DUE: September 28, 2005

READ:
- Chapters 1, 2, 3 (except 3.3), 4 and 5 of the Sima book
- Sections of Chapter 3, 4 and Appendix A of the Hennessy book

ASSIGNMENT: Solve all homework and exam problems as shown in class and past exam solutions.

1) Define the following terms related to parallelism and computational methods:
   a) Computational granularity, and
   b) Degree of parallelism.

2) Comment on potential for parallelism and cost-effectiveness of control-flow, data-flow and demand-driven computational methods.

3) Solve Problem 4.6 of the Hennessy book.

   After listing the dependencies, you will rewrite the **high-level language code** based on your dependence list so that there is loop-level parallelism. That is, all iterations are independent of each other such that loop body statements for all iterations can be performed in parallel. Therefore, it would not matter whether iteration 28 is done first or iteration 43, the result will be correct. Note that you will **NOT** compile the loop to an assembly code.

4) Solve Problem 4.8 (a) of the Hennessy book.

   The problem is about an assembly code that implements the DAXPY application discussed in class. The DAXPY code has loop-level parallelism. Thus, the iterations can be performed in parallel. Assume that the loop is executed **even, non-zero number** of times to simplify the solution.

   The problem assumes the **unpipelined MIPS** processor is used as there is no instruction after the BNEZ to fill in the branch delay slot. That is, this is **machine model number 0**. Also, note that according to Figure 2.31 on page 137 of the Hennessy book, there is no “DSUBUI” instruction, even though the code in the problem uses it. The code has to use the available “DADDI” instruction. See the past exam questions below for the usage of the “DADDI” instruction.
In order to solve the problem, **first** show the execution of the loop for two iterations on the **statically scheduled CS613 pipeline** (*machine model number 1*), by using Figure 4.1 latencies, as done in class. Assume that there is a NOP after the BNEZ instruction so that the delayed branch requirement is satisfied.

**Then,** modify the loop, by using unrolling and show the timing for **one** iteration. You need to modify the loop to reduce the stall cycles (delays). Stall cycles are inevitable in the original code since the compiler did not generate it by considering the latencies of instructions shown in Figure 4.1 of the textbook. So, the order of original instructions has to be changed. However, changing the order is not sufficient: unfilled delay slots result in as the loop is very short. One solution would be using NOP instructions in unfilled delay slots! A better solution is loop unrolling in which case delay slots contain useful instructions, while the loop becomes longer.

**RELEVANT QUESTIONS AND ANSWERS**

Q1) Consider the following piece of old code :

```assembly
loop :  L.D   F2, 0(R1) ; Statement S1
   ADD.D F0, F0, F2 ; S2. F0 is initially 0
   DADDI R1, R1, #(-1)10 ; S3
   BNEZ R1, loop ; S4
   S.D 0(R2), F0 ; S5
```

R1 and F0 are already initialized. Assume that R1 is initially $(16)_{10}$.

This code is written for the **unpipelined** processor, since, for example, the ADD.D is right after the L.D., even though the ADD.D depends on the L.D. Therefore, this is *machine model number 0*.

Write down the corresponding “algorithm.”

A1) The “algorithm” is as follows :

```assembly
for (i = 1 ; i <= 2 ; i++)
    k = k + X[i] ;
```

We see that the first element of “X” is pointed by R1 initially. Variable “k” is pointed by R2 when the loop is over. There are only two iterations of the loop, since initially R1 is $(16)_{10}$. R1 is decremented by 8 and compared with 0 at end the loop.
Q2) Consider the following piece of code:

\[
\begin{align*}
&k = 0.0; \\
&\text{for } (i = 1; i \leq 100; i++) \\
&\quad k = k + A[i]
\end{align*}
\]

Assume all numbers except the loop index are double-precision FP numbers.

Assuming that the arithmetic pipeline latencies are as given on page 304 of the Hennessy book and there is a one-cycle delayed branch, write down the code needed so that the loop instructions are scheduled without any delay on the statically scheduled pipeline of CS613. That is, this is machine model number 1. Make observations if necessary.

A2) This piece of code adds the elements of a vector. There is a loop-carried dependence which is circular. The loop cannot be scheduled without stalls, given the latencies and the short length of the loop.

However, the stalls can be avoided if the loop is partitioned to even and odd sections. We add even and odd numbered vector elements separately, by using two instructions: S1 and S2 below. The even and odd parts are independent and can progress in parallel. Finally, we sum the results of the even and odd parts after the loop is over:

\[
\begin{align*}
LW & \quad Rc, #(i)_{10} \quad ; \text{Rc is the loop counter register : “i”} \\
L.D & \quad F0, #0 \quad ; \text{F0 accumulates additions on even elements} \\
L.D & \quad F2, #0 \quad ; \text{F2 accumulates additions on odd elements} \\
\text{loop: } & \quad L.D \quad F4, 0(Ra) \quad ; \text{Register Ra, already initialized, points at vector A} \\
L.D & \quad F6, 8(Ra) \\
ADD.D & \quad F0, F0, F4; \quad ; \text{Statement S1} \\
ADD.D & \quad F2, F2, F6 \quad ; \text{Statement S2} \\
DADDI & \quad Rc, Rc, #(-2)_{10} \\
BNEZ & \quad Rc, loop \\
DADDI & \quad Ra, Ra, #16 \quad ; \text{Branch delay slot} \\
ADD.D & \quad F0, F0, F2 \quad ; \text{Statement S3} \\
S.D & \quad 0(Rk), F0 \quad ; \text{Statement S4. Save result in “k”}
\end{align*}
\]

Two independent instructions have to be inserted between S3 and S4 to prevent data hazards. Note again that we assume that there are even number of elements in the vector.
Q3) Consider the following piece of old code written for the unpipelined CPU (machine model number 0):

```
L.D F8, #0 ; Load value zero (0) to F8
DADDI R1, R0, #(64)10 ; Memory accesses are commented below:
loop: L.D F0, 0(RA) ; RA points at vector A
     L.D F2, 0(RB) ; RB points at vector B
     L.D F4, 0(RC) ; RC points at vector C
     MUL.D F6, F0, F2
     ADD.D F8, F8, F6
     MUL.D F10, F4, F8
     S.D 0(RC), F10 ; Stores to vector C
     DADDI R1, R1, #(-1)10
     DADDI RA, RA, #8 ; RA is advanced
     DADDI RB, RB, #8 ; RB is advanced
     DADDI RC, RC, #8 ; RC is advanced
     BNEZ R1, loop
     S.D 0(Rk), F8 ; Stores to scalar k out of the loop.
```

Write the corresponding piece of high-level code in the style of class discussions and homework assignments and also by using the names of variables mentioned in the comment section above. Mention if this is implementing an application already studied in class.

A3) The high-level code implements the dot product and another operation:

```
k = 0;
for (i = 0 ; i < 64 ; i = i + 1)
{
    k = k + (A[i] * B[i]) /* S1 : Implements the dot product */
    C[i] = k * C[i] ; /*S2 : Implements the other operation */
}
```

The code implements the dot-product operation via statement S1 in the loop. The intermediate results of the dot product operation are used for another operation (S2, the second statement in the loop).