Computer Architecture
Homework #7 Solution

1. How could we modify the following code to make use of a delayed branch slot?

```
Loop: lw $2, 100($3)
      addi $3, $3, 4
      beq $3, $4, Loop
```

Obviously either the `lw` or the `addi` must occupy the branch delay slot. We can't just put the `addi` into the slot because the branch instruction needs to compare $3 with register $4 and the `addi` instruction changes $3. In order to move the load (`lw`) into the branch delay slot, we must realize that $3 will have changed. If you like, you can think if this as a two-step transformation:

First, we rewrite the code as follows:

```
Loop: addi $3, $3, 4
      lw $2, 96($3)
      beq $3, $4, Loop
```

Then, we can move the load into the branch delay slot:

```
Loop: addi $3, $3, 4
      beq $3, $4, Loop
      lw $2, 96($3) # branch delay slot
```

2. Exercise 6.4

```
   add $3, $4, $2
   sub $5, $3, $1
   lw $6, 200($3)
   add $7, $3, $6
```

There is a data dependency through $3 between the first instruction and each subsequent instruction. There is a data dependency through $6 between the `lw` instruction and the last instruction.

For a five-stage pipeline as shown in Figure 6.7, the data dependencies between the first instruction and each subsequent instruction can be resolved by using forwarding.
The data dependency between the load (`lw`) and the last `add` instruction cannot be resolved by using forwarding and the data hazard will cause a stall.

3. For each pipeline register in Figure 6.22 on page 400, label each portion of the pipeline register with the name of the value that is loaded into the register. Determine the length of each field in bits. For example, the IF/ID pipeline register contains two fields, one of which is an instruction field that is 32 bits wide.

Consider each register:
. IF/ID holds PC+4 (32 bits) and the instruction (32 bits) for a total of 64 bits.

. ID/EX holds PC+4 (32 bits), Read data 1 and 2 (32 bits each), the sign-extended data (32 bits), and two possible destinations (5 bits each) for a total of 138 bits.
. EX/MEM holds PC target if branch (32 bits), Zero (1 bit), ALU Result (32 bits), Read data 2 (32 bits), and a destination register (5 bits) for a total of 102 bits.
. MEM/WB holds the data coming out of memory (32 bits), ALU Result (32 bits), and the destination register (5 bits) for a total of 69 bits.

Note: It is OK to add control bits and have slightly different answers.

. Control signals for EX stage (RegDst, ALUOp1, ALUOp0, and ALUSrc) = 4 bits
. Control signals for MEM stage (Branch, MemRead, and MemWrite) = 3 bits
. Control signals for WB stage (RegWrite, MemtoReg) = 2 bits
So
. IF/ID = 64 bits
. ID/EX = 138 + 4 + 3 + 2 = 147 bits
. EX/MEM = 102 + 3 + 2 = 107 bits
. MEM/WB = 69 + 2 = 71 bits

4. Exercise 6.18

```
add $2, $3, $1
sub $4, $3, $5
add $5, $3, $7
add $7, $6, $1
add $8, $2, $6
```

The forwarding unit is seeing if it needs to forward. It is looking at the instructions in the fourth and fifth stages and checking to see whether they intend to write to the register file and whether the register written is being used as an ALU input. Thus, it is comparing $3 = $4? $3 = $2? $7 = $4? $7 = $2?

5. Exercise 6.19

The hazard detection unit is checking to see whether the instruction in the ALU stage is an `lw` instruction and whether the instruction in the ID stage is reading the register that the `lw` will be writing. If it is, it needs to stall. If there is an `lw` instruction, it checks to see whether the destination is register 6 or 1 (the registers being read).

6. Exercise 6.22

It will take 8 cycles to execute this code, including a bubble of 1 cycle due to the dependency between the `lw` and `sub` instructions.