**EE 4720** **Homework 1** Solution **Due: 9 February 2000**

**Problem 1:** Using the SPARC Architecture Manual (SAM) V8 answer the questions below. The SPARC Architecture Manual is distributed with the source for the microSPARC IIep in directory `.../models/sparc_v8/docs/pdf` of the distribution which can be downloaded from `http://www.sun.com/microelectronics/communitysource/sparcv8/`.
Alternate instructions will be given in class.

The SAM is 295 pages, so don't print it all out. It is not necessary that you understand everything in the SAM to answer these questions. See Appendix B to answer the last question.

- What size integers does the ISA support?

  8, 16, 32, and 64 bits.

- What size floating-point numbers does the ISA support?

  32, 64, and 128 bits.

- How many floating-point registers does the ISA support, how large are they, and how are the different-sized FP numbers placed in them?

  Thirty-two 32-bit registers which can be used as 16 64-bit registers or 8 128-bit registers.

- What is the binary coding of the following SPARC v8 instruction:

```
 ldsh  [%r8 + 2], %r9     ! Load signed half, r9 = Mem[r8 + 2]
```

| op | rd | op3 | rs1 | i | simm13 |
|---|---|---|---|---|---|
| 'b11 | 9 | 'b001010 | 8 | 1 | 2 |
| 31 30 | 29 25 | 24 19 | 18 14 | 13 13 | 12 0 |

**Problem 2:** Find the static and dynamic instruction count for the DLX program below. (DLX is described in Chapter 2 of the text and summarized in the last two pages. Comments, preceded by a `!`, describe what the instructions do.) The program adds up a table of numbers.

```
 lhi  r2, #0x1234      ! Load high: r2 = 0x12340000
 ori  r2, r2, #0x5678  ! r2 = r2  0x5678
 addi r4, r0, #10      ! r4 = r0 + 10,  r0 always = 0
 sub  r3, r3, r3       ! r3 = 0.  There are lots of ways to do this!
LOOP:
 lw   r1, 0(r2)     ! r1 = Mem[r2+0]
 add  r3, r3, r1    ! r3 = r3 + r1
 addi r2, r2, #4    ! r2 = r2 + 4
 subi r4, r4, #1    ! r4 = r4 - 1
 bneq r4, LOOP      ! if r4 != 0 goto LOOP
```

Static: 9 instructions.
Dynamic: $4 + 10 \times 5 = 54$.

**Problem 3:** DLX does not allow arithmetic instructions to access memory. Suppose they could and suppose all the addressing modes in Figure 2.5 of the text were available. Re-write the program to use as few instructions as possible (but still perform the same function).

Solution:

```
 lhi  r2, #0x1234      ! Load high: r2 = 0x12340000
 ori  r2, r2, #0x5678  ! r2 = r2  0x5678
 addi r4, r0, #10      ! r4 = r0 + 10,  r0 always = 0
 sub  r3, r3, r3       ! r3 = 0.  There are lots of ways to do this!
LOOP:
 add  r3, r3, (r2)+
 subi r4, r4, #1    ! r4 = r4 - 1
 bneq r4, LOOP      ! if r4 != 0 goto LOOP
```

**Problem 4:** Find the static and dynamic instruction count of the program written for the question above.

Static: 7, dynamic: $4 + 3 \times 10 = 34$.

**Problem 5:** What factors (relating to CPI and $\phi$) would one have to take into account to compare the execution time using the dynamic instruction count of the original program and the re-written program?

With the new add instruction the dynamic instruction count drops from 54 to 34. If the clock frequency and CPI of the two systems are the same execution time is $\frac{54-34}{54} \times 100\% \approx 37\%$ lower on the new system.

The CPI of the add instruction that accesses memory would likely be higher than the instructions it replaces and so the performance improvement may not be as low as the new, lower dynamic instruction count suggests.

It's also possible that to accommodate the new add instruction the clock frequency ($\phi$) had to be lowered, another reason why the lower dynamic count is optimistic.