Klasse *Label*


```
class Label {
    Code code;
        List<Integer> fixupList;     // code positions to patch
        int adr;                     // address of label in code


        // inserts offset to label at current pc
        void put ();

        // defines label to be at the current pc position
        void here ();
}
```

Klasse *Operand* – Erweiterung für Sprünge


```
class Operand {
   public enum Kind {
      Con, Local, Static, Stack, Fld, Elem, Meth, Cond
   }

      public Kind kind;
      public Struct type;
      public int val;          // Con: value
      public int adr;          // Local, Static, Fld, Meth: address
      public Obj obj;          // Meth: method object from the
symbol table

      public CompOp op;   // Cond: comparison operator
      public Label tLabel; // Cond: target for true jumps
      public Label fLabel; // Cond: target for false jumps
}
```

Klasse *Code* – Erweiterung für Sprünge


```
class Code {
        ...

        // generates unconditional jump instruction to lab
        void jump (Label lab);

        // generates conditional jump instruction for true jump
        // x represents the condition
        void tJump (Operand x);

        // generates conditional jump instruction for false jump
        // x represents the condition
        void fJump (Operand x);
}
```

Klasse *Label* - Methode *put*


```
// inserts offset to label at current pc
void put () {
        if (isDefined()) {
                code.put2(adr - (code.pc - 1));
        }
        else {
                fixupList.add(code.pc);
                // insert place holder
                code.put2(0);
        }
}
```

Klasse *Label* - Methode *here*


```
// defines label to be at current pc
void here () {
        if (isDefined()) {
                // should never happen
                throw new IllegalStateException("label defined
twice");
        }

        for (int pos : fixupList) {
                code.put2(pos, code.pc - (pos - 1));
        }

        fixupList = null;
        adr = code.pc;
}
```

Semantische Aktionen


```
Item CondTerm () {
        Item x = CondFact();
        while (sym == and) {
                code.fJump(x);
                scan();
                Item y = CondFact();
                x.op = y.op;
        }
        return x;
}
```

```
Item Condition () {
        Item x = CondTerm();
        while (sym == or) {
                code.tJump(x);
                scan();
                x.fLabel.here();
                Item y = CondTerm();
                x.fLabel = y.fLabel;
                x.op = y.op;
        }
        return x;
}
```

Ausschnitt aus **Statement** ()

```
        case if_:
                ...
                Item x = Condition();
                code.fJump(x);
                x.tLabel.here();
                ...
```

Semantische Aktionen

Ausschnitt aus **Statement** ()

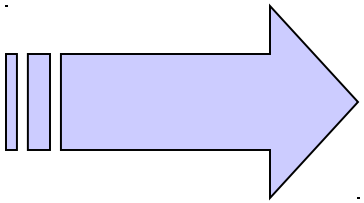
```
case while_:
        scan();
        check(lpar);
        Label top = new Label(code);
        top.here();
        Item x = Condition();
        code.fJump(x);
        x.tLabel.here();
        check(rpar);
        Statement();
        code.jump(top);
        x.fLabel.here();
```

Für die Codeerzeugung von "break" braucht Statement ein Label als Parameter

Beispiel: Methoden & Methodenaufrufe

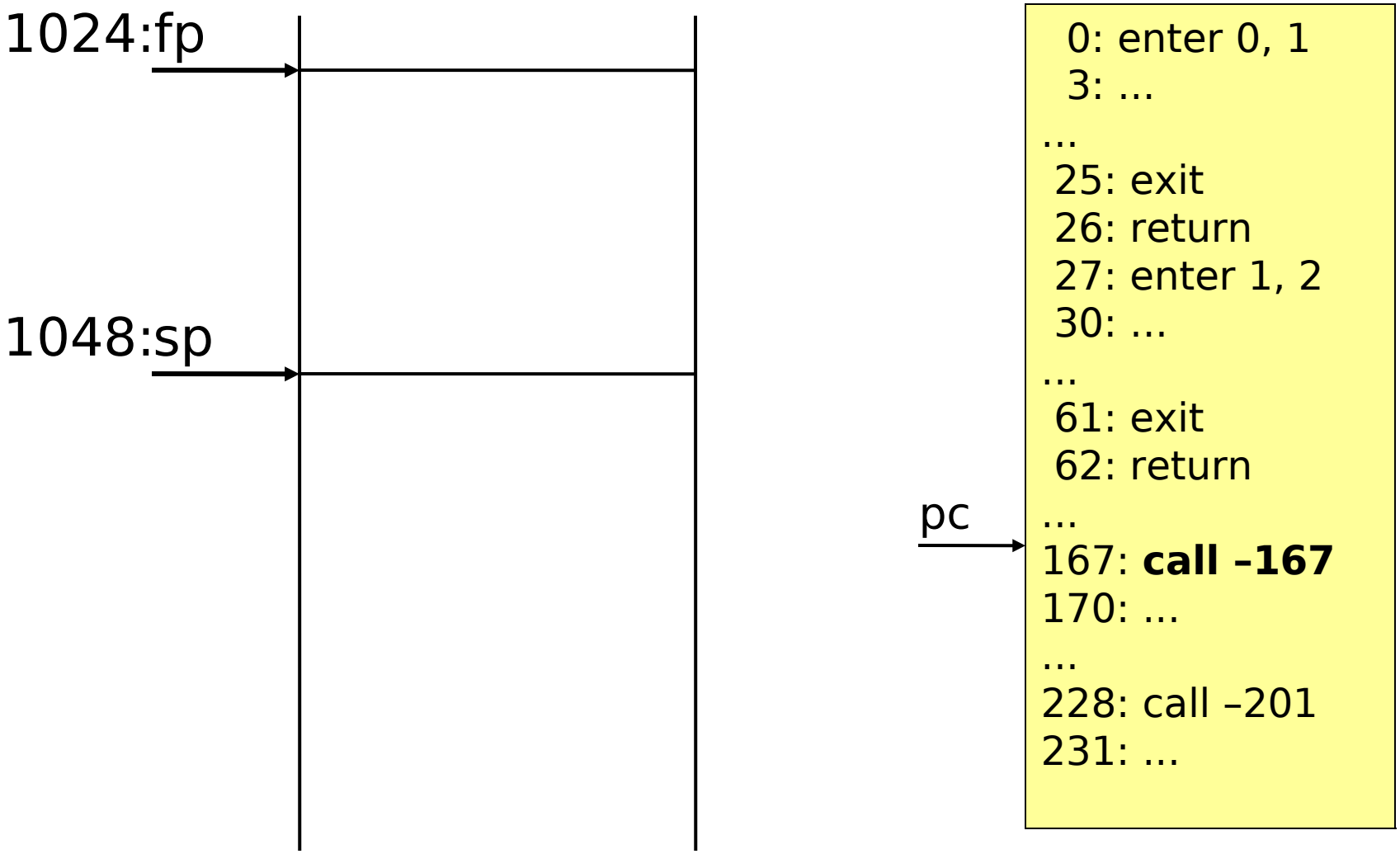
```
void m1 ()
        char c;
{...}

void m2 (int i)
        int j;
{...}
...
void main () ... {
        m1();
        ...
        m2(1);
}
...
```

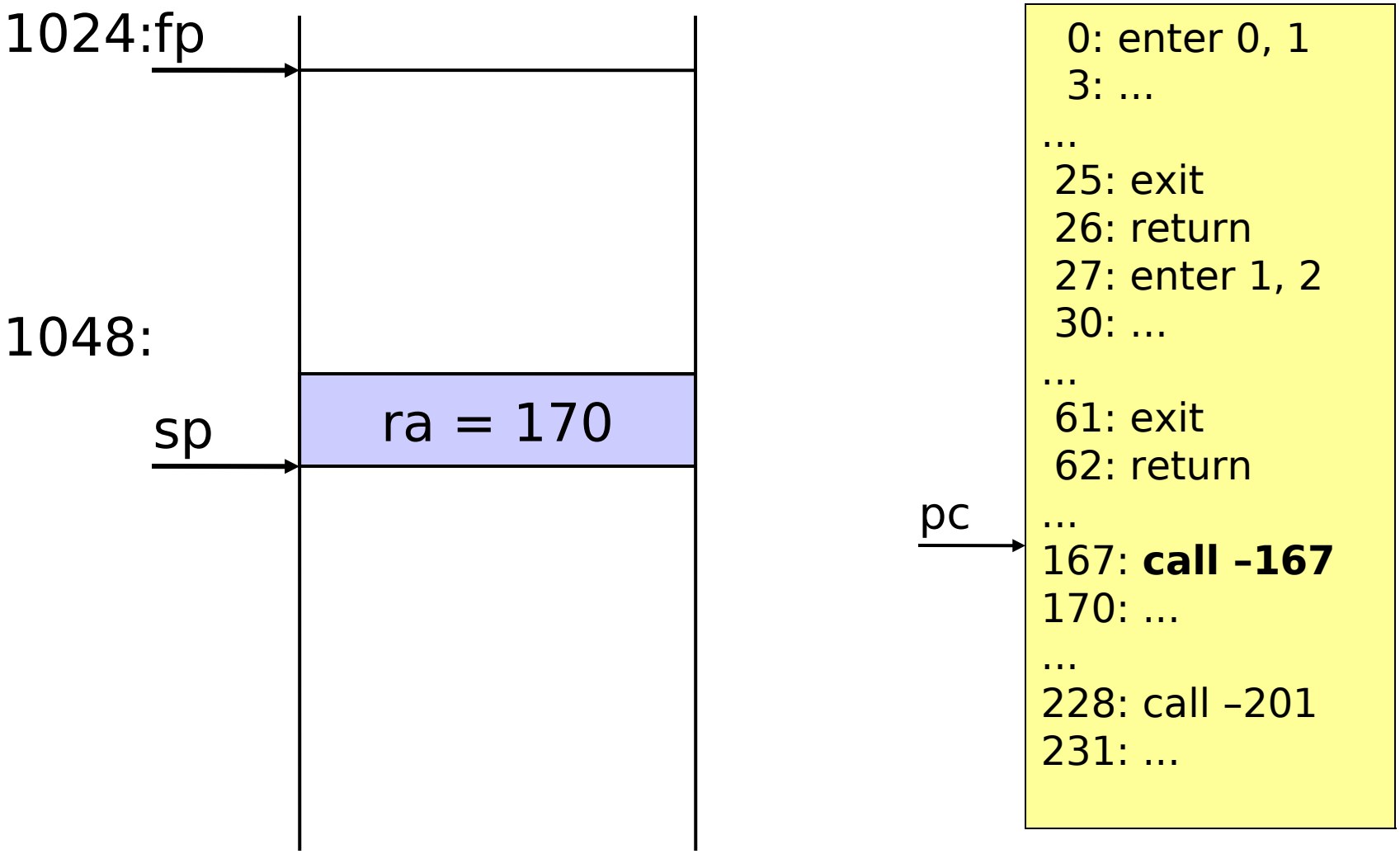


```
 0: enter 0, 1
 3: ...
 ...
25: exit
26: return
27: enter 1, 2
30: ...
 ...
61: exit
62: return
 ...
167: call –167
170: ...
 ...
228: call –201
231: ...
```

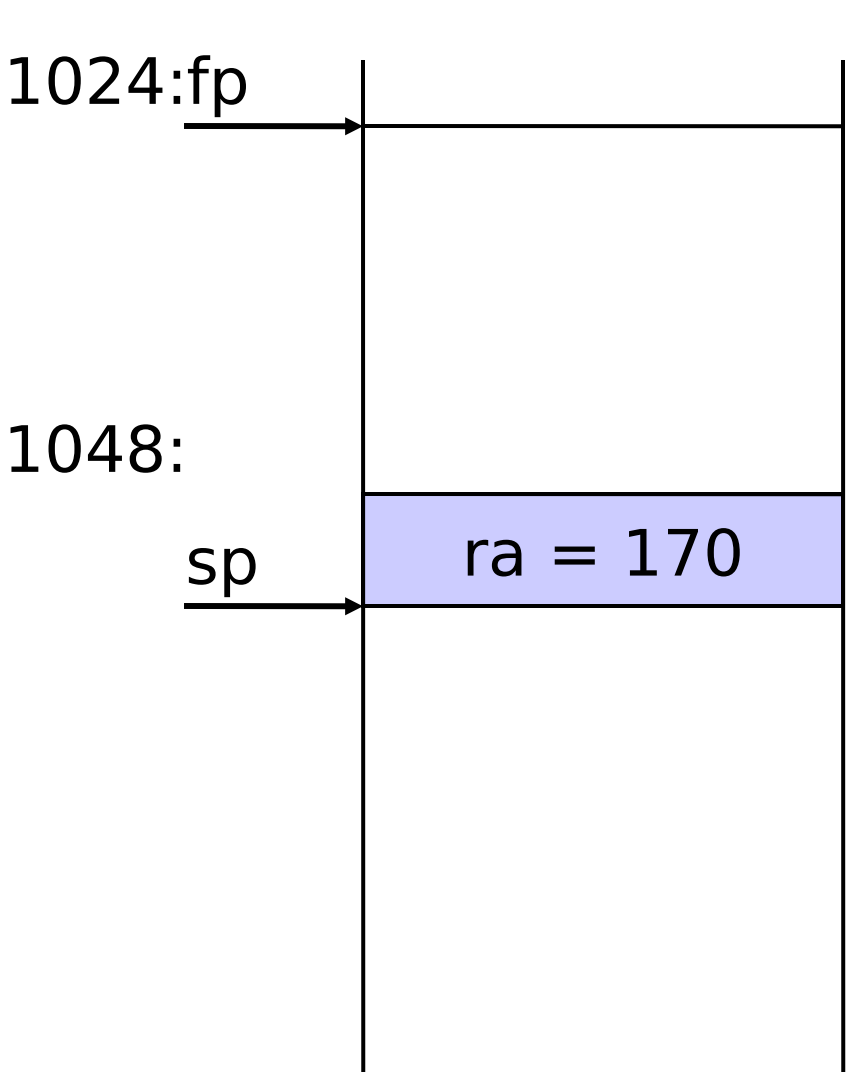
Methodenaufruf m1



Methodenaufruf m1



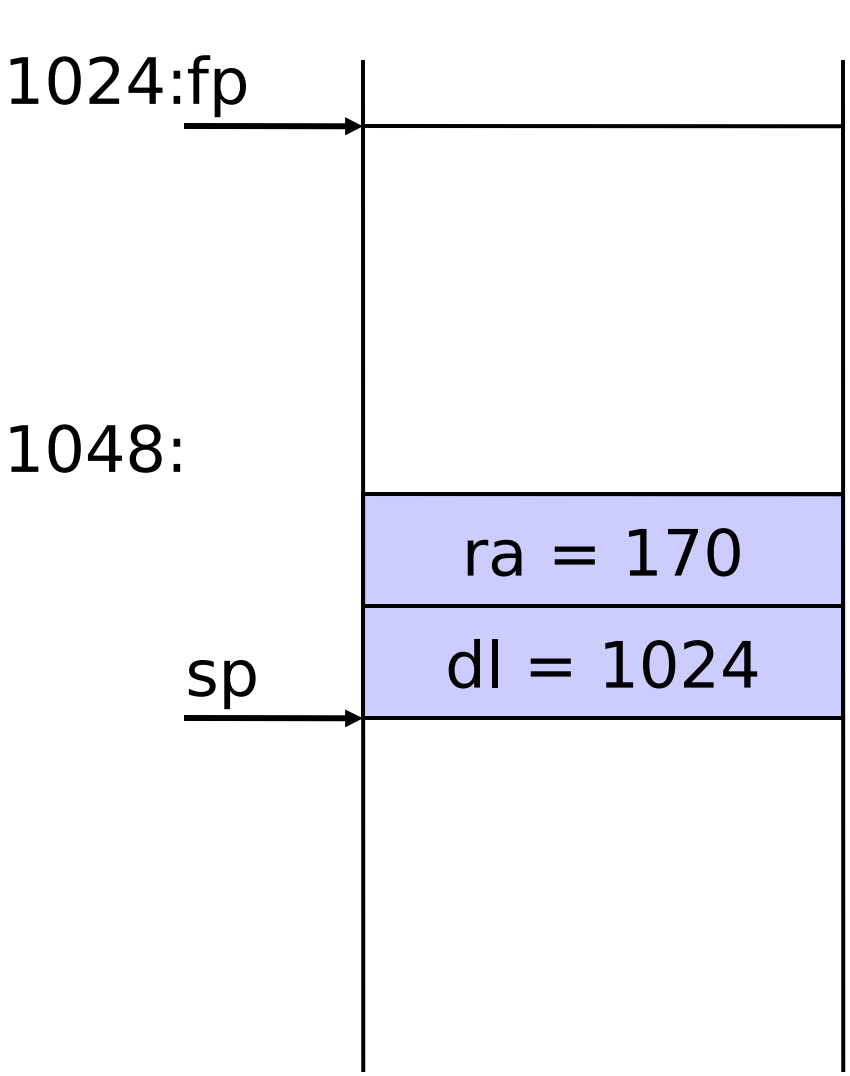
Einsprung in Methode m1



pc

```
 0: enter 0, 1
 3: ...
...
25: exit
26: return
27: enter 1, 2
30: ...
...
61: exit
62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

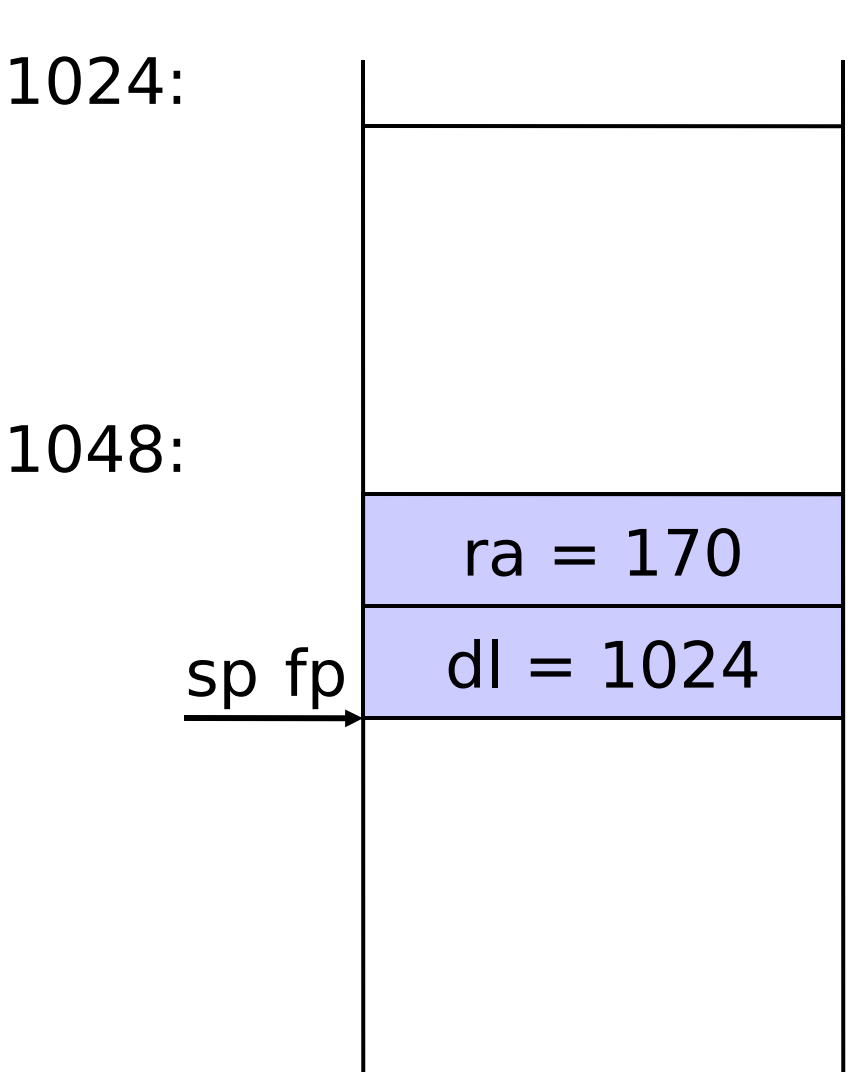
Einsprung in Methode m1



pc

```
 0: enter 0, 1
 3: ...
...
25: exit
26: return
27: enter 1, 2
30: ...
...
61: exit
62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

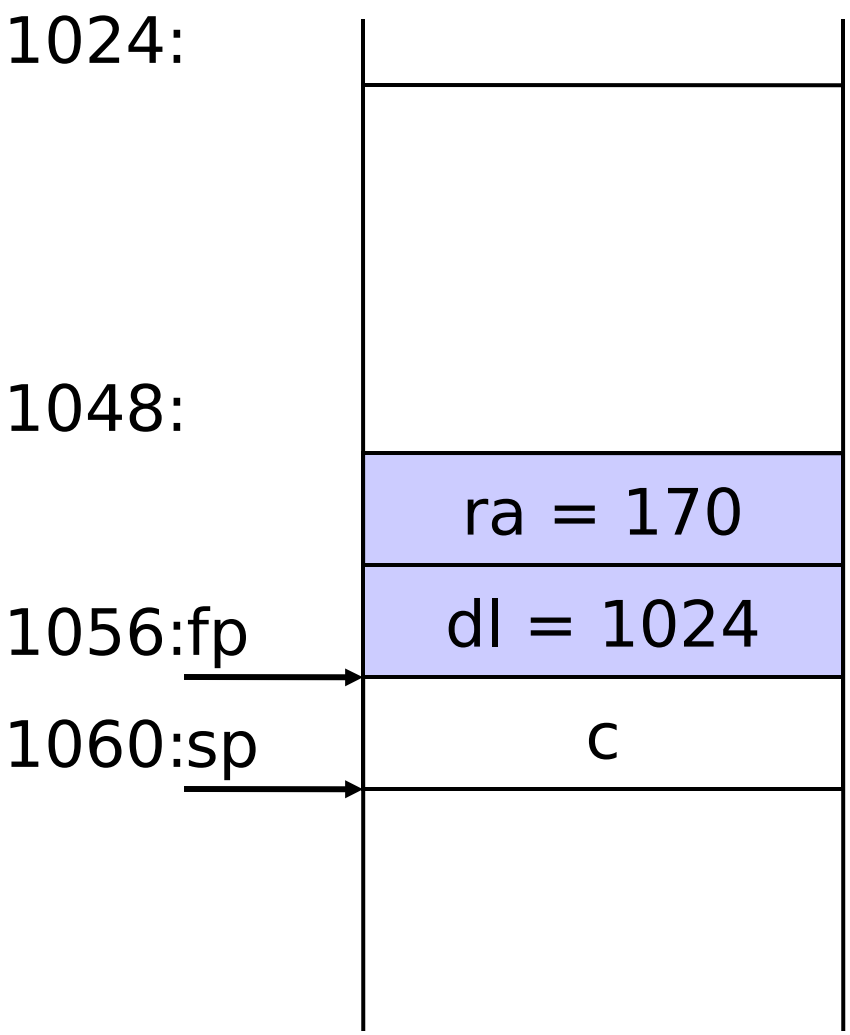
Einsprung in Methode m1



pc

```
 0: enter 0, 1
 3: ...
...
25: exit
26: return
27: enter 1, 2
30: ...
...
61: exit
62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

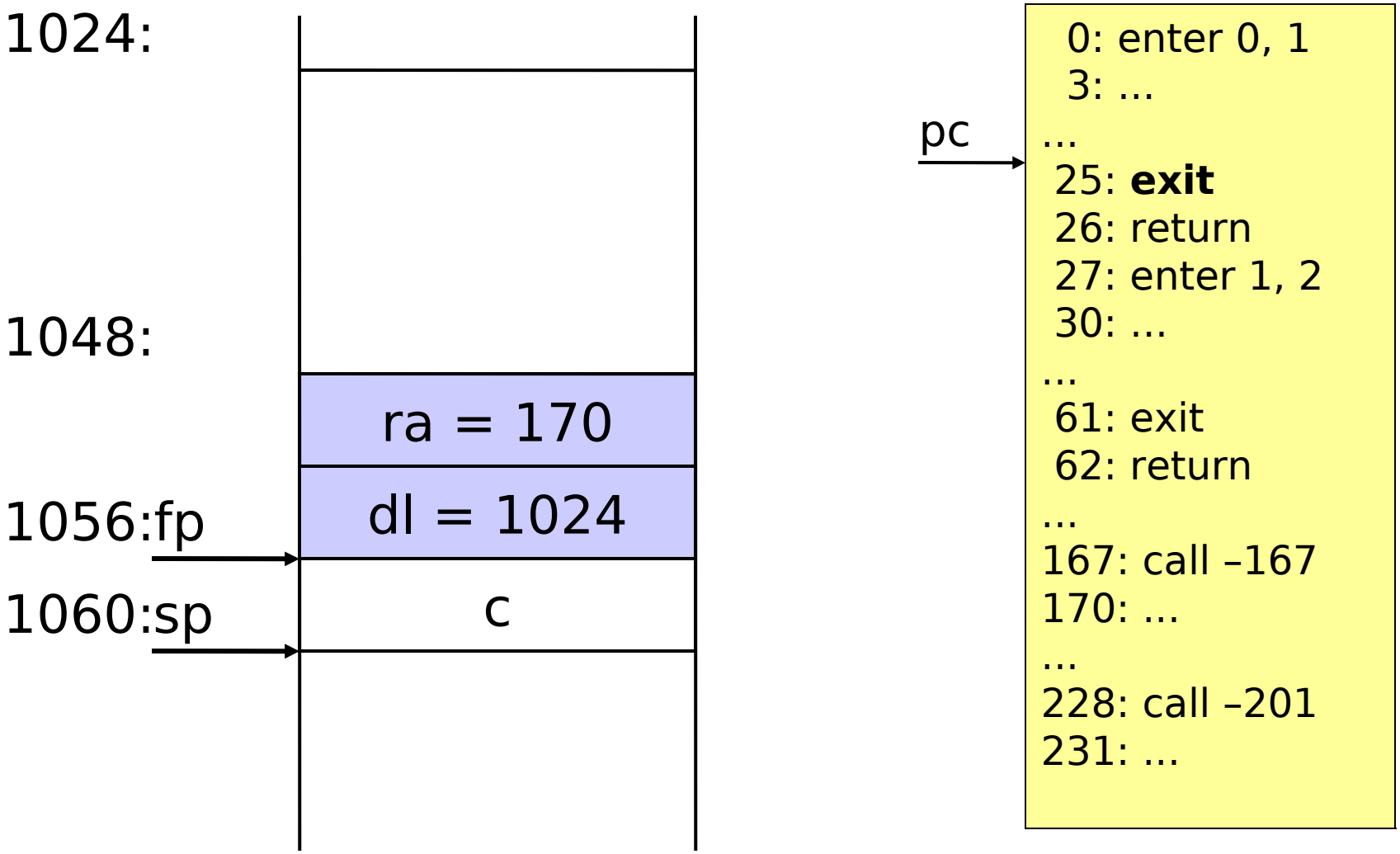
Einsprung in Methode m1



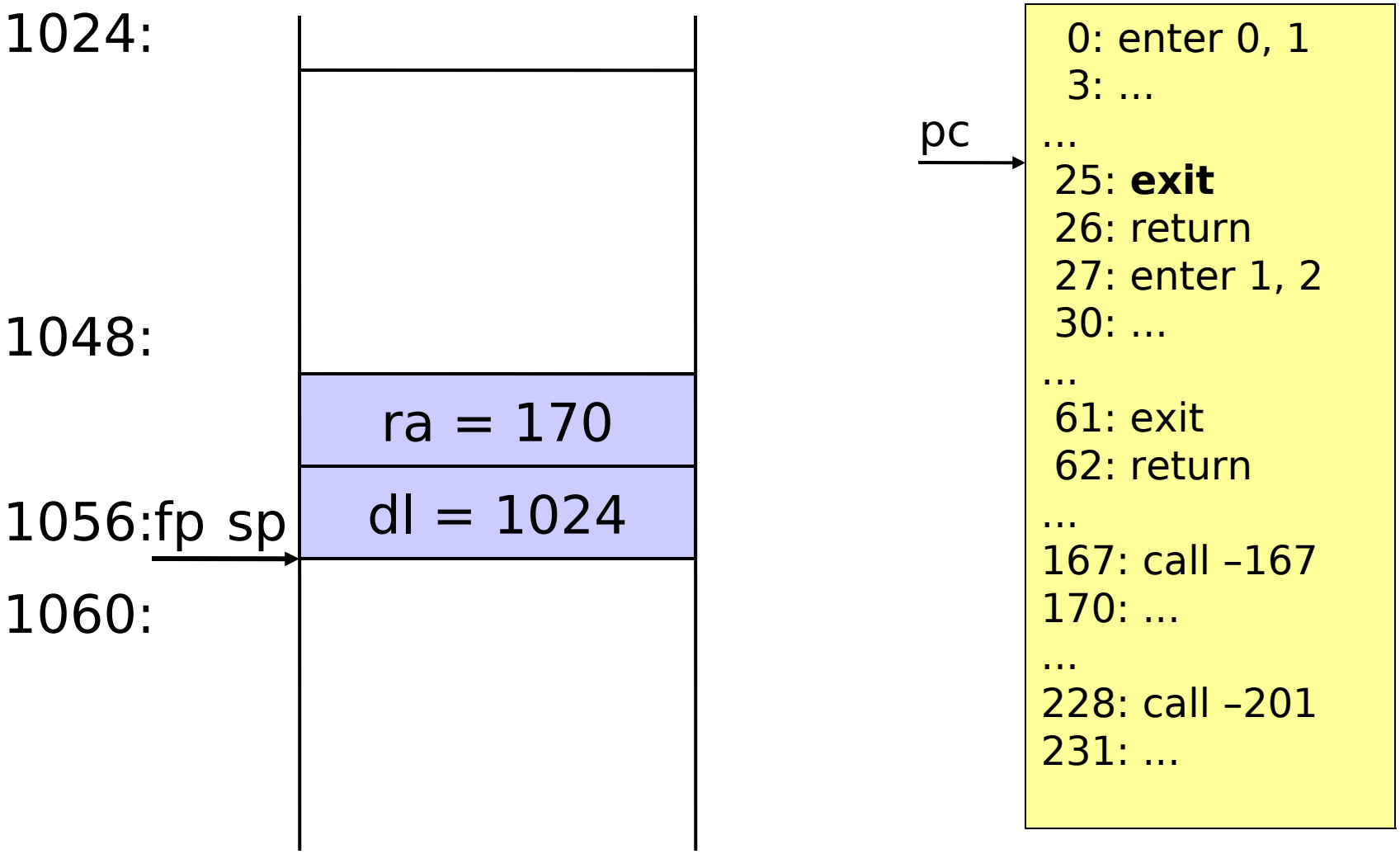
pc →

```
 0: enter 0, 1
 3: ...
...
25: exit
26: return
27: enter 1, 2
30: ...
...
61: exit
62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

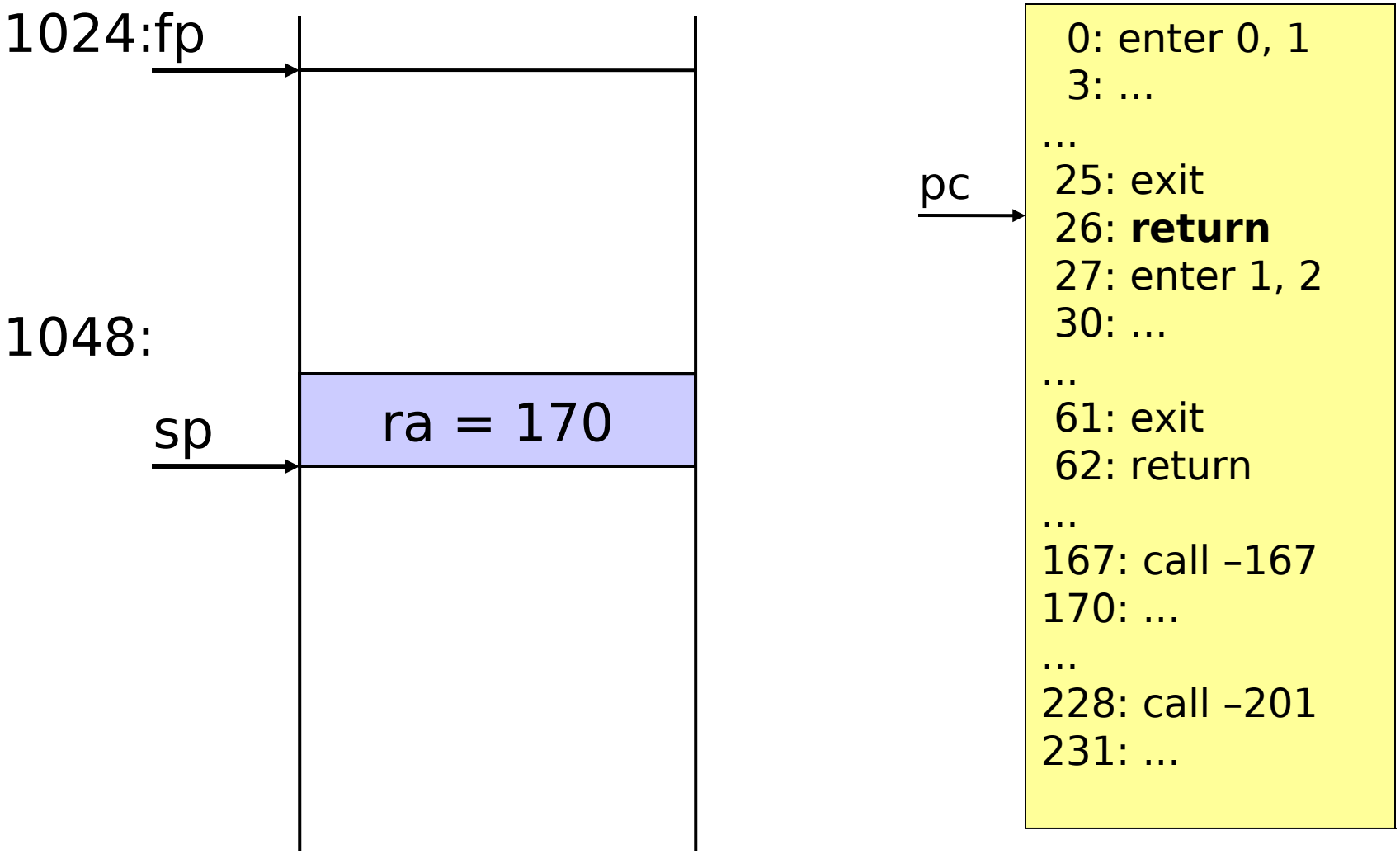
Ende der Methode m1



Ende der Methode m1



Rücksprung zum Rufer der Methode m1



Rücksprung zum Rufer der Methode m1