LSU EE 4720 Homework 1 Solution Due: 27 February 2009

Problem 1: Answer each question.

(*a*) Explain why the code below won't finish running.

```
LOOP:
 lw r1, 0(r2)
 xor r3, r3, r1
 bne r2, r4 LOOP
 addi r2, r2, 2
```

The `lw` effective address must be a multiple of four (the address alignment restriction) but it can't always be in the code fragment above since `r2` is incremented by 2 each iteration. The code won't finish because the `lw` will raise some kind of address misalignment exception at either the first or second iteration.

(*b*) Shorten the code below.

```
 lui r1, 0x1234
 ori r1, r1, 0x5678
 lw  r1,0(r1)
```

```
 # Solution
 lui r1, 0x1234
 lw  r1,0x5678(r1)
```

(*c*) Shorten the code below.

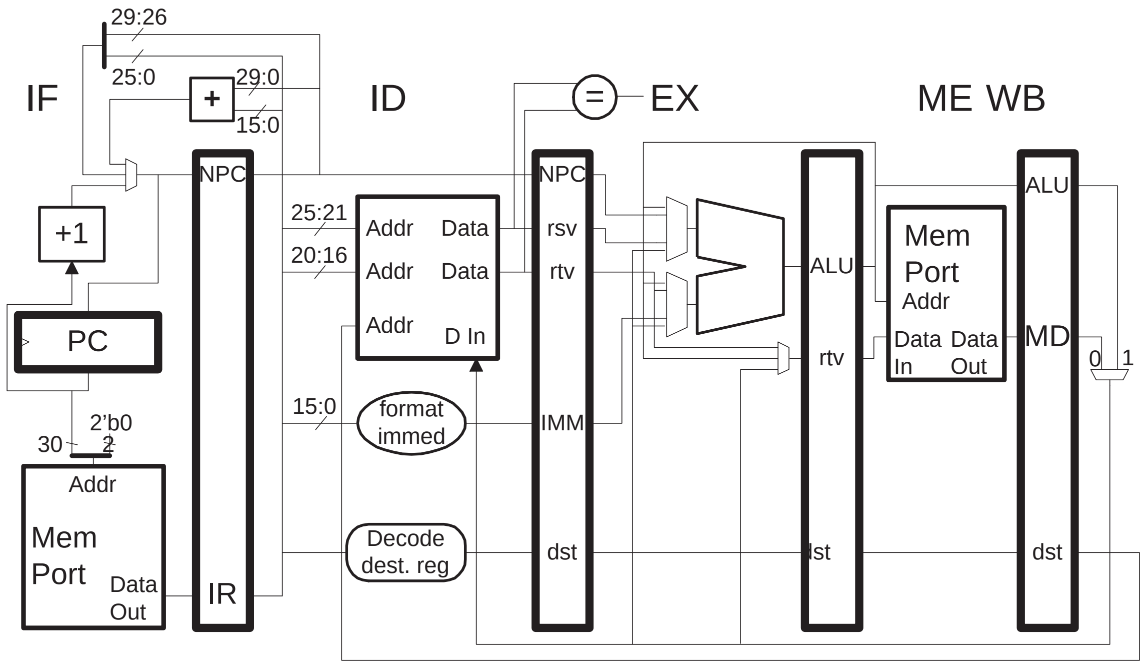
```
 xor r1, r2, r3
 beq r1, r0  TARG
 addi r1, r4, 1
```

In the code above `r1` will be zero only if `r2` is equal to `r3`, so there is no need for the `xor`. Note: In the original assignment the last instruction did not modify `r1`, so one could not safely remove the `xor`.

```
 # Solution.
 beq r2, r3  TARG
 addi r1, r4, 1
```

Problem 2: Consider the execution code below on the illustrated implementation.

```
LOOP:
 lw  r2, 0(r4)
 slt r1, r2, r3
 beq r1, r0 LOOP
 addi r4, r4, 4
```



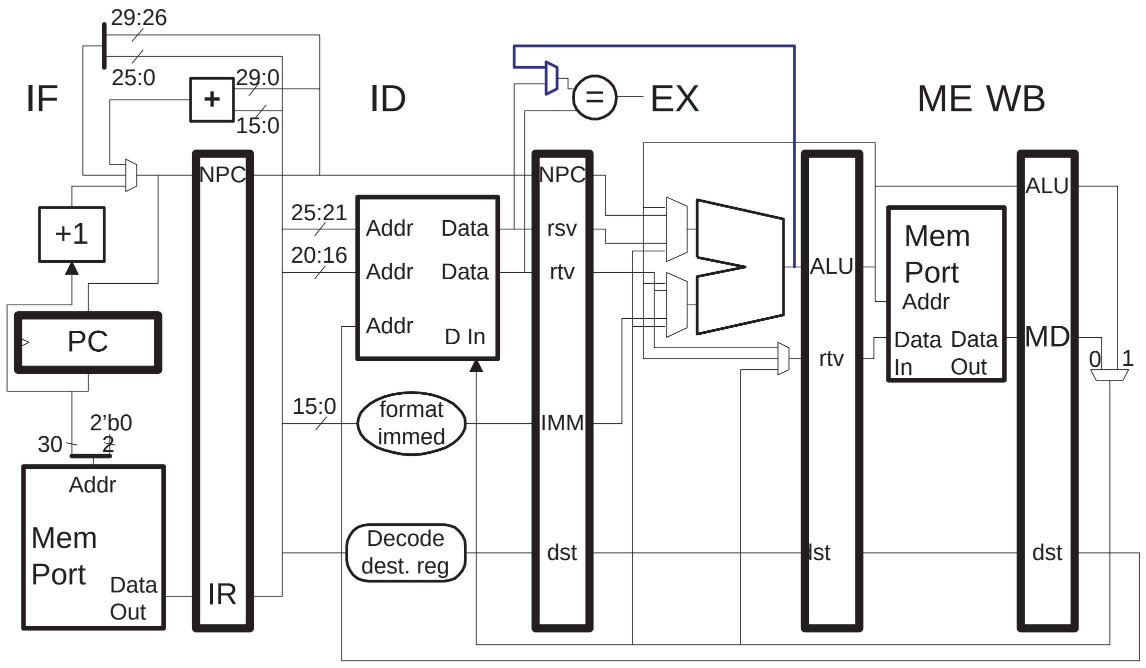
(*a*) Determine the execution rate in IPC (instructions per cycle) assuming a large number of iterations. Use a pipeline execution diagram to justify your answer. (No credit without one.)

```
LOOP:
 # Solution
 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
 lw  r2, 0(r4)    IF ID EX ME WB
 slt r1, r2, r3      IF ID -> EX ME WB
 beq r1, r0 LOOP        IF -> ID ----> EX ME WB
 addi r4, r4, 4               IF ----> ID EX ME WB
 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
 lw  r2, 0(r4)                        IF ID EX ME WB
 slt r1, r2, r3                          IF ID -> EX ME WB
 beq r1, r0 LOOP                            IF -> ID ----> EX ME WB
 addi r4, r4, 4                                   IF ----> ID EX ME WB
 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
 lw  r2, 0(r4)                                              IF ID EX ME WB
 ...
```

The code suffers two stalls, the first because the `slt` needs the `lw` value and the second (a two-cycle stall) because the `beq` needs the `slt` value. Iterations start (first instruction of the loop is in `IF`) in cycles 0, 7, and 14. Since the pipeline is in the same state in cycles 7 and 14 (`lw` in `IF`, `addi` in `ID`, and `beq` in `EX`) we can expect the iteration that starts at 14 to be identical to the one that starts at 7. The time for these iterations is $14 - 7 = 7$ cycles, and so the execution rate is $\frac{4}{7}$ IPC(or if you prefer, the instruction initiation interval is $\frac{7}{4}$ CPI).

(*b*) If the previous part was solved correctly there should be a stall due to the branch. Add a bypass path to avoid the branch stall.

The added bypass path appears below in blue. This bypass path eliminates the stall but would likely lower the clock frequency. (See the next problem.)



```
LOOP:
 # Solution - Execution with the bypass.
 # Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
 lw  r2, 0(r4)     IF ID EX ME WB
 slt r1, r2, r3       IF ID -> EX ME WB
 beq r1, r0 LOOP         IF -> ID EX ME WB
 addi r4, r4, 4                IF ID EX ME WB
```

(*c*) Why might the added bypass path impact clock frequency?

The `slt` at the ALU output would be ready late in the cycle, and these signals would still have to pass through the ID-stage comparison unit, then some control logic, the IF-stage mux, finally reaching the PC input. Since it's reasonable that the critical path passed through the ALU without this bypass, adding the bypass would increase the critical path and therefore reduce clock frequency. (A solution that bypassed from ME to ID would not impact clock frequency [unless it were taken from the memory port output] but it would only reduce the number of stall cycles from 2 to 1.)

(*d*) Suppose the clock frequency of the original pipeline were 1 GHz, and call the clock frequency of the added-bypass implementation ϕ. For what value of ϕ will the run time of the code fragment be the same on the original and added-bypass implementations (assuming a large number of iterations).

Without the bypass the code executes at $\frac{4}{7}1\,\text{GHz}$ IPS (instruction per second). With the bypass the code will execute at a rate of $\frac{4}{5}$ IPCor $\frac{4}{5}\phi$ IPS. Solving $\frac{4}{7}1\,\text{GHz} = \frac{4}{5}\phi$ yields $\phi = \frac{5}{7}\,\text{GHz}$.

(*e*) Suppose a `blt` (branch less than) instruction was available that could compare two registers (not just a register to zero). Re-write the code above for this instruction and add bypasses that are no worse than the added-bypass bypass. How would the performance of this `blt` implementation on the re-written code fragment compare to the added-bypass implementation on the original code fragment? Assume both systems have the same clock frequency.

The bypass needed for this part would be from `ME` to `ID`, since one value to compare arrives at through memory port. The execution rate of the original code on the bypassed pipeline is 5 cycles per loop iteration. The code with `blt` still suffers one stall and so executes at 4 cycles per iteration (see diagram below). The net result is improved performance. Note that the speedup (performance ratio) is $\frac{5}{4} = 1.25$ while the improvement in IPC is only $\frac{4/5}{3/4} = 1.0667$.

```
 # Solution: Code using blt
LOOP:
 lw  r2, 0(r4)       IF ID EX ME WB
 blt r2, r3 LOOP        IF ID -> EX ME WB
 addi r4, r4, 4            IF -> ID EX ME WB
```