CSC 462/562 Computer Architecture
Homework #6: Due Wednesday, March 27

This assignment covers chapter Appendix H and section 4.5. Answer any 5 of the 7 problems. Your answer must include at least one of 1 and 2 and at least one of 6 and 7. Word process all answers.

1. For the following two loops, identify all of the true, anti and output dependences and determine if the loop is parallelizable. For each dependence, list the variable, from which instruction and to which instruction the dependence is created.

```
a. for(i=0;i<n;i++) {
        a[i] = b[i] + c[i]; // S1
        b[i+1] = a[i] * d;  // S2
        a[i]++;             // S3
   }
b. for(i=0;i<n;i++) {
        w[i] = x[i] * y[i]; // S1
        x[i]--;             // S2
        x[i+1] = z[i];      // S3
        y[i] = x[i+1] * c;  // S4
   }
```

2. Apply the GCD test on the following loops and determine if there is definitely a loop carried dependence. If the test does not prove a dependence, can you still infer one? Explain.

```
a. for(i=0;i<n;i++) a[5*i-1] = a[3*i+1]*x;
b. for(i=0;i<n;i++) b[3*i+3] = b[i-1] - 1;
c. for(i=0;i<n;i++) c[24*i+1] = c[9*i-1];
d. for(i=0;i<n;i++) d[2*i-1] = d[4*i];
```

3. In the following C code (top of next page), assume c is already in a register. In each loop iteration, either c (a register) is incremented, or a[i] is incremented. If a[i] is incremented, it is written back to memory. Let's see the consequences of speculating this code.
    a. Write the C code in RISC-V without any speculation using register x7 for c (already loaded), x8 for 1000 (the value to determine if the loop should exit), x9 for the byte offset into array a (initialized to 0, use the displacement 10000 for your memory references), and x6 for i, already initialized to 0.
    b. Revise your code speculating the if clause executes much more often than the else clause.
    c. Assuming branches take 2 cycles to execute (because of the branch penalty) and all other instructions take 1 cycle to execute (assume no stalls), how much faster does your speculated code run over your non-speculated code if the if clause executes 80% of the time?
    d. Repeat part c assuming 50%.
    e. Assume the else clause is executed more often. Why would this be harder to speculate?

```
for(i=0;i<1000;i++)
     if(c<a[i]) c++;
     else a[i]++;
```

4. Consider the following two if-else statements in C:

```
if(a < b) c = d; else c = e;
if(a < b) c = e; else d = e;
```

   Assume the if clause is much more likely than the else clause and rewrite both sets of code in RISC-V first without speculation and then by speculating that the if clause will execute more often. Next, explain why its easier to speculate in the case of the first if-else statement than the second. Use x1, x2, x3, x4, x5 for variables a, b, c, d, e respectively.

5. Explain how a predicated (or conditional) instruction can be used to avoid a branch penalty. Next, describe restrictions we need to place on the use of predicated instructions – that is, we can only create predicated instructions for certain types of conditional operations, what are the limitations? Provide an example of a type of operation (high level language code) where it would not be feasible to create a predicated instruction for, and explain why.

6. Use software pipelining to reschedule the following loop to remove all stalls. You do not need to provide the start-up and finish-up code. Assume the fmult.d takes 4 cycles to execute and the fadd.d takes 3 cycles to execute.

```
Loop: fld      f0, 0(x1)
      fld      f1, 0(x2)
      fmult.d  f2, f0, f1
      fadd.d   f4, f2, f3
      addiw    x1, x1, 8
      addiw    x2, x2, 8
      bne      x2, x3, Loop
```

7. Use the following RISC-V code and show how it will be scheduled in the EPIC. Use a figure similar to H.8b on page H-37 but you can omit all operands (show the template number and what is in each slot but without operands). Schedule the code to execute in as few cycles as possible, not as few empty slots as possible (as shown in H.8a). Assume the mult.d takes 4 cycles to compute and the fadd.d 2 cycles to compute. Compute the CPI.

```
Loop: fld      f0, 0(x1)
      fmult.d  f2, f0, f1
      fld      f3, 0(x2)
      fadd.d   f5, f3, f2
      fsd      f5, 0(x3)
      addiw    x1, x1, 8
      addiw    x2, x2, 8
      addiw    x3, x3, 8
      bne      x1, x4, Loop
```