EECS 370
Homework #3

Problem #1 [4 points]:

a) What is the average CPI (clocks per instruction) for a standard 5 stage LC-2K4 pipeline (as discussed in lecture) assuming all data dependencies can be handled by forwarding, every fifth instruction is a branch, and branches are predicted correctly 50% of the time.

b) What would be the average CPI if branches are predicted correctly 75% of the time?

c) What would be the average CPI if branches are predicted correctly 95% of the time?

Problem #2 [4 points]:

Rewrite the following LC-2K4 program so that it is functionally the same (at the end of your function all the registers and data memory locations would be exactly the same) but requires as few cycles as possible to execute on the 5 stage LC-2K4 pipeline discussed in lecture. Assume all branches are predicted as not taken and forwarding takes place wherever possible. The label "offset" is at some location unknown to you. Only modify registers and memory that are modified in the original program. You must also provide a count of how many cycles your program requires for execution.

```
loop    lw      0    2    one
        lw      0    1    two
        lw      5    3    offset
        nand    3    3    3
        sw      5    3    offset
        add     5    2    5
        beq     1    5    done
        beq     0    0    loop
done    halt
one     .fill   1
two     .fill   2
```

```
*****************************************************************
* A Note on Problems 3 and 4                                    *
* "LC-2 pipeline as presented in class" means that registers are*
* written on the first half of a cycle, and read on the second       *
* half of a cycle. This applies to problems 3 and 4. Although*
* other forwarding is disabled, this internal forwarding in the      *
* register file is still allowed.                               *
*****************************************************************
```

Problem #3 [4 points]:

Suppose we use the 5 stage pipeline presented in class, with no data hazard detection, control hazard detection, or forwarding. Insert nops into the code below to ensure correct execution. (You are not allowed to reorder the original instructions.) Also calculate the run time, in cycles, of your LC2k4 code after inserting nops.

Assume the following values before start of this code:

```
        Reg 1 = 1
        Reg 2 = 1
        Reg 3 = 0
        Reg 4 = 0
        Reg 6 = 28

start   add 3 1 3
        add 3 2 3
        lw  0 4 value
        add 4 4 4
        add 3 4 3
        beq 3 6 stop
        add 3 4 3
        beq 3 3 start
stop    add 3 3 3
        halt
value   .fill 1
```

Problem #4 [4 points]:

Consider a pipelined LC-2 implemented similar to the lecture slides that resolves data hazards by stalling rather than forwarding. For the following code fragment, indicate which cycle each instruction completes on (i.e., the number of the cycle in which the instruction is in the writeback WB stage). Assume that cycle 1 is the fetch of the first "add" instruction. The first two instructions are already done for you.

Instruction	Completes on cycle
add 4 5 6	5
nand 4 5 6	6
add 1 2 3	___
lw 3 4 2	___
sw 6 3 7	___
add 4 1 2	___
nand 7 1 4	___

```
nand  2  3  5        ___

sw    2  5  4        ___

add   4  3  2        ___

lw    2  2  2        ___

halt                 ___
```

Problem #5 [4 points]

The alpha version of the LC2K4 processor did not support using immediate values in the main ALU as operands, Hence, the LC2K4-Alpha had an extra stage (with its own special ALU) in the pipeline: AC (Address Calculation). Address calculations (a base register + an immediate offset) are done in this stage. Consider the following two possible pipelines for LC2K4-Alpha:

```
IF ID AC EX MEM WB
IF ID EX AC MEM WB
```

Assuming each pipeline has complete data forwarding paths, which performs better for the following program? How many cycles does each style require for the following program?

```
add  1 2 3
lw   3 4 100
add  4 3 7
lw   7 5 75
sw   5 7 100
```

Problem #6 [4 points]

(a) For the following LC-2 code running on the pipeline presented in class, identify which dependencies are data hazards that will be resolved via forwarding? Please mark the instructions to indicate which ones acquire one or more inputs from the EX-to-EX stage bypass (e.g., pulling the value out of EX/MEM pipeline register, if so, label the instruction with an "E") or the MEM-to-EX stage bypass ("M")? (For the first part of this problem, there is no MEM-to-MEM stage bypass.) Also mark which instructions must stall ("L") even with the forwarding support?

```
add  1  2  3
add  3  1  2
lw   3  1  100
sw   3  1  100
add  3  2  4
lw   4  6  100
```

(b) Complete the problem again, this time adding a MEM-to-MEM stage bypass (permitting a load result to be stored to memory in the next cycle). Indicate where an instruction receives an operand from the MEM-to-MEM bypass ("S").