EE 4720 Homework 2 Due: 5 November 2001

Problem 1: Answer the following questions about the MIPS Technologies 4Km processor core. The processor is documented in
`http://www.mips.com/declassified/Declassified_2000/MD00016-2B-4K-SUM-01.15.pdf`.

(*a*) For each stage in the statically scheduled DLX implementation show where the same work is done in the 4Km pipeline. Note that work done in one DLX stage might be performed in more than one 4Km pipeline stage.

(*b*) The 4Km documentation uses the term *stall* differently than used in class. How do their usages differ? What term does the documentation use that is close to stall as used in class? (See section 2.8.1)

(*c*) A MIPS implementation needs to do all of the following:

(1) arithmetic and logical operations for ordinary instructions
(2) compute the target of a branch
(3) compute the effective address of a load or store

In the first pipelined DLX implementation all of these were performed by the ALU. MIPS has a branch instruction in which a branch is taken if two registers are equal (`beq`) or not equal (`bne`). So it must also

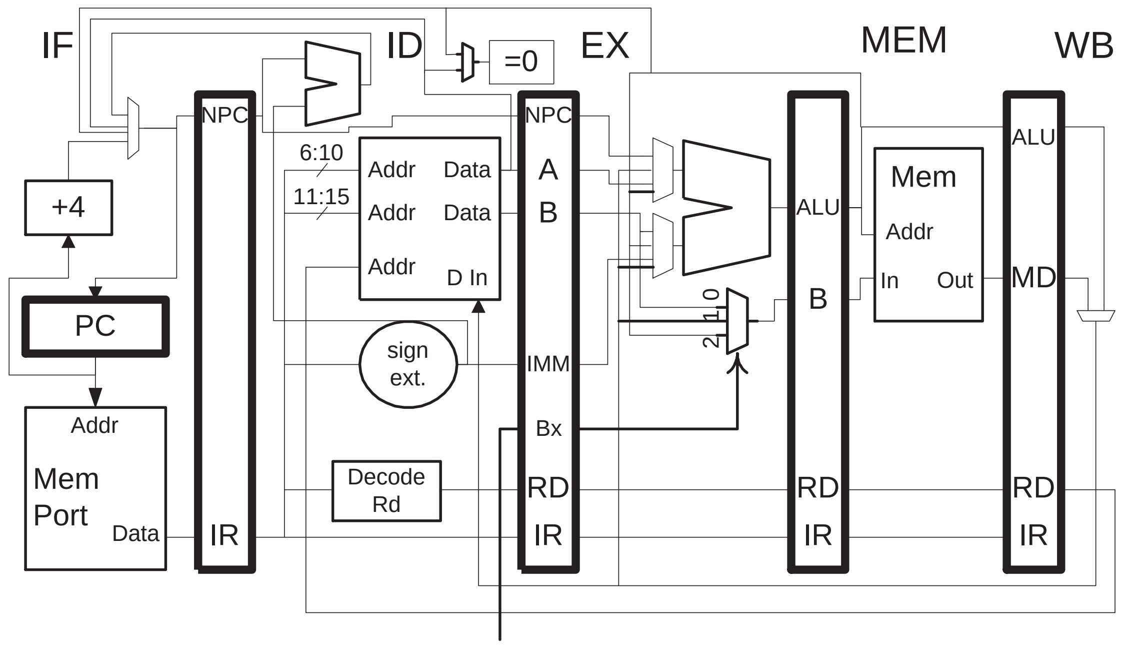
(4) determine if two values are equal

How many of these are shared? If they are not shared, why not? (The documentation does not state exactly what hardware is present, answer the question by looking at how instructions execute.)

Problem 2: The program below runs on the DLX implementation shown below. The hardware makes no special provisions for the tricky technique used. The coding for a `nop` (actually `add r0, r0, r0`) is all zeros.

Why isn't this an infinite loop? (For those who know why it matters, assume there is no cache.)

Why will the code run for at least one iteration?



```
LOOP:
 lw r1, 0(r2)
 addi r2, r2, #4
 add r3, r3, r1
 sw 0x100(r0), r0
LINE:  LINE = 0x100
 j LOOP
```

Problem 3: Show a pipeline execution diagram for the code below running on a 4-way statically scheduled superscalar processor. All needed bypass paths are available, including one for the branch condition. Determine the CPI for a large number of iterations.

```
 and r2, r2, r8
LOOP:  ! LOOP = 0x1008
 lw r1, 0(r2)
 add r3, r3, r1
 addi r2, r2, #4
 sub r4, r2, r5
 bneq r4, LOOP
```

Problem 4: The code from the problem above can be improved (stalls can be removed) to a small extent by scheduling, but that would still leave some stalls. This might see like a good candidate for loop unrolling.

(*a*) Show why it would take alot of unrolling to eliminate all stalls. (You don't have to show the unrolled code, since it would be long.)

(*b*) Use software pipelining and scheduling to remove the stalls. (Hint: to software pipeline switch the `lw` and `add` instructions, and make any other necessary changes.) What is the CPI for a large number of iterations of the modified code?

(*c*) Would loop unrolling provide further gains?