EE 4720 **Homework 1** **Due: 18 February 1998**

The code fragment below, in C source and assembler forms, is referred to in the problems below. The distribution of elements in `array` is unknown.

```
/* C Source Code Fragment */

#define ISIZE 10
#define JSIZE 20
#define BINS 1024
short int array[ISIZE*JSIZE]; /* sizeof(short int) = 2 */
int hist[BINS];        /* sizeof(int) = 4 */

for(i=0; i<ISIZE; i++)
  for(j=0; j<JSIZE; j++)
    {
      int e = array[ i * JSIZE + j ];
      hist[ e ]++;
    }

! DLX Assembly Code Below (Simplified)
! Register usage:
! r1 = i,  r2 = j,  r3 = hist,  r4 = array

ADDI r10, r0, #20  ! r10 = JSIZE
MOVI2FP f0, r10    ! Move r10 to FP register for int mult.
ADDI r1, r0, #0    ! i=0
NEXTI:
SGEI r10, r1, #10  ! if i >= ISIZE ...
BNEZ r10, DONEI    ! ... exit loop.
ADDI r2, r0, #0    ! j=0
NEXTJ:
SGEI r10, r2, #20  ! if j >= JSIZE ...
BNEZ r10, DONEJ    ! ... exit loop
MOVI2PF f1, r1     ! Move i to FP register.
MULT f1, f1, f0    ! i * JSIZE
MOVFP2I r10, f1
ADD  r10, r10, r2  ! i * JSIZE + j
SLLI r10, r10, #1  ! ( i * JSIZE + j ) * sizeof(short int)
ADD  r10, r10, r4  ! r10 = &array[ i*JSIZE + j ]
LH   r10, 0(r10)   ! r10 = array[ i*JSIZE + j ]
SLLI r10, r10, #2  ! r10 = e = array[ i*JSIZE + j ] * sizeof(int)
ADD  r10, r10, r3  ! r10 = &hist[ e ];
LW   r11, 0(r10)   ! r11 = hist[ e ];
ADDI r11, rll, #1  ! r11 = hist[ e ] + 1;
SW   0(r10), rll   ! hist[e] = r11
ADDI r2, r2, #1    ! r2 = j+1
J NEXTJ
DONEJ:
ADDI r1, r1, #1    ! r1 = i+1
J NEXTI
DONEI:
```

Problem 1: In the program above, identify memory accesses that exhibit temporal locality, memory accesses that exhibit spatial locality, and memory accesses that may exhibit neither property.

Problem 2: Compute the dynamic and static instruction counts of the DLX program above.

Problem 3: Port the program above to an ISA derived from DLX by adding the addressing modes in Figure 2.5 of the text. In this ISA any operand can use any addressing mode. The ISA also includes an integer multiply instruction that actually uses the integer registers. The ported program should use fewer instructions than the original one, the fewer the better. What are the new static and dynamic instruction counts?

Problem 4: Suppose an implementation of the original DLX is clocked at 1 GHz, and uses 0.25 CPI. Suppose the instruction time of an implementation of the new ISA is also 0.25 CPI. At what clock frequency will the implementation of the new ISA be just as fast as the old one (based on your answers to the previous questions)?

To make headlines nowadays you need at least a 1-GHz clock frequency. An alternate implementation of the new ISA *is* clocked at 1 GHz. At what instruction execution time (CPI) will this implementation be the same speed as the original one?

Problem 5: Optional (zero credit, but you'll feel good about it and it may help you on future assignments). Type the C code above into a file and have a C compiler generate assembler code without optimization. From the assembler code, determine the static and dynamic instruction counts. How does the real assembler code differ from the code above? Compile the code with optimization and repeat.

On the Suns running Solaris 2.X, the `-S` switch directs the C compiler to produce assembler output (in file foo.s). For example,

```
[sol] % cc sample.c -S
```

puts the compiled output in assembler form in file `sample.s`. The `-fast` switch (not used) tells the compiler (and linker) to optimize (for speed).