Problem 1: Answer each question.

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

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

(b) Shorten the code below.

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

(c) Shorten the code below.

```assembly
xor r1, r2, r3
beq r1, r0 TARG
addi r4, r4, 1
```
Problem 2: Consider the execution code below on the illustrated implementation.

```assembly
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.)

(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.

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

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

(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.