1.

.var 2 is b1 Z from L0 to L1
.var 3 is b2 Z from L0 to L1
.var 4 is i I from L0 to L1
    iload_2
    ifeq L5
    iload_3
    ifeq L5
    iconst_1
    goto L6
L5:
    iconst_0
L6:
    ifeq L3
    iconst_0
    istore 4
    goto L4
L3:
    iconst_1
    istore 4
L4:
L1:

2.

(i)

.var 2 is b1 Z from L0 to L1
.var 3 is b2 Z from L0 to L1
.var 4 is i I from L0 to L1
    iload_2
    ifeq L5
    iload_3
    // ifeq L5 // two instructions can be removed
    // iconst_1
    goto L6
L5:
    iconst_0
L6:
    ifeq L3

   1
(ii) The second template works for VC because

- true is always represented by 1 and false by 0,
- boolean expressions and other expressions cannot be mixed

3.

The general technique for generating better code for boolean expressions is discussed in Section 8.4 of the Dragon Book.

Both the original and modified code templates for Boolean expressions works independently of the context in which an boolean expression is used. Of course, the modified version is preferred since two more instructions are saved.

To generate optimised code for an boolean expression, in general, we need to know the context in which the boolean expression is used. For example, if we know that \( b_1 \&\& b_2 \) is the RHS of the following assignment:

\[
b = b_1 \&\& b_2;
\]

we would generate:

```
.var 2 is b1 Z from L0 to L1
.var 3 is b2 Z from L0 to L1
.var 4 is i I from L0 to L1
.iload_2
.ifeq L5
.iload_3
.ifeq L5
.icnst_1
.goto L6
L5:
.icnst_0
L6:
.istore 4
```

There is one major difference between the optimised code for the if statement given in Q1(c) and the optimised code sequence for the assignment given above. The result of \( b_1 \&\& b_2 \) is left on the operand stack in the latter but not in the former! Thus, the optimal code generation is context-dependent.