that 95.13% of the immediate instructions have small enough immediates yields 14.6% of the dynamic instructions. Assuming instruction count is proportional to execution time, the double-op instructions will reduce execution time from 1.0 to .854.

```
[drop] % echo /opt/local/lib/gcc-lib/sparc-sun-solaris2.6/2.95.2/cc1 \
  els.i -O3 -quiet  isize
Analyzed 156423240 instructions:
  48483403 (  31.0%) ALU Immediate
  23886739 (  15.3%) ALU Two Source Register
   6353567 (   4.1%) sethi
  34039161 (  21.8%) Loads and Stores
  30331049 (  19.4%) Branches
  13329321 (   8.5%) Other
```

ALU Immediate Size Distribution

Bits	Pct	Cum	
0	25.96%	25.96%	******************
1	18.32%	44.28%	*************
2	11.12%	55.40%	********
3	15.59%	70.99%	***********
4	3.16%	74.15%	***
5	6.10%	80.25%	*****
6	7.31%	87.56%	******
7	6.81%	94.37%	*****
8	0.76%	95.13%	*
9	2.13%	97.26%	**
10	2.64%	99.90%	**
11	0.01%	99.91%	*
12	0.08%	99.99%	*
13	0.01%	100.00%	*

SETHI Immediate Size Distribution

Bits	Pct	Cum	
0	0.00%	0.00%	*
1	0.00%	0.00%	*
2	0.02%	0.02%	*
3	0.72%	0.74%	*
4	0.43%	1.17%	*
5	0.09%	1.26%	*
6	0.66%	1.93%	*
7	2.00%	3.93%	**
8	4.53%	8.45%	****
9	3.14%	11.60%	***
10	5.86%	17.46%	*****
11	5.54%	23.00%	****
12	72.67%	95.67%	**
13	0.09%	95.76%	*
14	0.13%	95.89%	*
15	0.10%	95.99%	*
16	0.01%	96.00%	*
17	0.00%	96.01%	*
18	0.49%	96.50%	*
19	3.14%	99.63%	***
20	0.02%	99.66%	*
21	0.33%	99.99%	*
22	0.01%	100.00%	*

Memory Offset Distribution

Bits	Pct	Cum	
0	4.93%	4.93%	****
1	0.11%	5.04%	*
2	2.51%	7.55%	**
3	15.95%	23.50%	**********
4	24.41%	47.91%	**********
5	10.36%	58.27%	********
6	4.90%	63.17%	****

```
    7   6.89%  70.06% *****
    8   5.79%  75.85% *****
    9   4.98%  80.82% ****
   10  16.09%  96.92% ************
   11   2.38%  99.30% **
   12   0.70% 100.00% *
   13   0.00% 100.00%

Branch Displacement Distribution
 Bits    Pct     Cum
    0   0.00%   0.00%
    1   0.00%   0.00% *
    2  13.06%  13.06% **********
    3  22.20%  35.26% ****************
    4  16.58%  51.84% ************
    5  18.94%  70.79% **************
    6  15.69%  86.48% ***********
    7   6.74%  93.22% *****
    8   3.47%  96.69% ***
    9   1.63%  98.31% **
   10   0.71%  99.02% *
   11   0.27%  99.29% *
   12   0.41%  99.69% *
   13   0.01%  99.71% *
   14   0.00%  99.71%
   15   0.00%  99.71% *
   16   0.21%  99.91% *
   17   0.01%  99.92% *
   18   0.08% 100.00% *
   19   0.00% 100.00%
   20   0.00% 100.00%
   21   0.00% 100.00%
   22   0.00% 100.00%
```

Problem 2: It's time to go instruction hunting!

(*a*) The Alpha does not have a general set of double-op instructions but it does have one that can replace the two SPARC V9 instructions below. What is it? Replace the two instructions below with the Alpha instruction. (For full credit [another 0.5 point, maybe] take into account that SPARC V9 and not SPARC V8 was specified.)

```
 sll %g2, 2, %g1    ! g1 = g2 << 2;
 add %g3, %g1, %g1  ! g1 = g3 + g1

 S4ADDQ r1, r2, r3
```

(*b*) SPARC V9 does not have a full set of predicated instructions, but it does have a predicated instruction that can replace the code fragment below. What is it?

```
 subcc %g1, 0, %g0   ! Set integer condition codes.
 be SKIP             ! Branch if result equal to zero.
 nop
 add  %g3, 0, %g4    ! g4 = g3 + 0
SKIP:

 movrnz %g1, %g3, %g4
```

Problem 3: Complete Spring 2002 Homework 2 Problems 2 and 3.
(At `http://www.ece.lsu.edu/ee4720/2002/hw02.pdf`.) (The Verilog part is optional.) This is a very important type of problem, similar problems will be appearing all semester. You must solve the problem, that is, scratch your head, figure it out, and work it through. If you're stuck, feel free to ask for help. When you're done look at a solution and assign yourself a grade. Grade on a scale of 0 to 1 (real, not integer!)

Not solving it or solving it with too many glances at the solution will leave you ill-prepared for the test. Yes, you can solve it the night before the test (if you have time), but that won't help you understand everything presented in class between now and then. You have been warned.