Q1

1.

```
E → { E1.false := E.true; E1.true := newlabel } E1 ⊼
{ E2.false := E.true; E2.true := E.false } E2
{ E.code := E1.code || gen(E1.true ':') || E2.code }
```

2.

```
S → repeat { S1.next := newlabel } S1 until
{ E.true = S.next; E.false = newlabel } E
{ S.code := gen(E.false ':') || S1.code || gen(S.next ':') || E.code }
```

Q2

`E.truelist / E.falselist`： 表达式为真 / 为假时的未定跳转链表
`S.nextlist`： 从语句跳出的未定跳转链表
`M.instr`： 记住当前 `nextinstr`
语义函数： `makelist`、`merge`、`backpatch` 与讲义一致
空产生式： `M → ε { M.instr := nextinstr }`

1.

在 S-翻译中应写成

```
E → E1 ⊼ M E2
    { backpatch(E1.truelist, M.instr);
      E.truelist := merge(E1.falselist, E2.falselist);
      E.falselist := E2.truelist; }
```

2.

同样插入两个标记 `M1`、`M2` ，表示循环体开始和条件开始处的指令序号：

```
S → repeat M1 S1 until M2 E
    { backpatch(S1.nextlist, M2.instr);   /  循环体顺接到条件  /
      backpatch(E.falselist, M1.instr);   /  条件假则回到循环首  /
      S.nextlist := E.truelist; }         /  条件真则跳出 repeat 语句  /
```

其中两次出现的 `M1`、`M2` 都是非终结符 `M` 的不同实例，均有：

```
M → ε { M.instr := nextinstr }
```

这样的话就给出了在拉链与代码回填 S-翻译模式下，对新增产生式的语义动作集合。

Q3

1.

```
A → A1 + A2 { A.instr := A2.instr || A1.instr || Plus }
```

```
A → A1 – A2 { A.instr := A2.instr || A1.instr || Minus }
```

2.

答案如下所示，其中
A1 > A2 等价于 (A1 − A2 ≥ 0) if (A1 − A2 ≠ 0)；
B1 ∧ B2 等价于 B2 if B1；
¬B1 等价于 1 if (B1 − 1 ≠ 0)。

```
E → E1 if B
    { E.instr := E1.instr || B.instr || Cond }

B → A1 > A2
    { B.instr := A2.instr || A1.instr || Minus || Cmp
              || A2.instr || A1.instr || Minus || Cond }

B → B1 ∧ B2
    { B.instr := B2.instr || B1.instr || Cond }

B → ¬B1
    { B.instr := Push 1 || Push 1 || B1.instr || Minus || Cond }
```