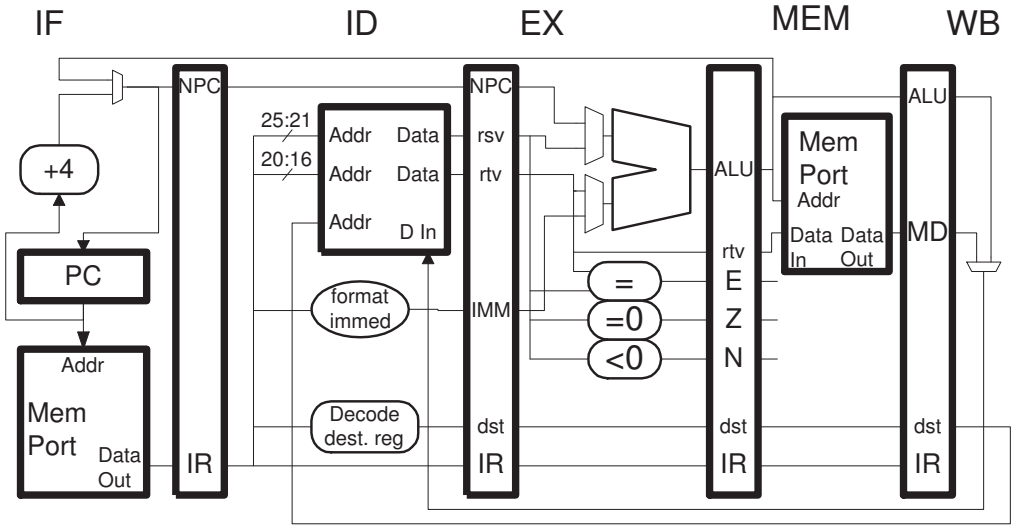
LSU EE 4720 **Homework 1** Solution **Due: 6 February 2013**

Problem 1: The MIPS code below executes on the illustrated implementation. The loop iterates for many cycles. The register file bypasses data from the write ports to the read port in the same cycle.




```
LOOP:
 lw r2, 0(r4)
 slt r1, r2, r7
 bne r1, r0 LOOP
 addi r4, r4, 4
 sw r4, 0(r6)
 jr r31
```

(*a*) Show the execution of the code above on the illustrated implementation up to and including the first instruction of the second iteration.

- Carefully check the code for dependencies.
- Be sure to stall when necessary.
- Pay careful attention to the timing of the fetch of the branch target.

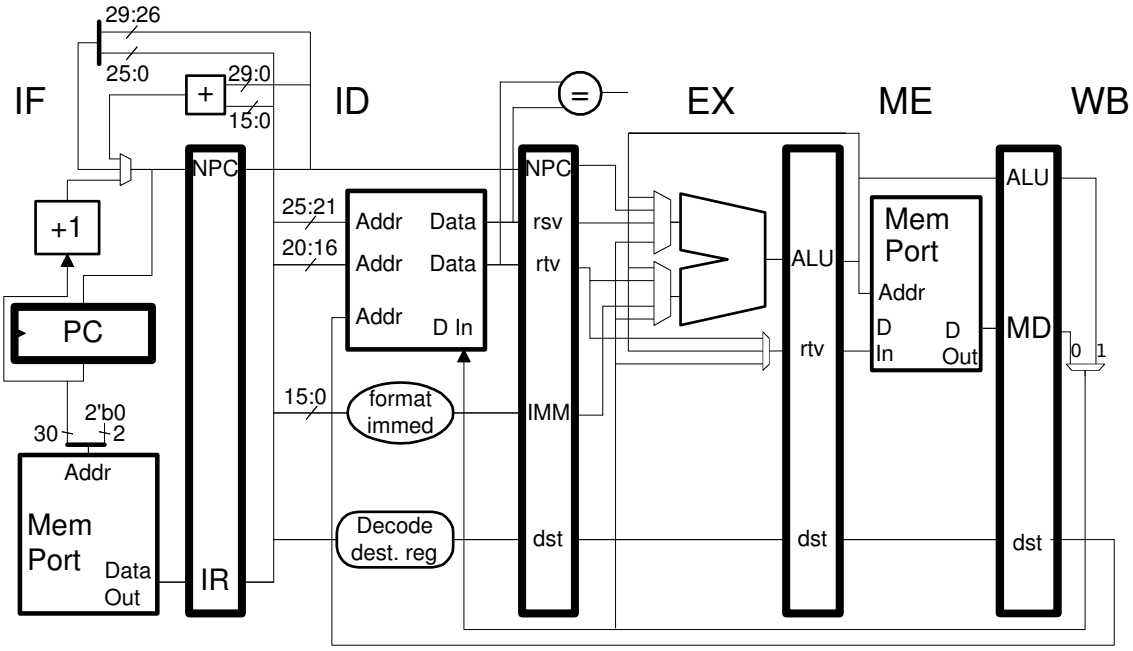
Solution appears below. Notice that there is a dependence between the `slt` and the `bne` (sometimes people forget that branches can have dependencies too).

```
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r2, 0(r4)      IF ID EX ME WB
 slt r1, r2, r7       IF ID ----> EX ME WB
 bne r1, r0 LOOP         IF ----> ID ----> EX ME WB
 addi r4, r4, 1                   IF ----> ID EX ME WB
 sw r4, 0(r6)                              IF IDx
 jr r31                                       IFx
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r2, 0(r4)                                    IF ID EX ME WB
```

(*b*) Compute the CPI for a large number of iterations.

Recall that we define an iteration to start when the first instruction is in `IF`. In the execution above the first iteration starts in cycle 0 and the second iteration starts in cycle 10, and so an iteration takes 10 cycles. There are four instructions in an iteration, so the CPI is $\frac{10}{4} = 2.5\,\mathrm{CPI}$.

Problem 2: The code fragment below is the same as the one used in the last problem, but the implementation is different (most would say better).




```
LOOP:
 lw r2, 0(r4)
 slt r1, r2, r7
 bne r1, r0 LOOP
 addi r4, r4, 4
 sw r4, 0(r6)
 jr r31
```

(*a*) Show the execution of the code on this new implementation.

- There will still be stalls due to dependencies, though fewer than before.

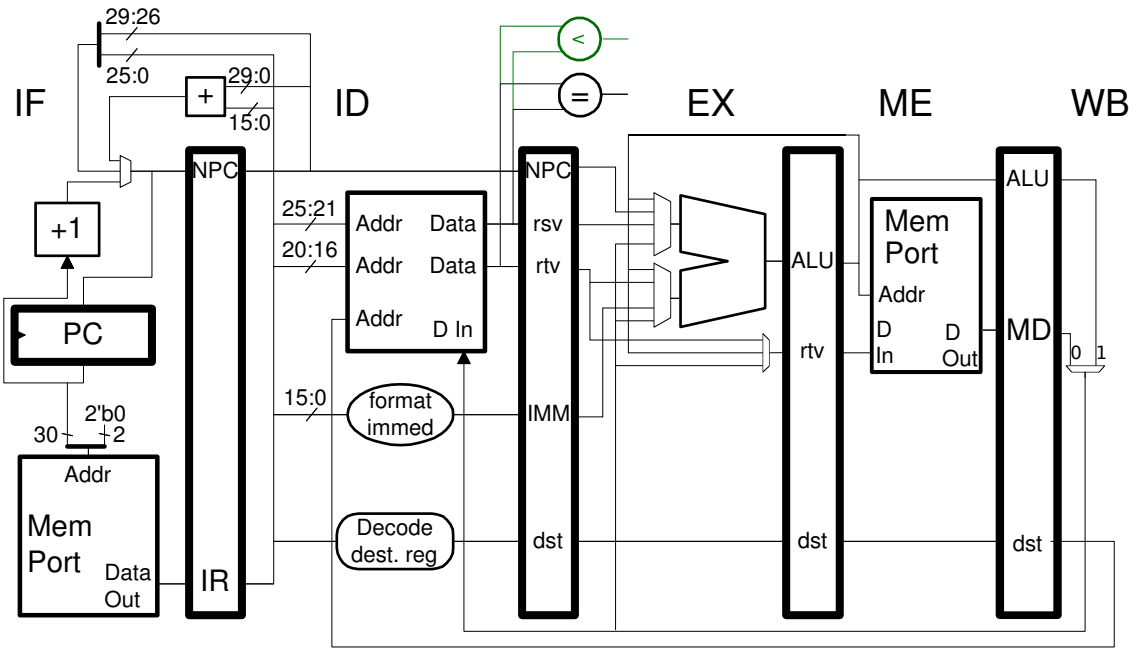
Solution appears below. Notice that fewer stalls are eliminated than one might have hoped because the load value is available later in the pipeline and because there are no bypasses for the branch.

```
 # SOLUTION
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r2, 0(r4)      IF ID EX ME WB
 slt r1, r2, r7       IF ID -> EX ME WB
 bne r1, r0 LOOP         IF -> ID ----> EX ME WB
 addi r4, r4, 1                IF ----> ID EX ME WB
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r2, 0(r4)                           IF ID EX ME WB
```

(*b*) Compute the CPI for a large number of iterations.

The iteration time is now 7 cycles, and so the CPI is $\frac{7}{4} = 1.75\,\text{CPI}$.

Problem 3: Consider once again the code fragment from the previous two problems, and the implementation from the previous problem. In this problem consider a MIPS implementation that executes a `blt` instruction, an instruction that is not part of MIPS. With such an instruction the code fragment from the previous problems can be shortened, and one would hope that the code would take less time to run. In this problem rather than hope we'll figure it out.



(*a*) Add the additional datapath (non-control) hardware needed to execute `blt`. *Hint: Just add one unit and a few wires.*

Solution appears above in green, where in the `ID` stage a less-than comparison unit was added above the equality unit.

(*b*) Show the execution of the code on the illustrated implementation up until the second fetch of `lw`.

Solution appears below. It looks like we saved two cycles by eliminating the `slt` and the stall it suffered.

```
# SOLUTION
LOOP: # Cycles      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r2, 0(r4)       IF ID EX ME WB
 blt r2, r7  LOOP      IF ID ----> EX ME WB
 addi r4, r4, 4           IF ----> ID EX ME WB
LOOP: # Cycles      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r2, 0(r4)                     IF ID EX ME WB

 sw r4, 0(r6)
 jr r31
```

(*c*) As we discussed in class, doing a magnitude comparison in `ID` might stretch the critical path, forcing a reduction in clock frequency. Suppose the clock frequency without `blt` is 1 GHz. At

what clock frequency will the `blt` implementation, the one in this problem, be just as fast as the implementation from the prior problem on their respective code fragments?

- Be sure to pick a sensible meaning of *just as fast*. **Do not** define just-as-fast in terms of CPI.

Most would agree that the important measure of computer performance is how long it takes to finish your program. We have two implementations, the original bypassed MIPS, and the `blt` version. Lets call the code fragments used in this assignment our programs. The first program in this assignment is for the original MIPS, the program with `blt` is for the `blt` version of MIPS. What's important to us is how long it takes to run these programs on their respective implementations.

Lets assume that in a run of either program the loop iterates 1000 times. The original MIPS implementation takes 7 cycles per iteration, for a total of 7000 cycles, and that corresponds to a time of $t_{\text{original}} = \frac{7000\,\text{cycles}}{\phi_{\text{orig}}} = \frac{7000\,\text{cycles}}{1\,\text{GHz}} = 7\,\mu\text{s}$, where $\phi_{\text{orig}} = 1\,\text{GHz}$ is the clock frequency of the original implementation.

In the `blt` version of MIPS an iteration takes only 5 cycles and so execution takes 5000 cycles. Execution time is $t_{\text{blt}} = \frac{5000\,\text{cycles}}{\phi_{\text{blt}}}$, where ϕ_{blt} is the clock frequency of the blt version. For this problem we need to find a value of ϕ_{blt} that will make the execution time of the blt version $7\,\mu\text{s}$. That is we need to solve $\frac{5000\,\text{cycles}}{\phi_{\text{blt}}} = 7\,\mu\text{s}$, for ϕ_{blt}, which is $\phi_{\text{blt}} = 714\,\text{MHz}$.

This means that if the hardware needed to implement `blt` slows down the clock frequency, but the clock frequency is still $> 714\,\text{MHz}$ the `blt` implementation will be faster.

Note that one cannot just look at something like CPI, since that would ignore the fact that the two different programs execute a different number of instructions.

(*d*) Explain why the code fragments in these problems might exaggerate the benefit of the `blt` instruction.

The code fragment was short, could use a `blt`, and the `blt` reduced execution by two cycles rather than the one cycle it would for other cases. If we look at other code samples we will probably find that a `blt` is usable less frequently than every five instructions, in part because not every branch is based on a magnitude comparison (see the example below).

That means the actual reduction in the number of cycles will not be as great as 7 to 5. Suppose the number of clock cycles drops from 100 trillion to 98 trillion. In that case, we can tolerate a much smaller drop in clock frequency and still get better performance.

Grading Notes: Many answered that the benefit is exaggerated because the clock frequency would be lower. That is the right answer to a different question. This question asked about the code fragments *in these problems*. The lower clock frequency would affect any code.

```
LOOP:
 lw r2, 0(r4)
 beq r2, r7  LOOP  # blt won't help here.
 addi r4, r4, 4
 sw r4, 0(r6)
 jr r31
```