CSE 560 – 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 is 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:   load    f0, 0(r1)         ;f0 ← array element
(2)           addf    f4 ← f0, f2       ;add scalar in f2
(3)           store   0(r1), f4         ;store result
(4)           addi    r1 ← r1, #-8      ;decrement pointer 8 bytes (sizeof double)
(5)           bneq    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?

2. Rename this instruction sequence:

mul r4, r5 $\rightarrow$ r1
add r1, r2 $\rightarrow$ r3

Map table	
r1	p1
r2	p2
r3	p3
r4	p4
r5	p5

Free-list
p6
p7
p8
p9
p10

3. Dispatch this instruction:

div p7, p6 → p1

Insn	Inp1	R	Inp2	R	Dst	Age

Ready bits	
p1	y
p2	y
p3	y
p4	y
p5	y
p6	n
p7	y
p8	y
p9	y

4. Determine which of the following instructions are ready.

Insn	**Inp1**	**R**	**Inp2**	**R**	**Dst**	**Age**
add	p3	y	p1	y	p2	0
mul	p2	n	p4	y	p5	1
div	p1	y	p5	n	p6	2
xor	p4	y	p1	y	p9	3

(a) Which will be issued on a 1-wide machine?

(b) Which will be issued on a 2-wide machine?

(c) What information will change if we issue the instruction from part (a)?

5. Using the revised pipeline diagrams presented in class, show the execution of the following instruction sequence:

div	r2 ← r3, r5
add	r1 ← r2, r4
mul	r4 ← r6, r6

You should assume that the execution units for the three instructions are as follows:

4 clocks for an add
10 clocks for a multiply
20 clocks for a divide

Start by showing the instructions after renaming, and then show the pipeline diagram for a dual-issue processor.