Problem 1. The block diagrams for the five-stage pipeline in class latch the outputs of
the register file and place the forwarding mux at the inputs to the ALU. Under what
conditions will this introduce a branch delay beyond the 1 cycle delay due to the
instruction prefetch?

1.b. The forwarding logic ultimately sets the select input to the forwarding mux. What is
the required logic? (Give Boolean expression or equivalent.)

1.c. How might you reduce the branch delays of 1.a by rearranging the block diagram?
Under what conditions will your alternative organization have a larger cycle time than the
original? Smaller cycle time?

Problem 2. We skipped over the basic intro to caches in lecture, since you all seem to
know basic cache organization (associativity, write-policy, placement, etc.). So let’s
explore some of the relationships of caches and pipelines.

2.a. How is an instruction cache miss handled within the basic 5-stage pipeline?

2.b. How is a data cache miss handled?

2.c. How can the pipeline be optimized to operate at less than the hit time of the
instruction cache, if that cache is direct mapped?

Problem 3. Let’s get beyond handwavy comparisons of instruction sets. You will play
compiler for the C function below in roughly MIPS, 360, and B5000. First you’ll need to
play instruction set designer a bit and essentially bring the 360 and B5000 up to the
timeframe of the MIPS. Data values and addresses are 32 bits. You don’t need to work
out the full instruction set, just articulate the classes of instructions and their format. For
the 360, keep the registers, addressing modes, etc. What changes, if any, do you need to
accommodate 32-bit addresses? For the B5000, you’ll need to nail down the particulars
beyond the arithmetic/logic operations that work on the TOS. You need push, pop, call,
return and such. In each case, explain how local and non-local variables are managed.

```c
int x;

function foo (int A, int B)
{
```
int m = A + 3;
return ((x+A) - (m+B)) * m;
}

Compare the instruction count, total code side, and opportunities for parallelism in the three solutions.

Problem 4. An analysis technique that is often used in pipeline design, especially where a collection of statically determined flows are to be scheduled onto the pipe, is that of a reservation table. It has a row for each resource. The columns represent successive cycles. For a single initiation, the resources used during each cycle are described by a reservation table, as shown below. Here each operation uses stage 1 for 1 cycle, then stage 2 for 2 cycles, and so on. The latency is the number of cycles per initiation. The initiation interval is the number of cycles between initiations. (The MIPS 5-stage pipe would have a mark on the diagonal entries. It has a latency of 5 and an initiation interval of 1.) The structural hazards are apparent by laying copies of the reservation table over each other and checking for collisions. A new operation can be initiated if none occur.

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>x</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 2</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Stage 3</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Stage 4</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Pipeline control is much simpler if greedy initiation is optimal. As soon as resources are available, launch the next operation into the pipe.

4.a. For the reservation table above, what is the minimum initiation interval?

4.b. Prove that for any reservation table, the average initiation interval >= max number of marks in a row.

4.c. Sometimes increasing latency (adding delays) improves throughout. Modify this reservation table by introducing a delay so that it achieves the lower bound with a greedy control strategy. Explain what inserting delays in a pipeline schedule implies for the hardware.