

# Klasse *Label*



```
class Label {  
    boolean defined;    // true if label has been defined  
    int adr;            // if (defined) adr == position of label in code  
                        // else adr == position of prev. fixup label  
  
    // generates code for a jump to this label  
    void put ();  
  
    // defines label to be at the current pc position  
    void here ();  
}
```

# Klasse *Item* - Erweiterung für Sprünge



```
class Item {
    static final int          // item kinds
        Con=0, Local=1, Static=2, Stack=3, Fld=4, Elem=5, Meth=6,
        Cond = 7;

    int kind;
    Struct type;          // Typ des Operanden
    int val;              // Con: Wert
    int adr;              // Local, Static, Fld, Meth: Adresse
    Obj obj;              // Meth: Methodenobjekt aus Symbolliste
    int op;               // Cond: relational Operator (eq=0,ne=1,...)

    Label tLabel,         // Cond: destination of true jump
          fLabel;        // Cond: destination of false jump
}
```

# Klasse *Code* - neue Methoden 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 (Item x);  
  
    // generates conditional jump instruction for false jump  
    // x represents the condition  
    void fJump (Item x);  
}
```



# Klasse *Label* - Methode *put*

*// inserts offset to label at current pc*

```
void put () {  
    int pc = codeBuf.pc;  
  
    if (defined) {  
        codeBuf.put2(adr - (pc-1));  
    } else {  
        codeBuf.put2(adr);  
        adr = pc;  
    }  
}
```

# Klasse *Label* - Methode *here*



*// defines label to be at current pc*

```
void here () {  
    if (defined) throw new Error(Messages.getString("LAB_DEF"));  
  
    while (adr != 0) {  
        int pos = adr;  
        adr = codeBuf.get2(adr);  
        codeBuf.put2(pos, codeBuf.pc - (pos-1));  
    }  
    defined = true;  
    adr = codeBuf.pc;  
}
```

# Semantische Aktionen



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

```
Ausschnitt aus Statement ()  
case Token.if_:  
    [...]  
    Item x = Condition();  
    code.fJump(x);  
    x.tLabel.here();  
    [...]
```

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

# Semantische Aktionen

Ausschnitt aus **Statement** ()

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

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

# Beispiel: Methoden & Methodenaufrufe

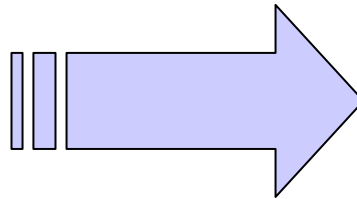


```
void m1 ()  
    char c;  
{...}
```

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

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

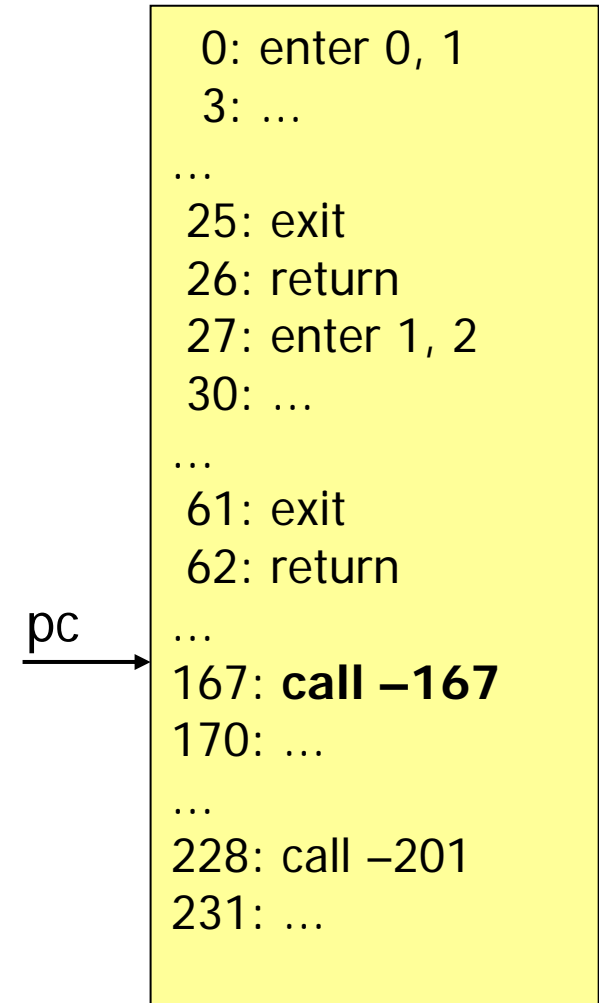
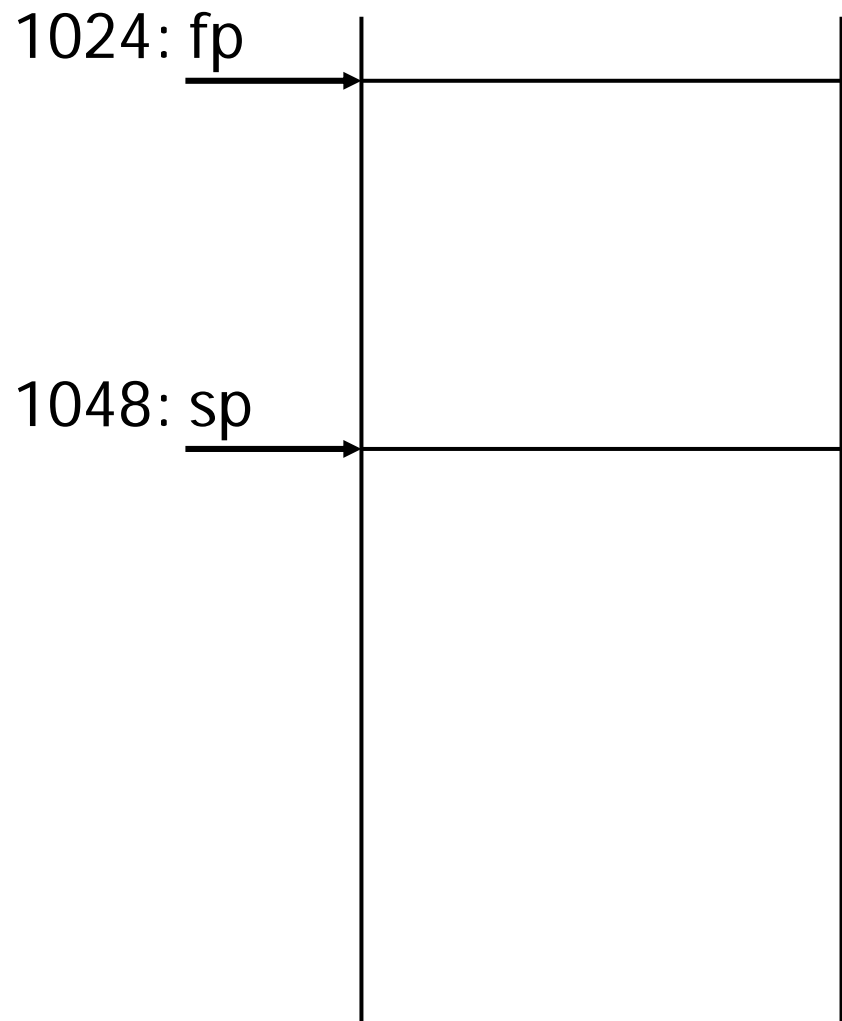
...



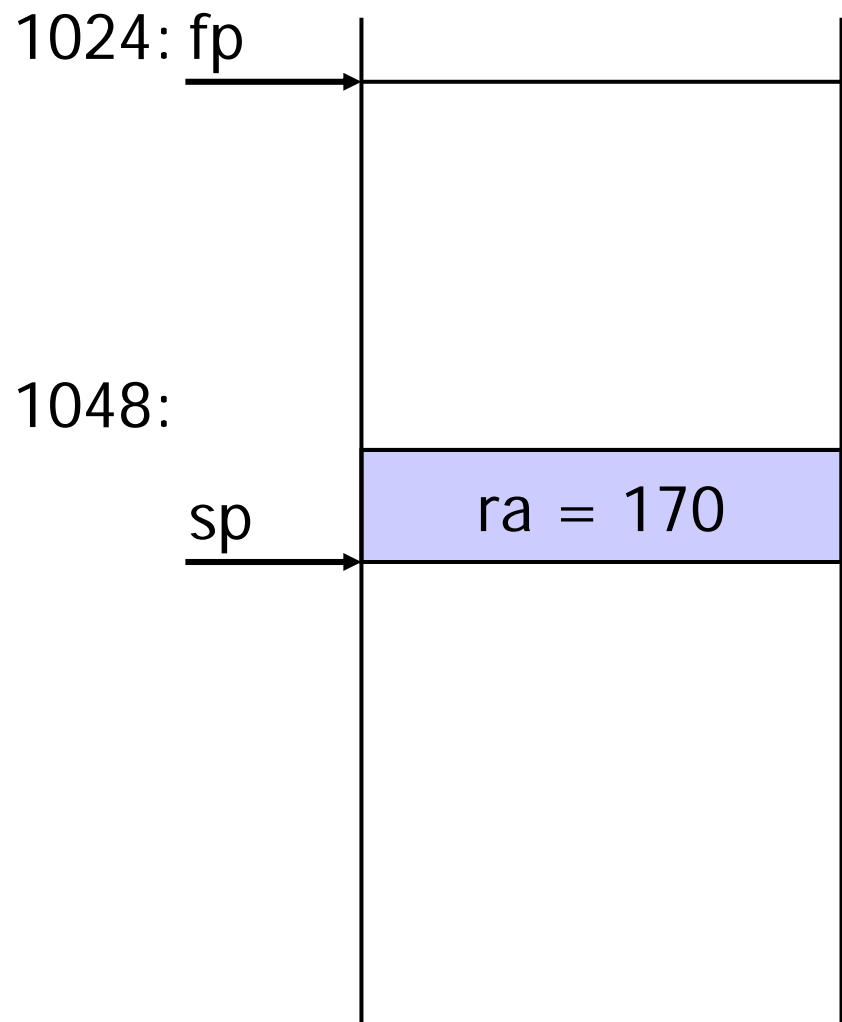
```
0: enter 0, 1  
3: ...  
...  
25: exit  
26: return  
27: enter 1, 2  
30: ...  
...  
61: exit  
62: return  
...  
167: call -167  
170: ...  
...  
228: call -201  
231: ...
```



# Methodenaufruf m1



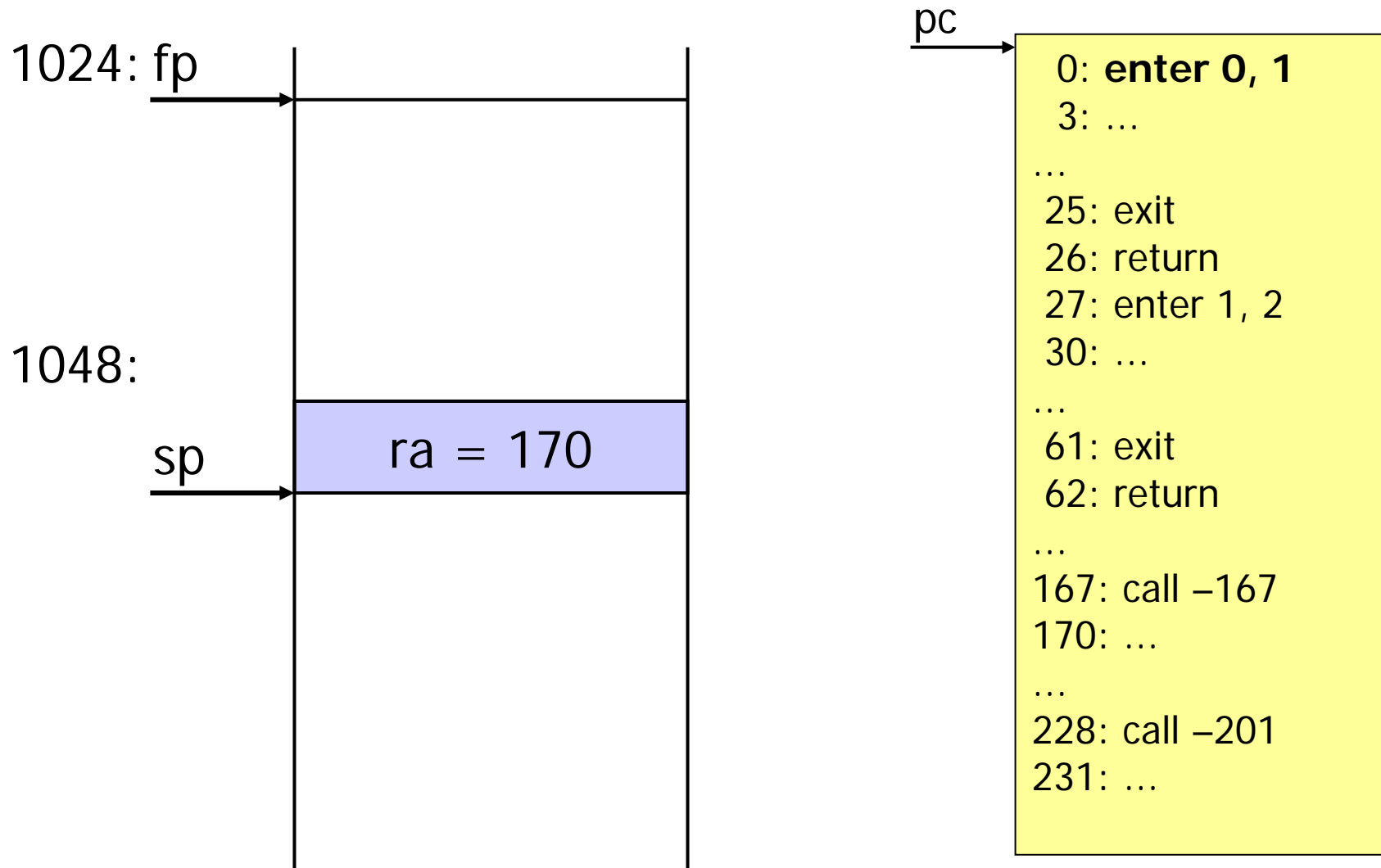
# Methodenaufruf m1



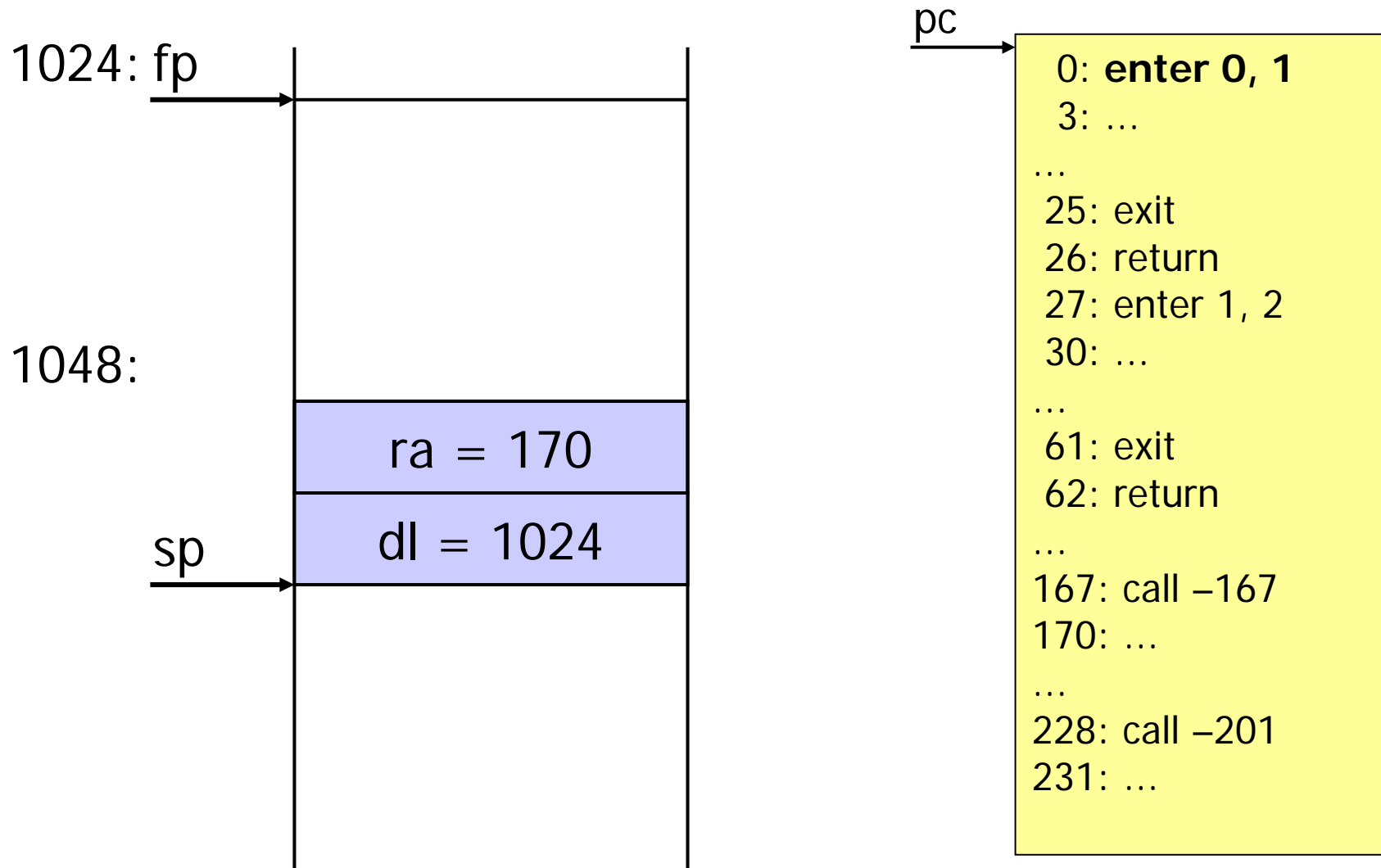
```
0: enter 0, 1
3: ...
...
25: exit
26: return
27: enter 1, 2
30: ...
...
61: exit
62: return
...
167: call -167
170: ...
...
228: call -201
231: ...
```

pc →

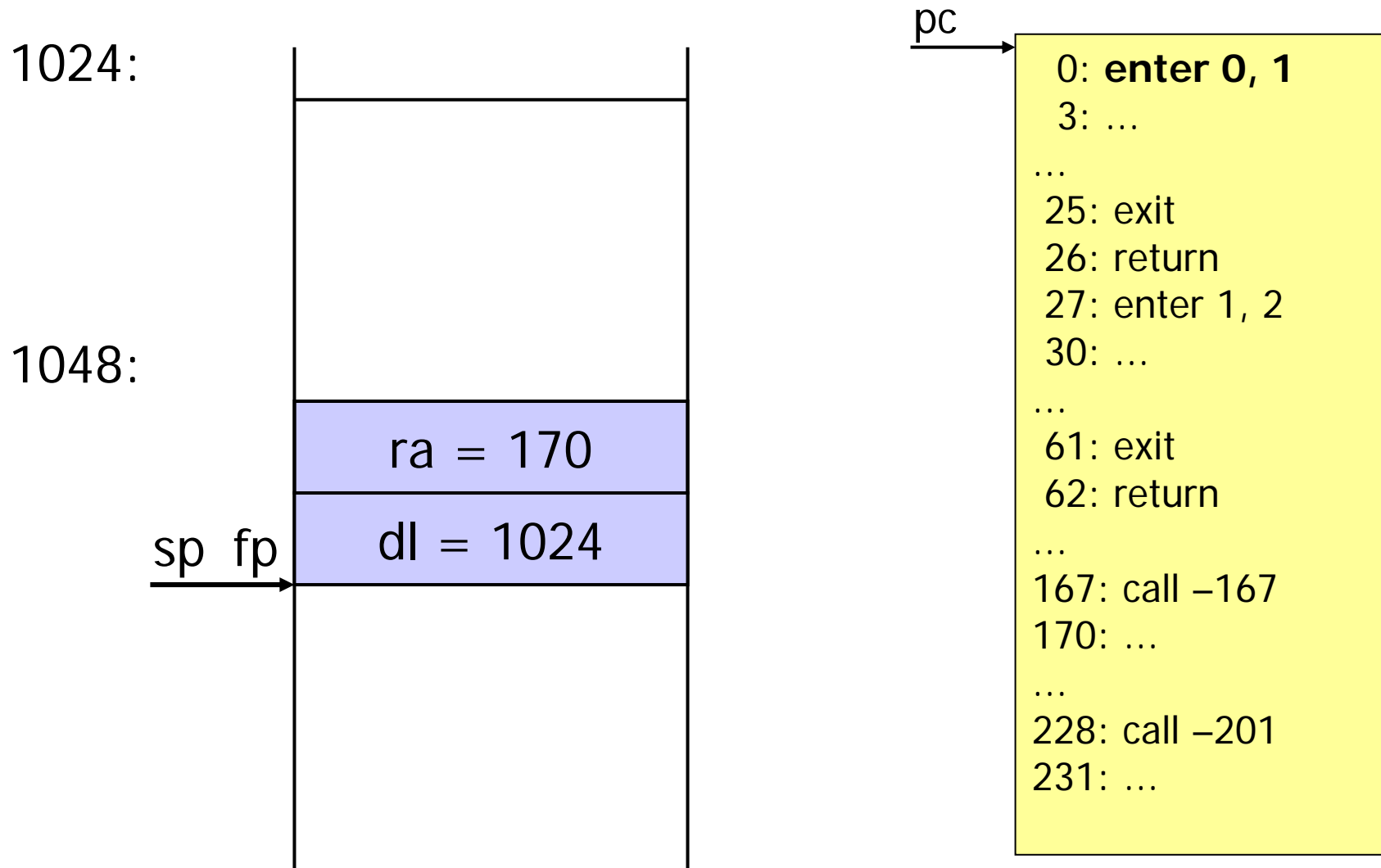
# Einsprung in Methode m1



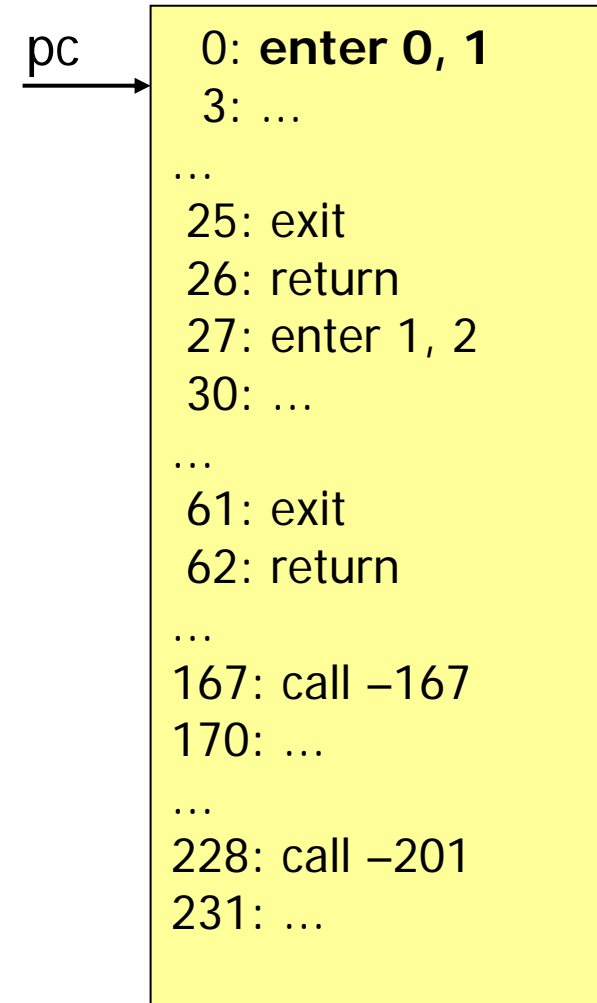
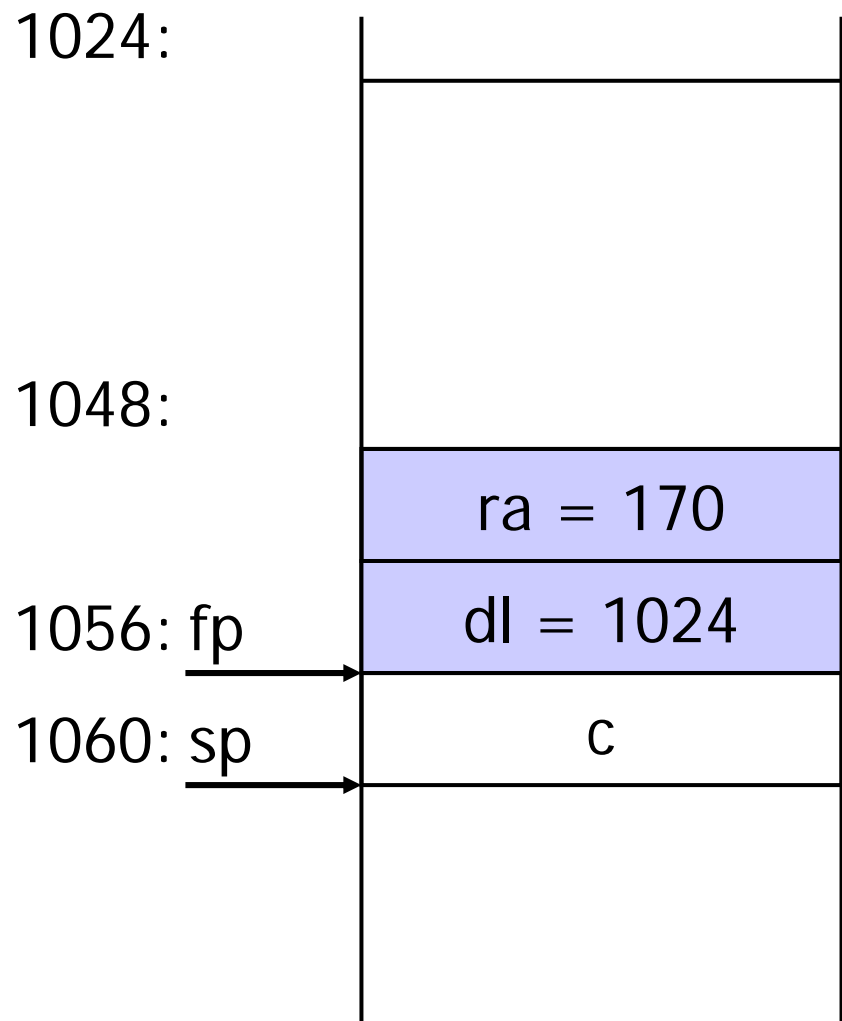
# Einsprung in Methode m1



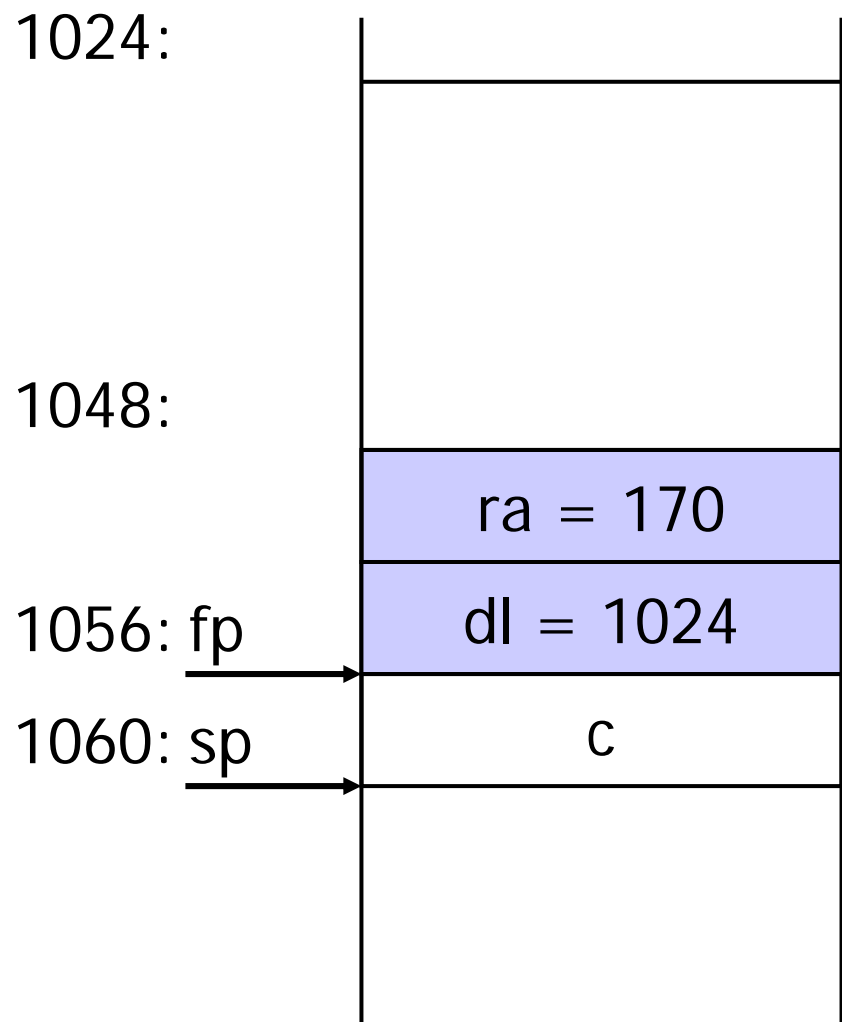
# Einsprung in Methode m1



# Einsprung in Methode m1



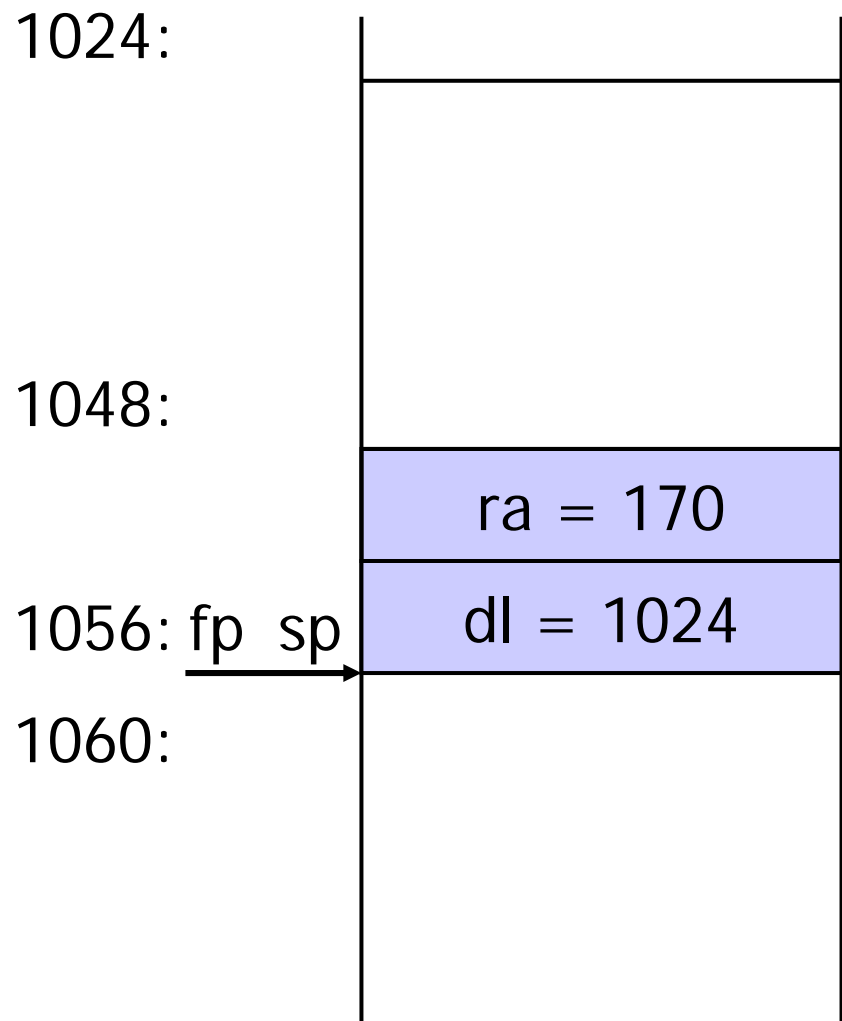
# Ende der Methode m1



```
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: ...
```

pc →

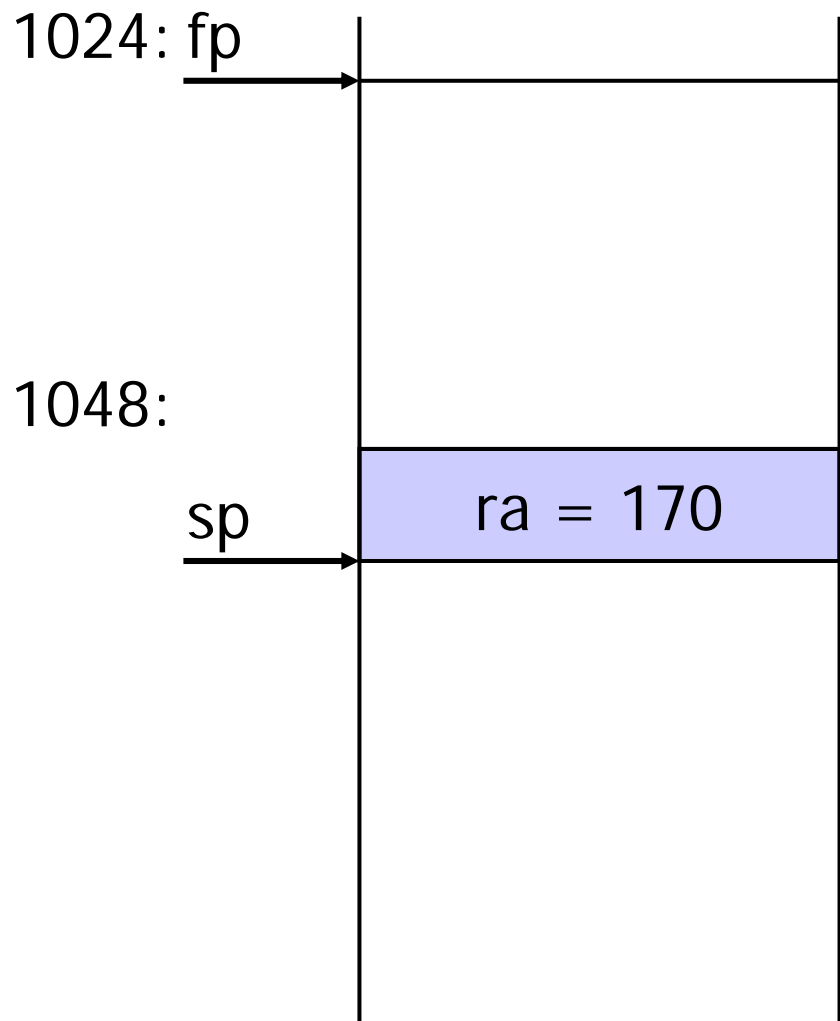
# Ende der Methode m1



```
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: ...
```

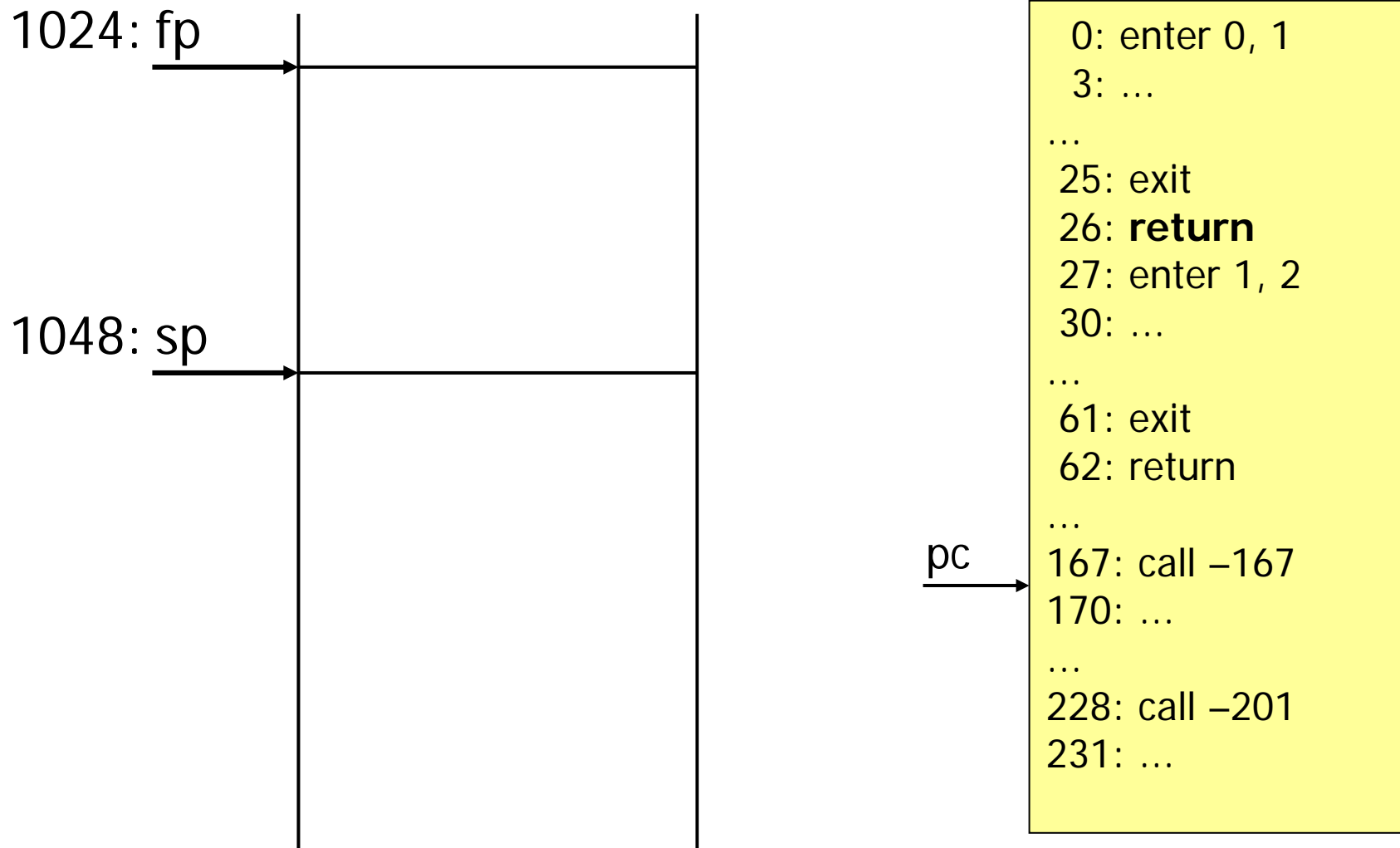


# Rücksprung zum Rufer der Methode m1



```
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: ...
```

# Rücksprung zum Rufer der Methode m1



Bsp 11: **if (i <= n) n=0;**

*Deklaration:* **class A**

```
    final int max = 12;           // Konstante
    char c; int i;                // globale Variablen
    class B { int x, y; }         // innere Klasse mit Feldern
{ void foo () int[] iarr; B b; int n; {...} }
```

10: getstatic 1

13: load\_2

14: **jgt 5** (--> 19)

17: const\_0

18: store\_2

19: ...

Bsp 12: `if (i <= n && n < 0) n=0;`

*Deklaration:* `class A`

```
    final int max = 12;           // Konstante
    char c; int i;                // globale Variablen
    class B { int x, y; }         // innere Klasse mit Feldern
{ void foo () int[] iarr; B b; int n; {...} }
```

```
10:  getstatic 1
13:  load_2
14:  jgt 10          (--> 24)
17:  load_2
18:  const_0
19:  jge 5           (--> 24)
22:  const_0
23:  store_2
24:  ...
```

Bsp 13: `if (i <= n || n < 0) n=0;`

*Deklaration:* `class A`

```
    final int max = 12;           // Konstante
    char c; int i;                // globale Variablen
    class B { int x, y; }         // innere Klasse mit Feldern
{ void foo () int[] iarr; B b; int n; {...} }
```

```
10:  getstatic 1
13:  load_2
14:  jle 8           (--> 22)
17:  load_2
18:  const_0
19:  jge 5           (--> 24)
22:  const_0
23:  store_2
24:  ...
```

Bsp 14: **if (i<=n || n<0 && i>0) n=0;**

*Deklaration:* **class A**

```
    final int max = 12;      // Konstante  
    char c; int i;          // globale Variablen  
    class B { int x, y; }    // innere Klasse mit Feldern
```

```
{ void foo () int[] iarr; B b; int n; {...} }
```

10: getstatic 1

13: load\_2

14: jle 15 (--> 29)

17: load\_2

18: const\_0

19: jge 12 (--> 31)

22: getstatic 1

25: const\_0

26: jle 5 (--> 31)

29: const\_0

30: store\_2

31: ...

Bsp 15: **while (i<=n) n++;**

*Deklaration:* **class A**

```
    final int max = 12;           // Konstante
    char c; int i;                // globale Variablen
    class B { int x, y; }         // innere Klasse mit Feldern
{ void foo () int[] iarr; B b; int n; {...} }
```

**10:** getstatic 1

**13:** load\_2

**14:** jgt 9 (--> 23)

**17:** inc 2 1

**20:** jmp -10 (--> 10)

**23:** ...

Bsp 16: **if (i <= n) n=0 else n=1;**

*Deklaration:* **class A**

```
    final int max = 12;           // Konstante
    char c; int i;                // globale Variablen
    class B { int x, y; }         // innere Klasse mit Feldern
{ void foo () int[] iarr; B b; int n; {...} }
```

```
10:  getstatic 1
13:  load_2
14:  jgt 8      (--> 22)
17:  const_0
18:  store_2
19:  jmp 5      (--> 24)
22:  const_1
23:  store_2
24:  ...
```