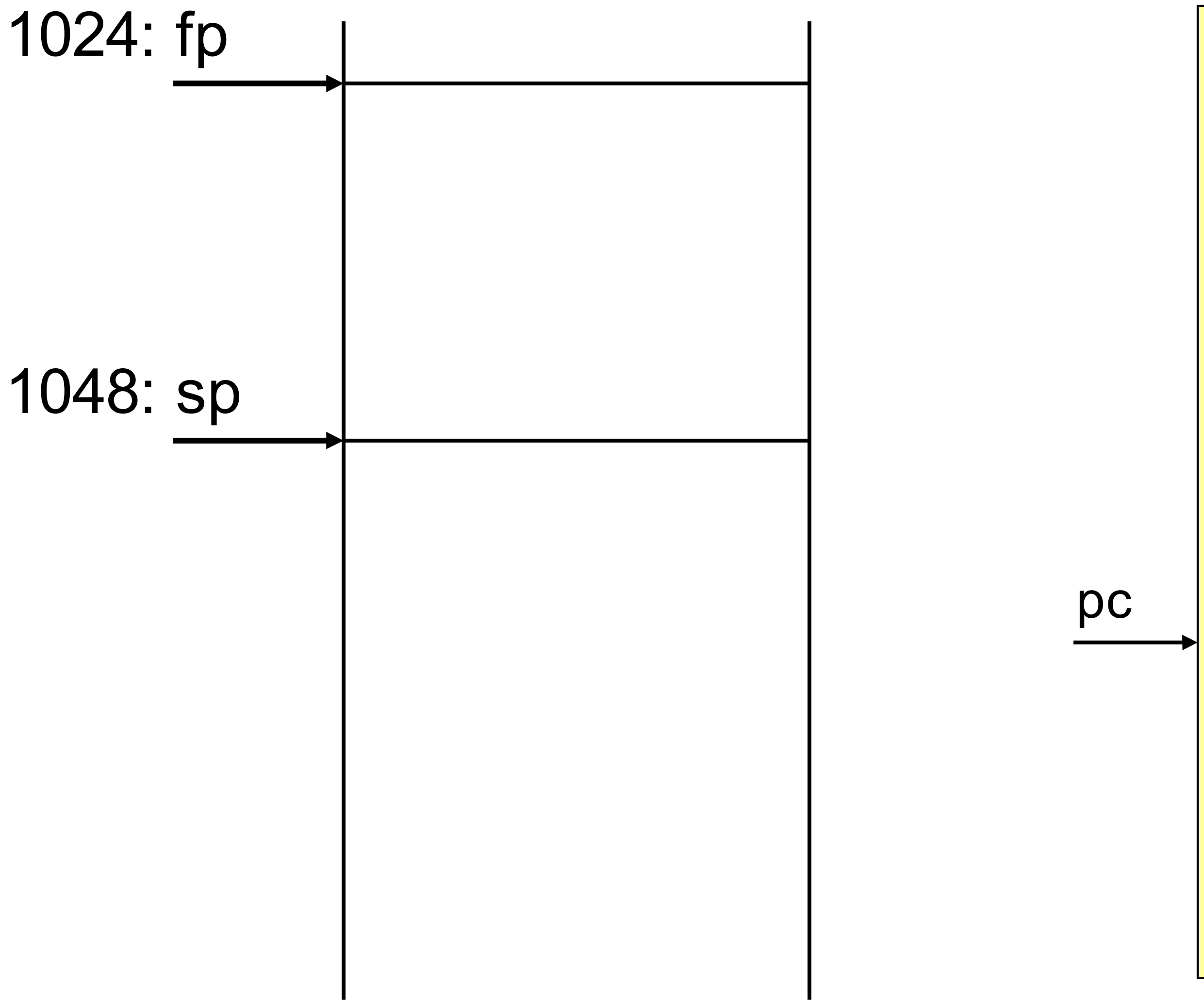
Beispiel: Methoden & Methodenaufrufe

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

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

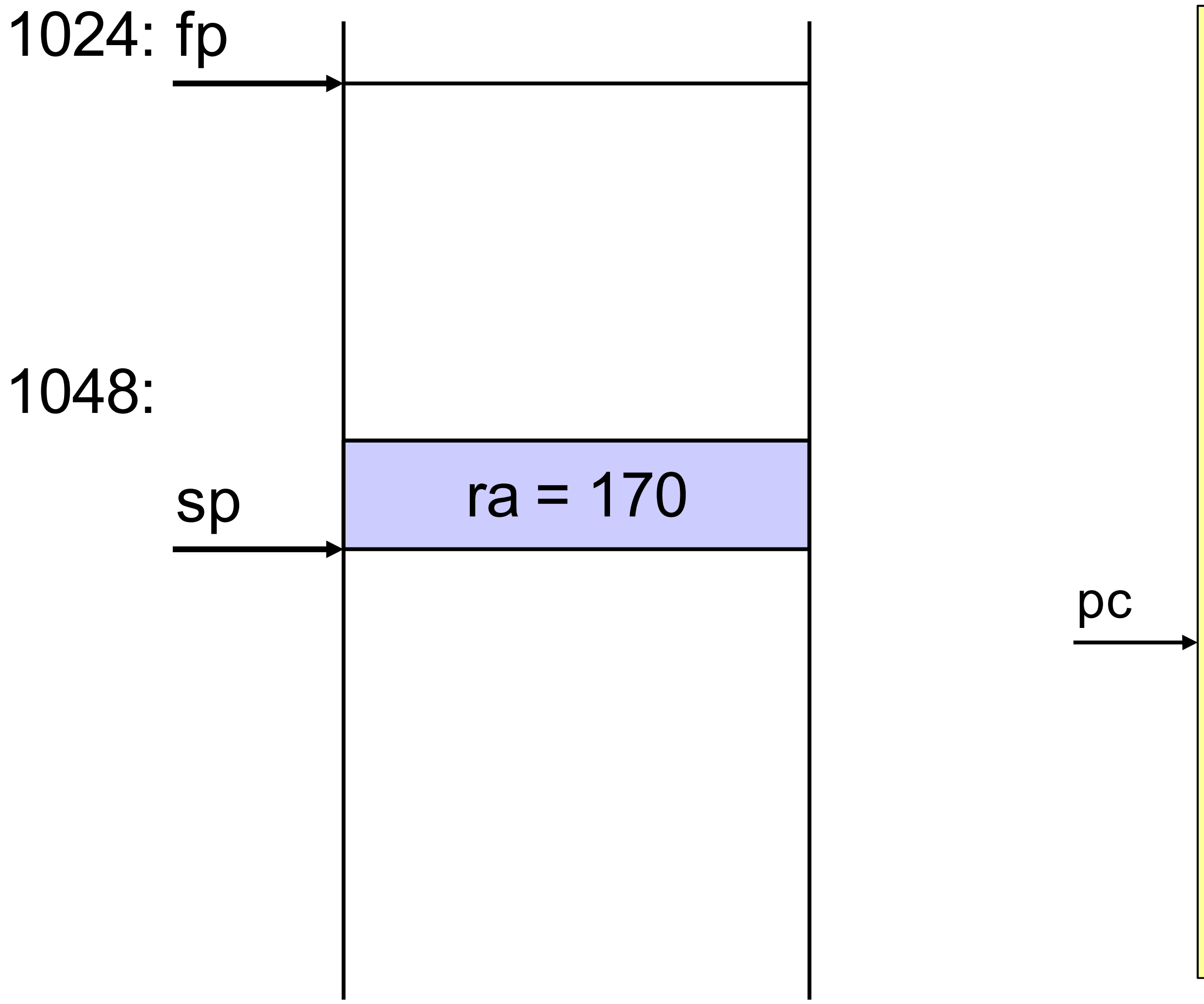
```
  0: enter 0, 1
  3: ...
  ...
 25: exit
 26: return
 27: enter 1, 2
 30: ...
  ...
 61: exit
 62: return
  ...
167: call –167
170: ...
  ...
228: call –201
231: ...
```

Methodenaufruf m1



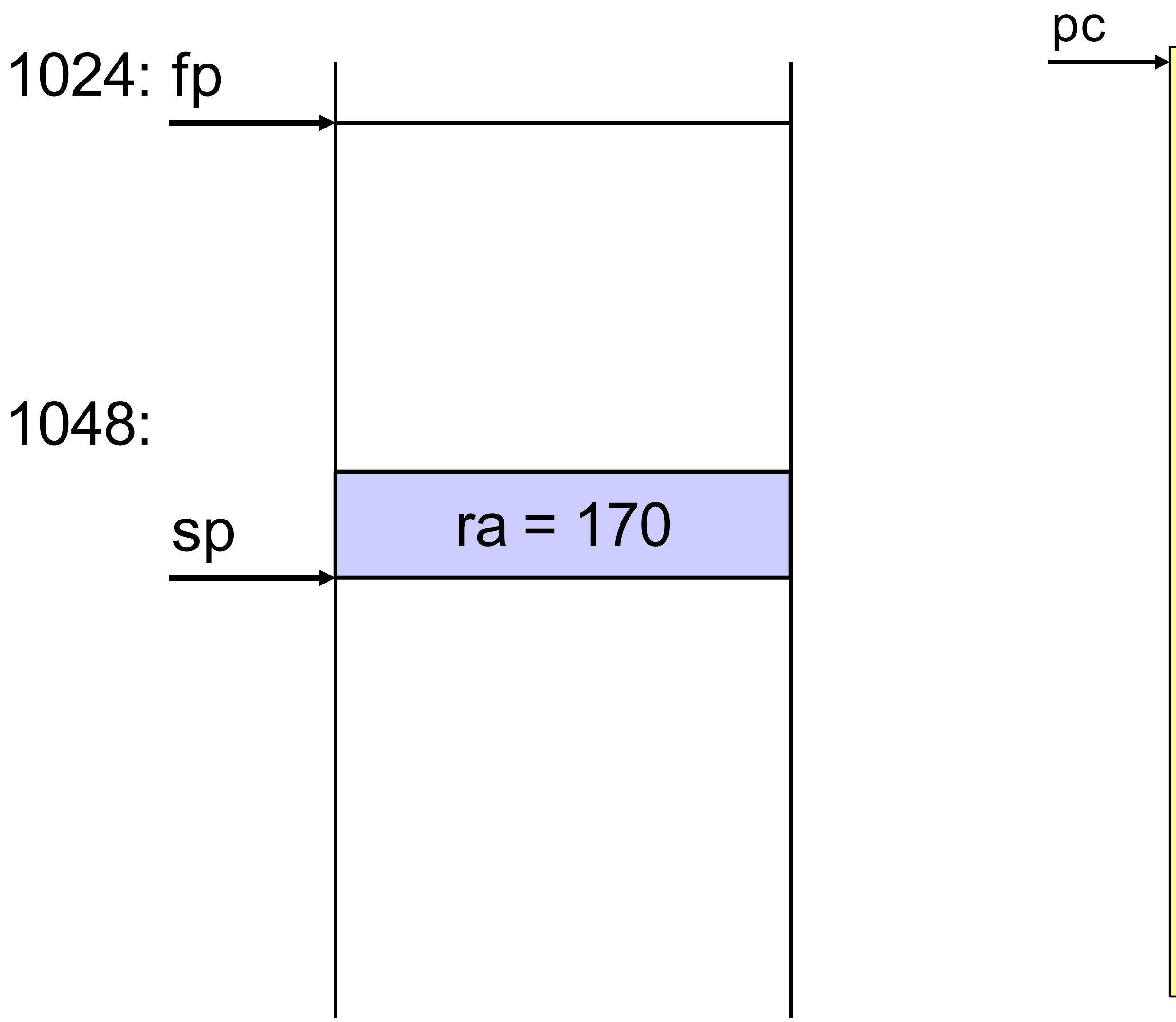
```
 0: enter 0, 1
 3: ...
...
25: exit
26: return
27: enter 1, 2
30: ...
...
61: exit
62: return
...
167: **call –167**
170: ...
...
228: call –201
231: ...
```

Methodenaufruf m1



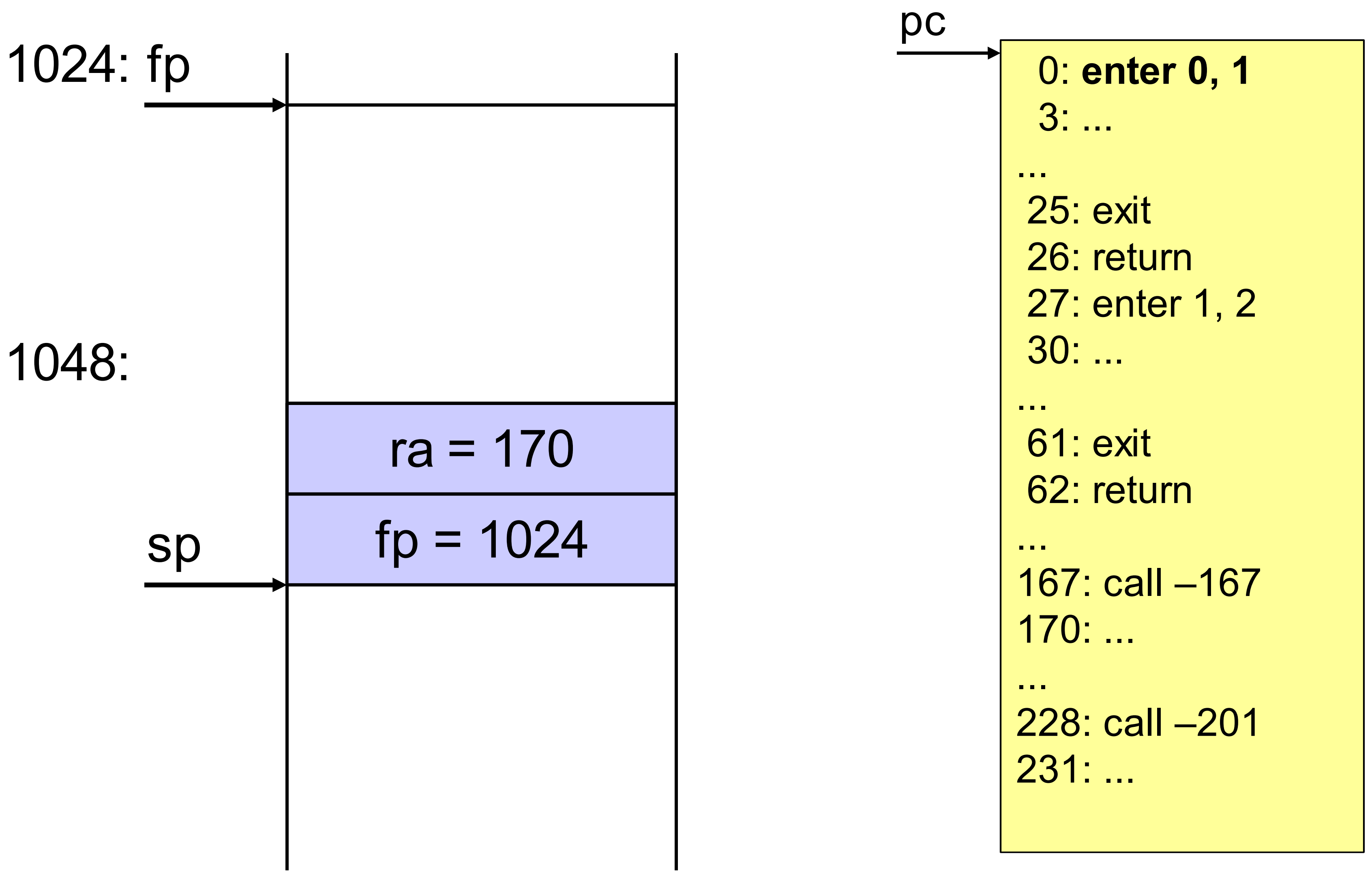
```
 0: enter 0, 1
 3: ...
...
 25: exit
 26: return
 27: enter 1, 2
 30: ...
...
 61: exit
 62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

Einsprung in Methode m1



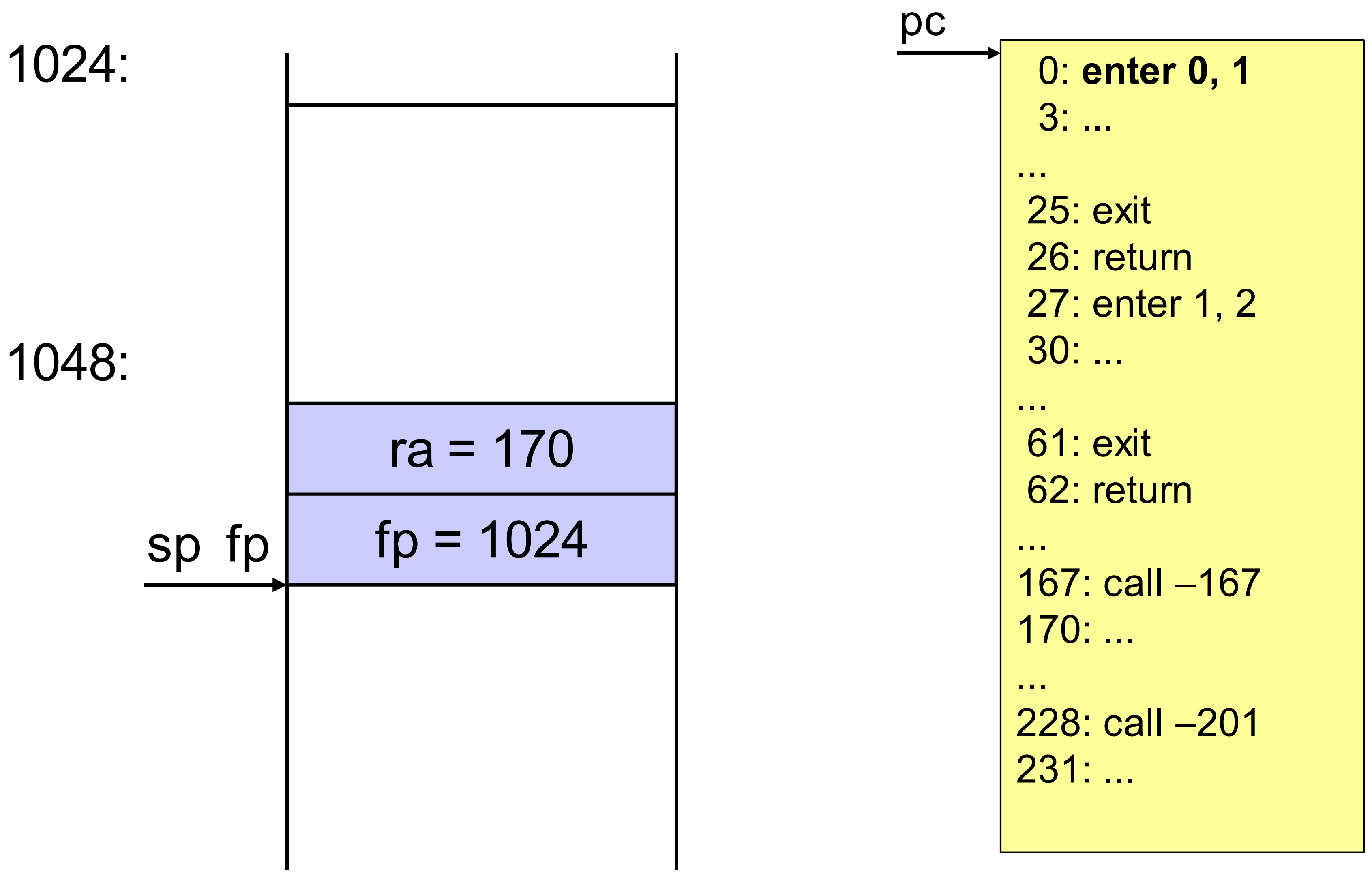
```
 0: enter 0, 1
 3: ...
...
 25: exit
 26: return
 27: enter 1, 2
 30: ...
...
 61: exit
 62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

Einsprung in Methode m1



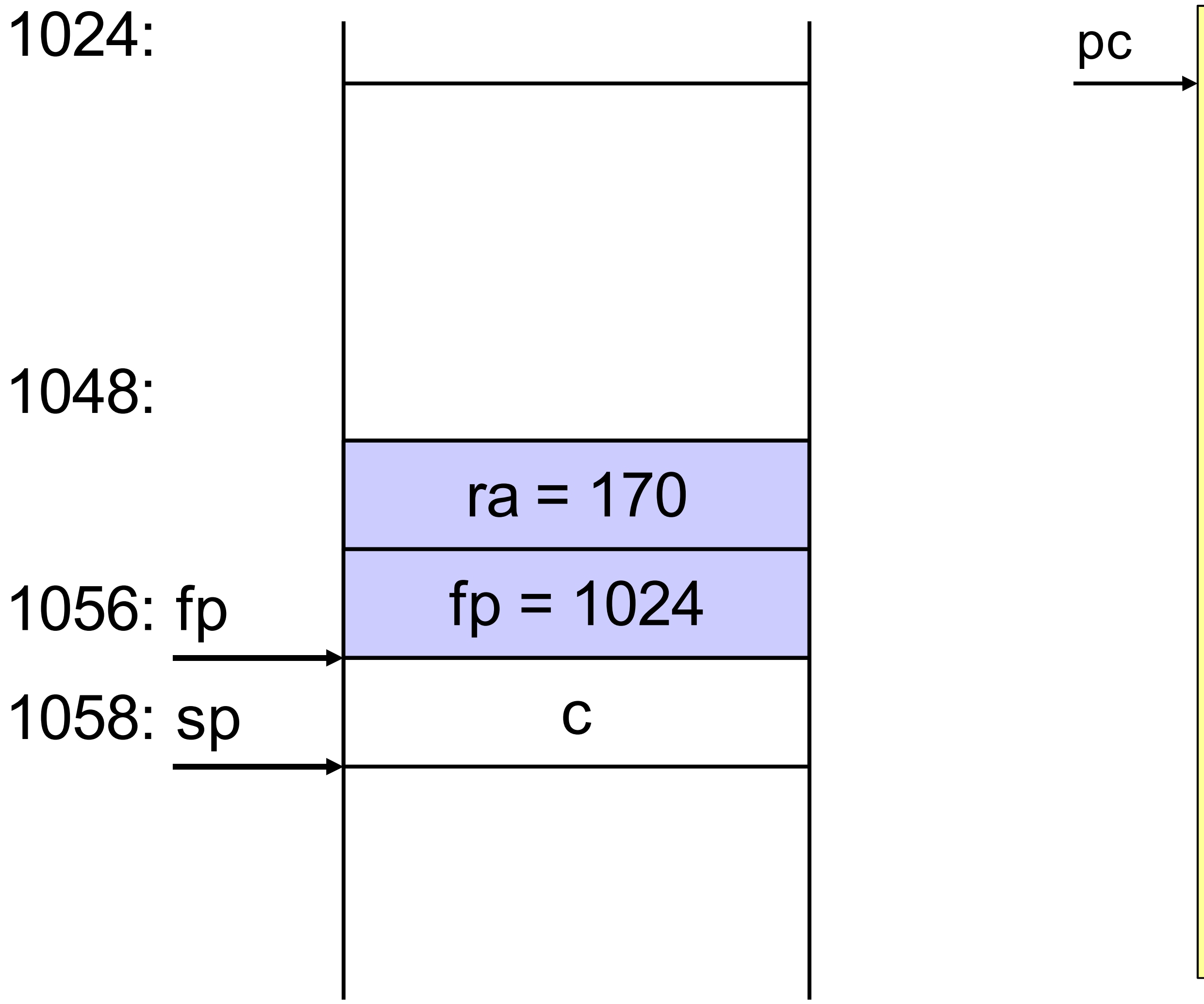
```
 0: enter 0, 1
 3: ...
...
 25: exit
 26: return
 27: enter 1, 2
 30: ...
...
 61: exit
 62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

Einsprung in Methode m1



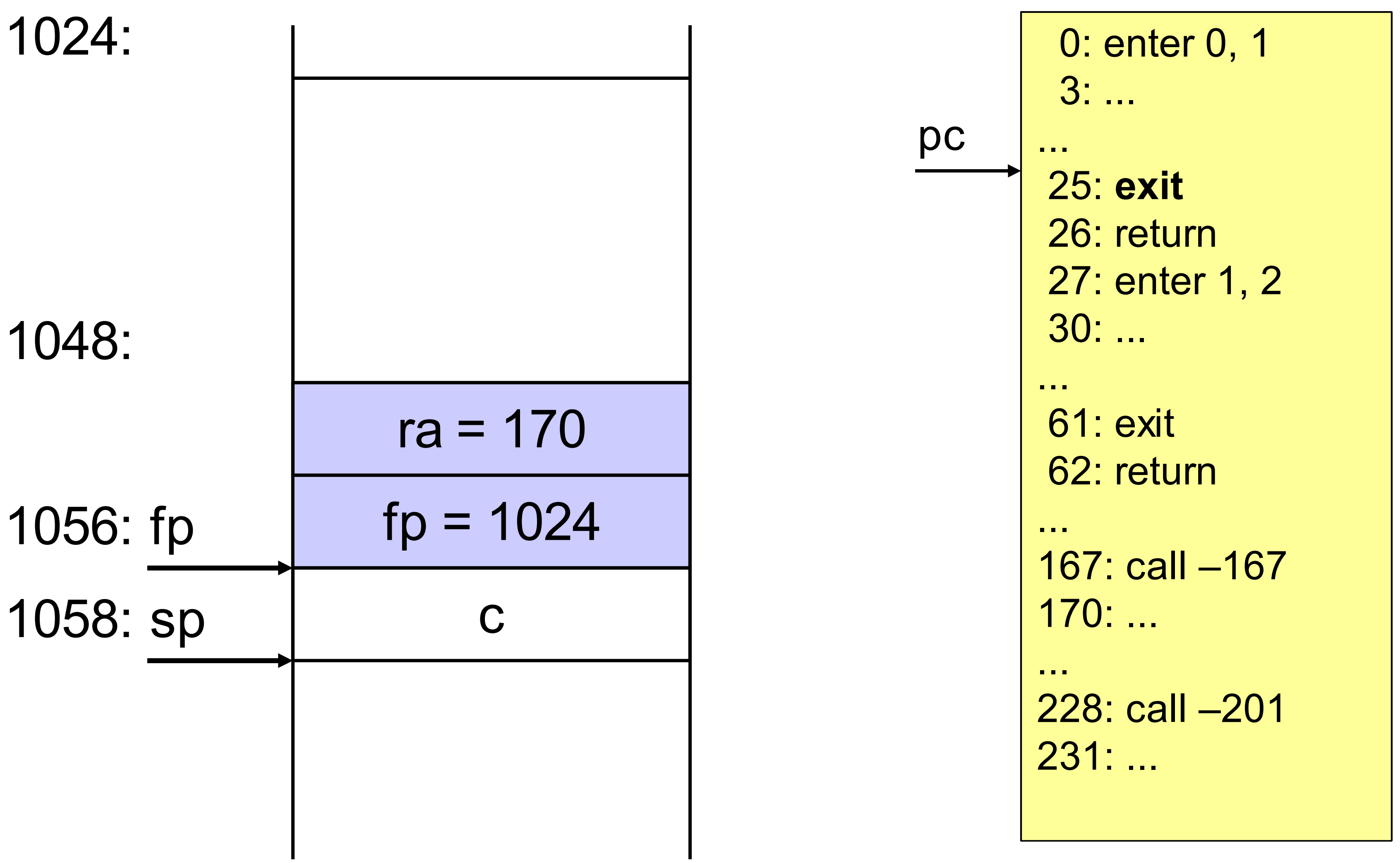
```
 0: enter 0, 1
 3: ...
...
 25: exit
 26: return
 27: enter 1, 2
 30: ...
...
 61: exit
 62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

Einsprung in Methode m1



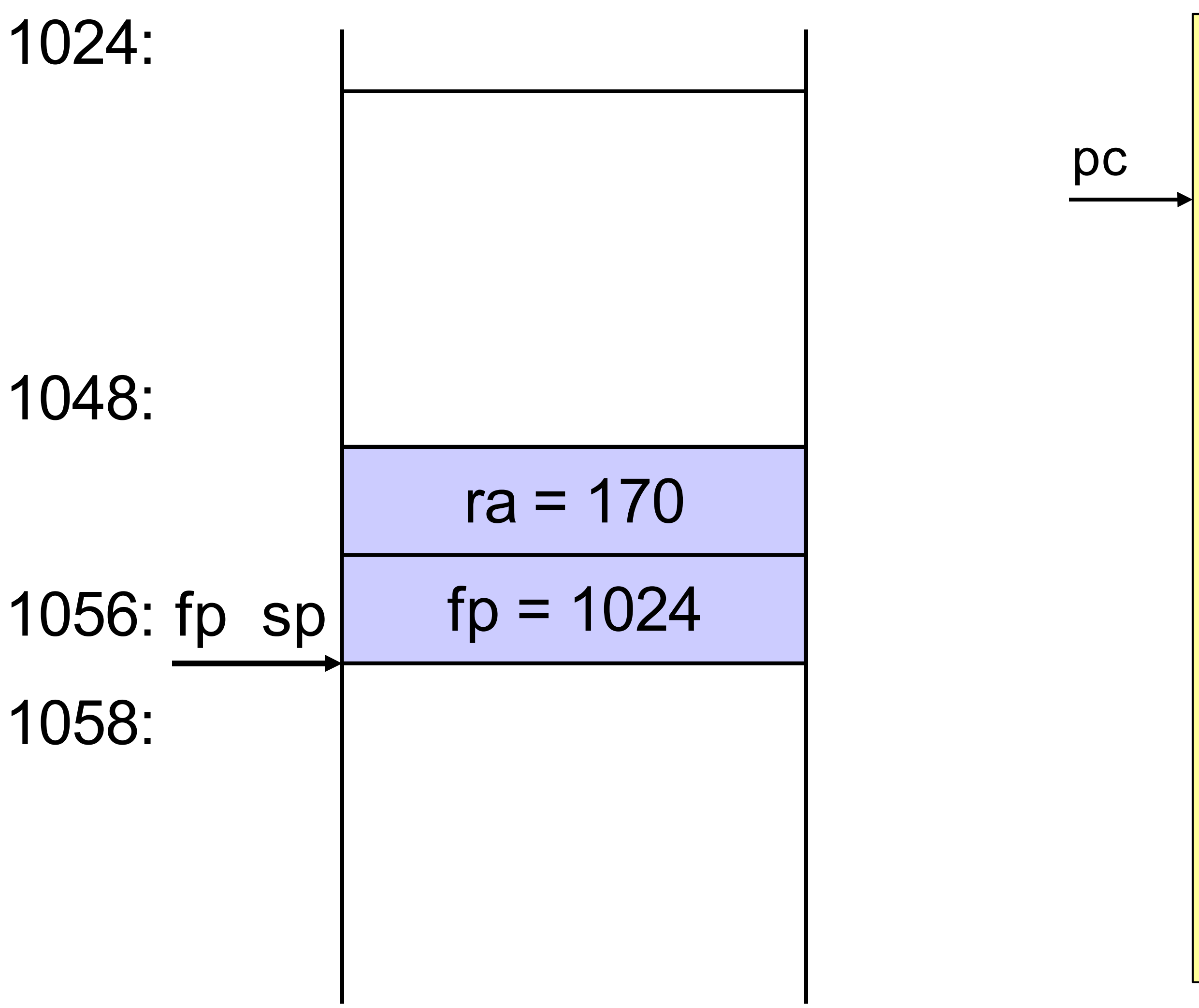
```
  0: enter 0, 1
  3: ...
...
 25: exit
 26: return
 27: enter 1, 2
 30: ...
...
 61: exit
 62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

Ende der Methode m1



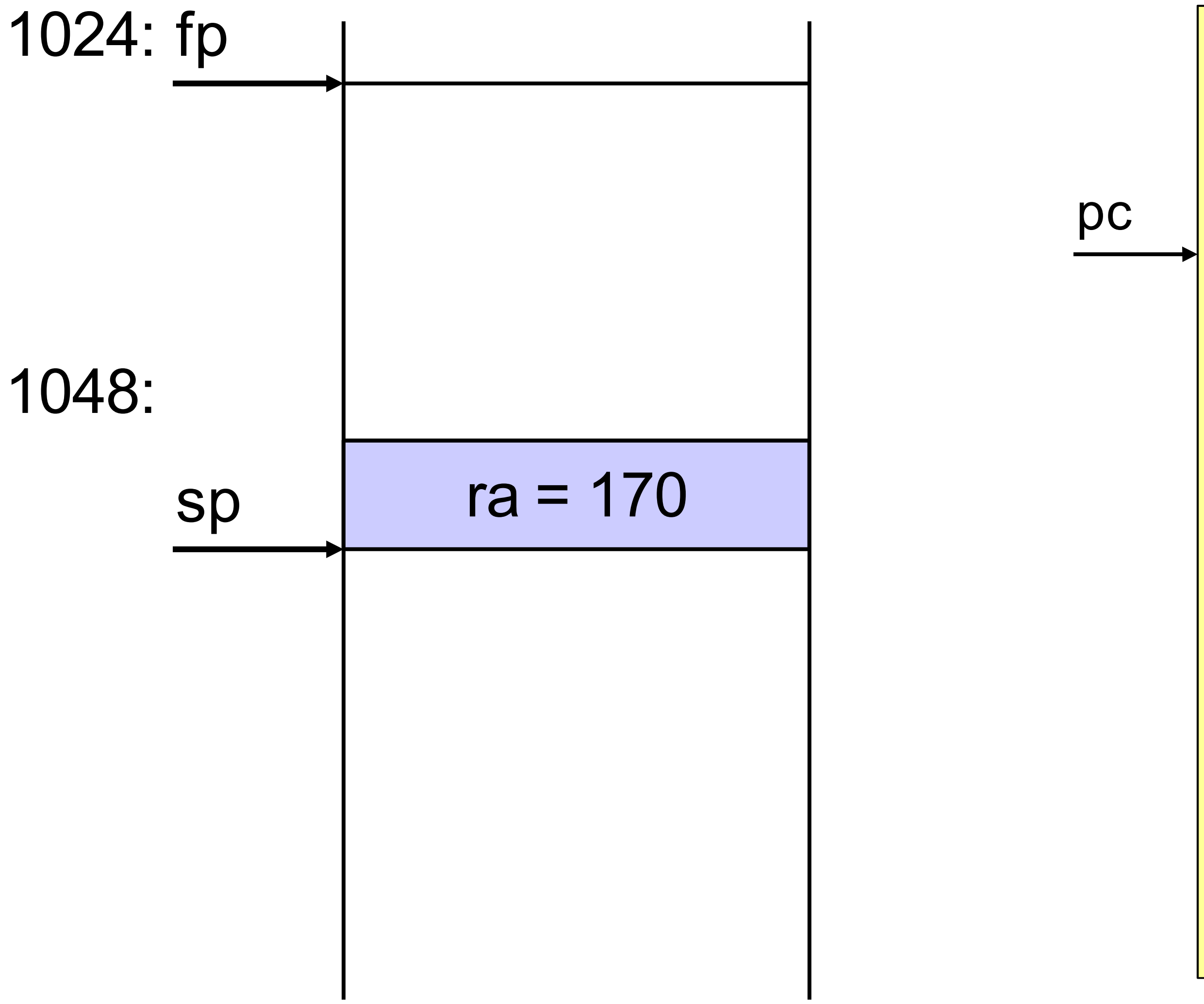
```
 0: enter 0, 1
 3: ...
...
 25: **exit**
 26: return
 27: enter 1, 2
 30: ...
...
 61: exit
 62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

Ende der Methode m1



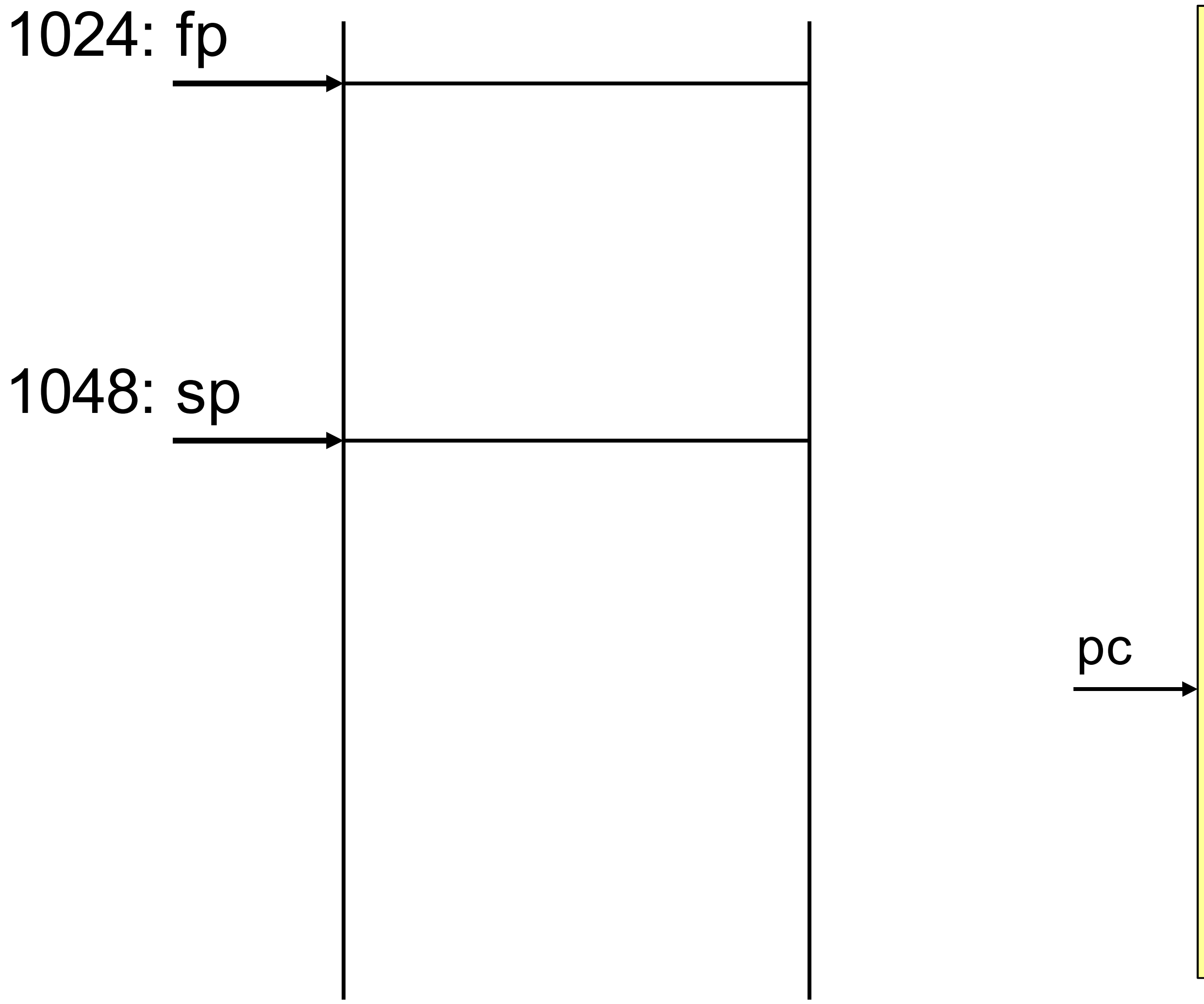
```
 0: enter 0, 1
 3: ...
...
25: **exit**
26: return
27: enter 1, 2
30: ...
...
61: exit
62: return
...
167: call –167
170: ...
...
228: call –201
231: ...
```

Rücksprung zum Rufer der Methode m1



```
 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



```
  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: ...
```

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

    Label ();           // creates a new, undefined label

    // inserts offset to label at current pc
    void put ();
    // defines label to be at current pc
    void here ();
    // defines this label to be at position of dest
    void setTo (Label dest);

    // links the other's forward jumps with this's
    // NOT needed for MicroJava-Programs
    void add (Label other);
}
```

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
    Obj obj;              // Meth: Methodenobjekt aus Symbolliste
    int adr;              // Con: Wert; Local, Static, Fld, Meth: Adresse
                          // Cond: Operator (eq=0,ne=1,…)
    Label  tLabel,        // Cond: true jump
           fLabel;        // Cond: 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 = Code.pc;
    if (defined) Code.put2(adr - (pc-1));
    else { Code.put2(adr); adr = pc; }
}
```

Klasse *Label* - Methode *here*

```
// defines label to be at current pc
void here () {
    if (defined) throw new Error(LAB_DEF);

    fixup(Code.pc);
}

// fixup forward jumps to jump to destPC
private void fixup (int destPC) {
    while (adr != 0) {
        int pos = adr;
        adr = Code.get2(adr);
        Code.put2(pos, destPC - (pos-1));
    }
    defined = true; adr = destPC;
}
```

Bsp 10: **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 11: **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 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:   jle 8                  (--> 22)
17:   load_2
18:   const_0
19:   jge 5         (--> 24)
22:   const_0
23:   store_2
24:   ...
```

Bsp 13: **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 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:   ...
```

Übersetzung einer do-while-Anweisung

```
do                          top:
    Statement                   code for Statement
while                           code for Condition
    Condition;                  tJump to top
...                             ...
```

```
DoStatement =               (. Item x; Label top; .)
    "do"                    (. top = new Label(); top.here(); .)
    Statement
    "while"
    "(" Condition↑x         (. x.tLabel.setTo(top);
                               Code.tJump(x); .)
    ")"                     (. x.fLabel.here(); .)
    ";"
```

Klasse *Label* - Methode *setTo*

```
// defines this label to be at position of dest
void setTo (Label dest) {
    if (defined) throw new Error(LAB_DEF);
    if (!dest.defined) throw new Error(DEST_UNDEF);

    fixup(dest.adr);
}
```

Bsp 15: **do n++ while (i<=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:    inc 2 1
13:    getstatic 1
16:    load_2
17:    jle -7          (--> 10)
20:    ...
```

Bsp 14: **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:    ...
```