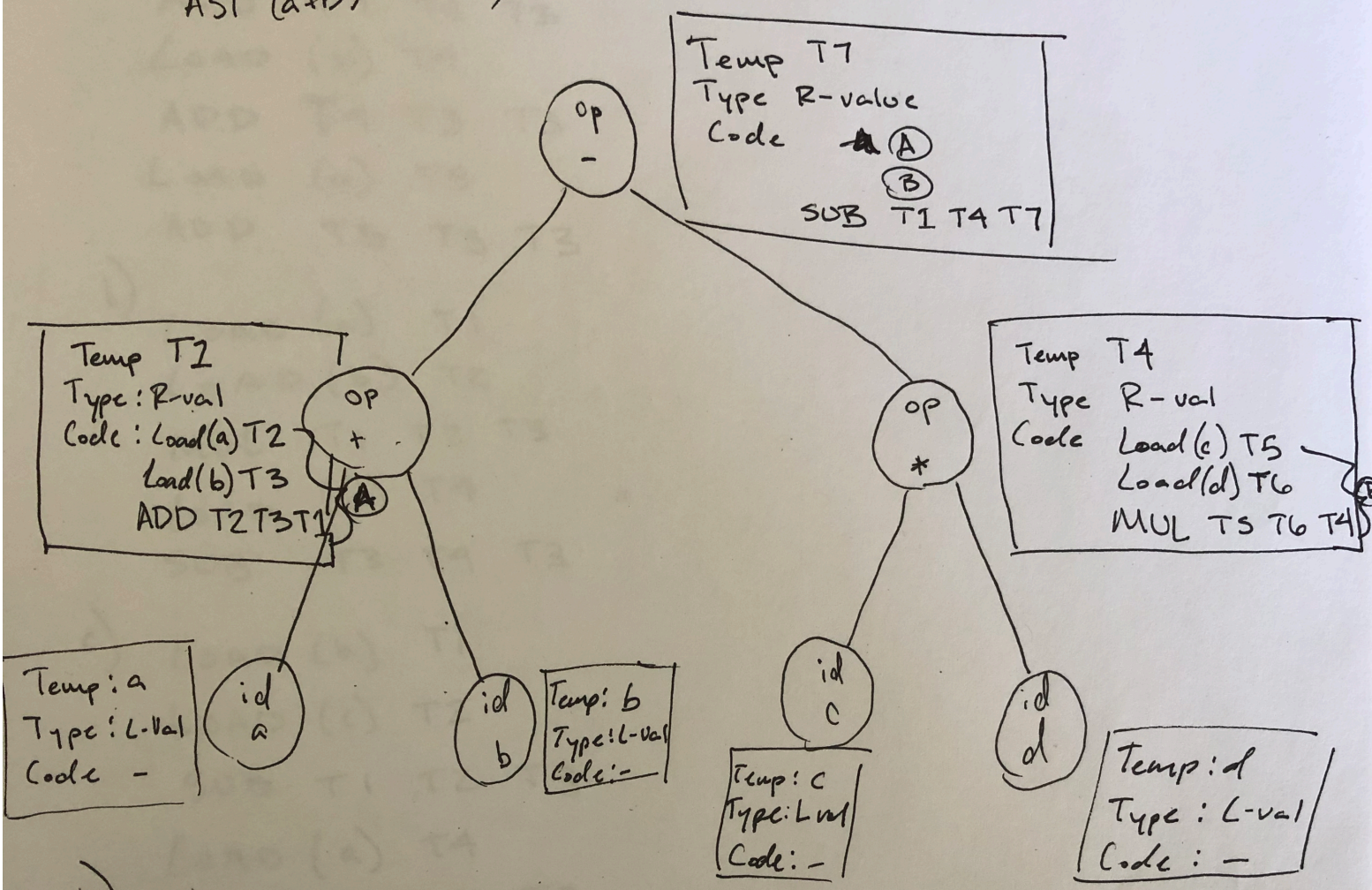
1)

a)

AST (a+b) - (c*d)



b) L-value : An L value is an object that stores data that is representative of Literals.

R-value : An R value is an operation (addition subtraction...) that is performed on L-values and literals.

c) LOAD (a) T2
LOAD (b) T3
ADD T2 T3 T1
LOAD (c) T5
LOAD (d) T6
MUL T5 T6 T4
SUB T1 T4 T7

2) Load (c) T1

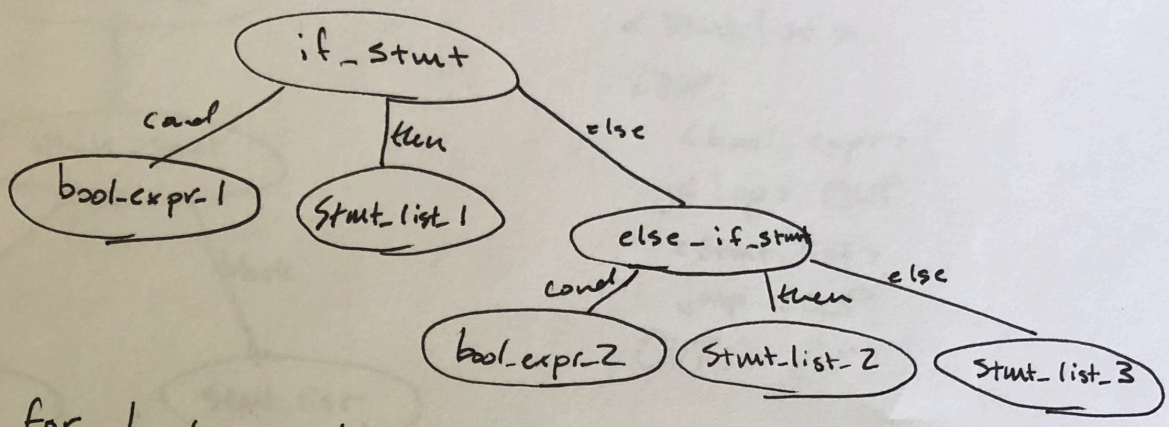
a) LOAD (d) T2
ADD T1 T2 T3
LOAD (b) T4
ADD T4 T3 T3
LOAD (a) T5
ADD T5 T3 T3

b)
LOAD (a) T1
LOAD (b) T2
MUL T1 T2 T3
LOAD (c) T4
SUB T3 T4 T3

c) LOAD (b) T1
LOAD (c) T2
SUB T1 T2 T3
LOAD (a) T4
MUL T3 T4 T3

3)

(a)



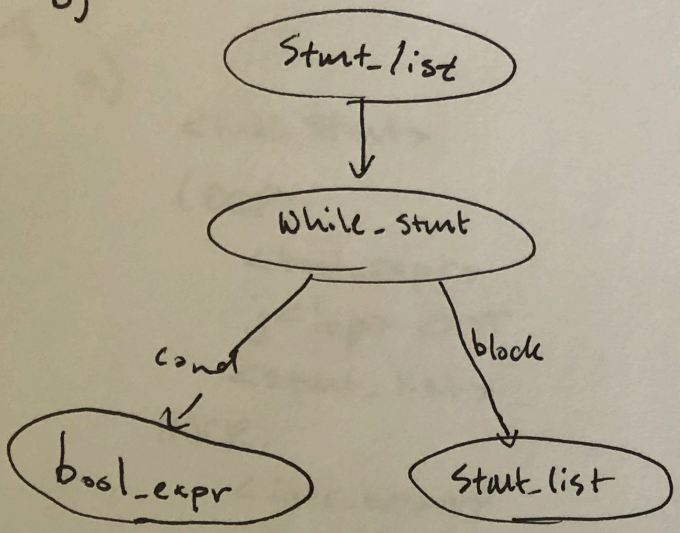
<code for bool_expr_1>
← j<!op> ELSE_1
← <code for stmt_list_1>
jmp OUT_1

ELSE_1:
<code for bool_expr_2>
j<!op> ELSE_2
<code for stmt_list_2>
jmp OUT_1

ELSE_2:
<code for stmt_list_3>

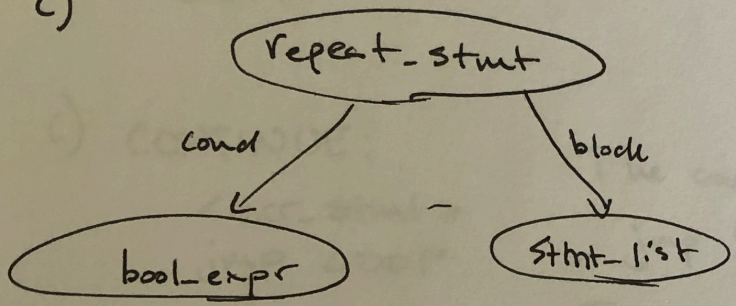
OUT1:

b)




```
< Stmt_list >
LOOP:
  <bool_expr>
  j<!op> OUT
  <stmt_list>
  jmp LOOP
OUT:
```

c)




```
LOOP:
  <stmt_list>
  <bool_expr>
  j<!op> OUT
  jmp LOOP
OUT:
```

4

4

a)

```
<init_stmt>
LOOP:
    <bool_expr>
    j<!op> OUT
    <stmt_list>
INCR:
    <incr_stmt>
    jmp LOOP
OUT:
```

b)

```
<init_stmt>
LOOP:
  <bool_expr>
  j<!op> OUT
  <stmt_list>
  <incr_stmt>
  jmp LOOP
OUT:
```

c)

```
CONTINUE:
    <incr_stmt>
    jmp LOOP

BREAK:
    jmp OUT
```

The code for continue would be
jmp CONTINUE.

The code for break could be
jmp BREAK
it also could be
jmp OUT

Many times continue & break statements happen after boolean expressions, in these cases the code would be

```
<cont_bool_expr>
j<!op> CONTINUE

or

<break_bool_expr>
j<!op> BREAK || jmp OUT
```

5)

```
<init_stmt>
LOOP:
    <bool_expr>
    j<!op> OUT
    <stmt_list>
    jmp INCR
SKIP_2:
    <incr_stmt>
    <bool_expr>
    j<!op> OUT
    <incr_stmt>
    <bool_expr>
    j<!op> OUT
INCR:
    <incr_stmt>
    jmp LOOP
OUT:
```

jmp SKIP_2 ← ~~SKIP_2~~ would exist within the stmt_list, to make this implementation work, skipping 2 iterations and a third with our general INCR statement.