CSE 5600 – Practice Problem Set 5

1. In this question, you will investigate how the compiler can increase the amount of ILP via the scheduling of instructions on a single-issue, in-order pipeline. Our code uses a simple loop that adds a scalar value to an array in memory. The source code (in C) looks like this:

```
for (i=1000; i > 0; i=i-1)
        x[i] = x[i] + s;
```

We can see that this loop can be executed in parallel by noticing that the body of each iteration is independent. The first step is to translate the above code segment into assembly language. In the following code segment, r1 is initially the address of the element of the array with the highest address, and f2 contains the scalar value, s. Register r2 is pre-computed, so that 8(r2) is the last element to operate on. Straightforward assembly language code, not scheduled for the pipeline, looks like this:

```
(1)     loop:   ld      f0, 0(r1)       ;f0 ← array element
(2)             addf    f4 ← f0, f2     ;add scalar in f2
(3)             sd      0(r1), f4       ;store result
(4)             addi    r1 ← r1, -8     ;decrement pointer 8 bytes (sizeof double)
(5)             bne     r1, r2, loop    ;branch if r1 != r2
```

Assume floating point additions take 4 cycles, and the 5-stage pipeline has full bypassing paths available. Assume the branch predicts "not-taken" and miss-predicted branches flush the pipeline.

(a) Show the timing of this instruction sequence (i.e., draw a pipeline diagram) without any code transformations. How many clock cycles are required per iteration? For the entire code snippet?

(b) Re-schedule the code (make sure it still performs the required computation) to diminish the time required per iteration. Show the timing of this revised instruction sequence. How many clock cycles are required per iteration? For the entire code snippet?

(c) Unroll the loop 4 times (i.e., 4 copies of the original loop are computed each iteration). You may assume r1 is initially a multiple of 32, which means that the number of original loop iterations is a multiple of 4. Eliminate any obviously redundant computations and do not reuse any of the floating point registers (you may use additional registers as needed).

(d) Show the timing of the unrolled loop. How many clock cycles are required per iteration? For the entire code snippet? (Note: you can skip some columns in the middle of the diagram if they are simply repeating an earlier pattern.)

(e) Re-schedule the unrolled loop, show the timing of this re-scheduled loop. How many clock cycles are required per iteration? For the entire code snippet?