1.)

Loop:

```
addi $t1, $a0, 4
addi $v0, $v0, 1
lw $t0, 0($t1)
add $a0, $t0, $a1
bne $t0, $0, $a1
addi $a0, $a0, 4
jr $ra
```

This eliminates the stall and avoids faults

2.)

addi $t1, $a0, 4 & addi $v0, $v0, 1
lw $t0, 0($t1)
stall
add $a0, $t0, $a1 & bne $t0, $0, $a1
addi $a0, $a0, 4
jr $ra

Since the addi and the add and bne don't depend on each other, they can execute static dual issue

3.)

Depending on how we want to manipulate the way we want to change how an instruction was being executed, we will need to increase the PC and just add one more.

4.)

The problem we will come when we hit the BNE this can either be resolved in the Execute or in the ID so as long as there are two stalls before the BNE statement.

5.)

The problem we will come when we hit the BNE instruction because the correct value for the BNE will be wrong. So in order to prevent this we will need to correct it with stalls before the BNE

6.)

As stated above, we can add stalls to move MEM, but if we add stalls in either ID or EX we can avoid the problem of not having the correct values in BNE.

7.)

Refer to number 6 for this answer.

8.)

There will be conflicts of when data is being accessed and how it will be manipulated. This can be avoided by adding stalls or completely avoid this method all together.