P1. An important design aspect of pipeline architecture is the number of pipeline stages. Your design team has a choice between the following two architectures:

1. A classic 5-stage MIPS pipeline.  
   IF ID EXE MEM WB, with a cycle time of 1.8ns.
2. A 7-stage pipeline with memory access (both for instruction and data) requiring two stages. The result of the memory access is not available until the end of the second memory stage.  
   IF1 IF2 ID EXE MEM1 MEM2 WB, with a cycle time of 1.2ns.

Show dependences, forwarding and stalls for the following code on both architectures. Show it the way we did it in class, with arrows and bubbles.

```
loop: lw R3, 100(R5)
    add R6, R3, R2
    sub R9, R3, R8
    lw R1, 2000(R9)
    add R5, R4, R3
    addi R7, R1, #8
    bnez R7, loop
```

Assuming the loop runs for a long period of time, what architecture would you recommend for this code? Assume there is no branch delay slot, and the bnez is predicted correctly – thus no control flow stalls. State any other assumptions that you make.

P2. Compare the following two architectures with respect to steady-state CPI (ie, assume the branch is taken several times), for the reference code below.

1. Classic 5-stage MIPS pipeline, but with no support for forwarding (except reg-file forwarding). Also assume no branch delay slot, but rather the "assume branch not taken" approach. Assume the pipeline in figure A.24 for branch hazards.
2. Classic 5-stage MIPS pipeline, assuming normal forwarding logic. Also, assume a branch delay slot. You may move an instruction into the branch delay slot if possible, but assume no other code reordering.

```
loop: lw R1, 100(R2)
    addi R1, R1, #1
    sw R1, 0(R2)
    addi R2, R2, #4
```
P3. Consider the following piece of code. To enable greater performance, the compiler should determine what instruction could be safely moved into the branch delay slot (replacing the nop). What choice does the compiler have in the following two scenarios? Circle all instructions that could be moved into the BDS.

1. Software register renaming is enabled.
2. Software register renaming is disabled.

Assume you know nothing about the missing code (in ellipses), and no direct path from the fall through code to the taken path. Additionally, you have disabled instruction re-ordering optimizations.

```assembly
add R1, R2, R3
sub R5, R2, R3
and R7, R5, R2
lw R8, 1000(R5)
beq R7, R15, label
nop
add R6, R7, R5
sub R5, R7, R5
lw R9, 2000(R2)
... 
label: add R9, R7, R5
addi R5, R0, #0
```

P4. Suppose you’re given a code where 20% of instructions are conditional branches and 5% are jumps/procedure calls. Calculate the CPI for this code on an alternate scalar pipeline (IF ID EXE MEM WB) where the branch target is resolved at the end of ID, and the branch condition is resolved at the end of EXE. Given:

1. The architecture takes an "assume branch taken" strategy.
2. ISA uses branch delay slots for jumps/procedure calls as well as conditional branches.
3. Branches are taken 60% of the time, and we have a useful instruction in the branch delay slot 75% of the time.

P5. Using the web, find the number of pipeline stages for the following processors: Intel Haswell, Intel Atom (Bonnell), AMD Bobcat, ARM Cortex-A7 and ARM Cortex-A15. Make sure to mention your source.