EECS483 D14: Final Review

Chun-Hung Hsiao

April 19, 2013

Announcements

- PP5 Scoreboard
  - http://www.umich.edu/~chhsiao/pp5-scoreboard.html
- You should be able to pass the performance tests by doing a good register allocation
- Final exam would be cumulative
- Additional office hours
  - Supriya during 1:30pm-3pm Monday, April 29
  - Chun-Hung during 10:30am-12pm Tuesday, Apr 30

HW4 Q2 (1/2)

- S → int
S → float
S → S S +
S → S floor
- Type rules for "S → S floor" is easy

$$\frac{A \vdash S : \text{float\_type}}{A \vdash S \text{ floor} : \text{int\_type}}$$

- Type rules for "S → S S +"

$$\frac{\begin{array}{c} A \vdash S_1 : \text{int\_type} \\ A \vdash S_2 : \text{int\_type} \end{array}}{A \vdash S_1\ S_2 + : \text{int\_type}} \qquad \frac{\begin{array}{c} A \vdash S_1 : T_1 \\ A \vdash S_2 : T_2 \\ T_1 \text{ or } T_2 \text{ is float\_type} \end{array}}{A \vdash S_1\ S_2 + : \text{float\_type}}$$

HW4 Q2 (2/2)

- $S \rightarrow \text{int}$
  $S \rightarrow \text{float}$
  $S \rightarrow S\ S\ +$
  $S \rightarrow S\ \text{floor}$

- Alternately, type rules for “S → S S +” could be

$$\frac{\begin{array}{c} A \vdash S_1 : T_1 \\ A \vdash S_2 : T_2 \end{array}}{A \vdash S_1\ S_2 + : \max(T_1, T_2)} \qquad \frac{}{\text{int\_type} \leq \text{float\_type}}$$

- Type rules for “S → int” and “S → float”

$$\frac{}{A \vdash \text{int} : \text{int\_type}} \qquad \frac{}{A \vdash \text{float} : \text{float\_type}}$$

HW4 Q4 (1/4)

- Pass by value: the values of the actual arguments are copied to the stack

```
void main () {
  int value = 2, list[5] = {1, 3, 5, 7, 9};
  swap (value, list[0]);
  swap (list[0], list[1]);
  swap (value, list[value]);
    // addr = list + value * 4
    // PushParam *addr
    // PushParam value
    // LCall swap
}
```

HW4 Q4 (2/4)

- Pass by reference: the addresses of the actual arguments are copied to the stack; a dereference happens when accessing a formal parameter

```
void main () {
  int value = 2,
      list[5] = {1, 3, 5, 7, 9};
  swap (value, list[0]);
  swap (list[0], list[1]);
  swap (value, list[value]);
    // addr = list + value * 4
    // PushParam addr
    // PushParam &value
    // LCall swap
}
```

```
void swap (int a, int b) {
  int temp;
  temp = a;
    // temp = *a
  a = b;
    // *a = *b
  b = temp;
    // *b = temp
}
```

HW4 Q4 (3/4)

- Pass by name: “how the actual arguments are evaluated” are passed; an evaluation happens when accessing a formal parameter

```
void main () {
 int value = 2,
   list[5] = {1, 3, 5, 7, 9};
 swap (value, list[0]);
 swap (list[0], list[1]);
 swap (value, list[value]);
  // PushParam `list + value * 4`
  // PushParam `&value`
  // LCall swap
}
```

```
void swap (int a, int b) {
 int temp;
 temp = a;
  // a = &value
  // temp = *a
 a = b;
  // b = list + value * 4
  // a = &value
  // *a = *b
 b = temp;
  // b = list + value * 4
  // *b = temp
}
```

HW4 Q4 (4/4)

- Pass by value-result: the values of the actual arguments are copied, and the changes to formal parameters are copied back after function execution

```
void main () {
 int value = 2,
   list[5] = {1, 3, 5, 7, 9};
 swap (value, list[0]);
 swap (list[0], list[1]);
 swap (value, list[value]);
  // addr = list + value * 4
  // PushParam addr
  // PushParam &value
  // LCall swap
}
```

```
void swap (int a, int b) {
  // a' = *a
  // b' = *b
 int temp;
 temp = a;
  // temp = a'
 a = b;
  // a' = b'
 b = temp;
  // b' = temp
  // *a = a'
  // *b = b'
}
```

Topics after Midterm

- Type system
- Intermediate representation / 3-address code
- Activation records
- Dynamic dispatch
- Dataflow analysis
- Local / global optimizations
- Register allocation
- Garbage collection
- Static single assignment (SSA) form

3-Address Code

- Complete the 3-address code for the following Decaf program

```
int gcd(int x, int y){
  if (x % y == 0)
    return y;
  return gcd(y, x % y);
}
```

```
_gcd:
  BeginFunc 32;
  _t0 = x % y;
  _t1 = 0;
  _t2 = _t0 == _t1;

    // if-then

_L1:


    // gcd(y, x % y)


  Return _t3;
  EndFunc;
```

3-Address Code

- Complete the 3-address code for the following Decaf program

```
int gcd(int x, int y){
  if (x % y == 0)
    return y;
  return gcd(y, x % y);
}
```

```
_gcd:
  BeginFunc 32;
  _t0 = x % y;
  _t1 = 0;
  _t2 = _t0 == _t1;
  IfZ _t2 Goto _L1;
  Return y;
_L1:


    // gcd(y, x % y)


  Return _t3;
  EndFunc;
```

3-Address Code

- Complete the 3-address code for the following Decaf program

```
int gcd(int x, int y){
  if (x % y == 0)
    return y;
  return gcd(y, x % y);
}
```

```
_gcd:
  BeginFunc 16;
  _t0 = x % y;
  _t1 = 0;
  _t2 = _t0 == _t1;
  IfZ _t2 Goto _L1;
  Return y;
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParam 8;
  Return _t3;
  EndFunc;
```

Activation Records

- Draw the stack when Line 7 is executed

```
 1 _gcd: // gcd(x,y)
 2    BeginFunc 16;
 3    _t0 = x % y;
 4    _t1 = 0;
 5    _t2 = _t0 == _t1;
 6    IfZ _t2 Goto _L1;
 7    Return y;
 8 _L1:
 9    PushParam _t0;
10    PushParam y;
11    _t3 = LCall _gcd;
12    PopParams 8;
13    Return _t3;
14    EndFunc;
```

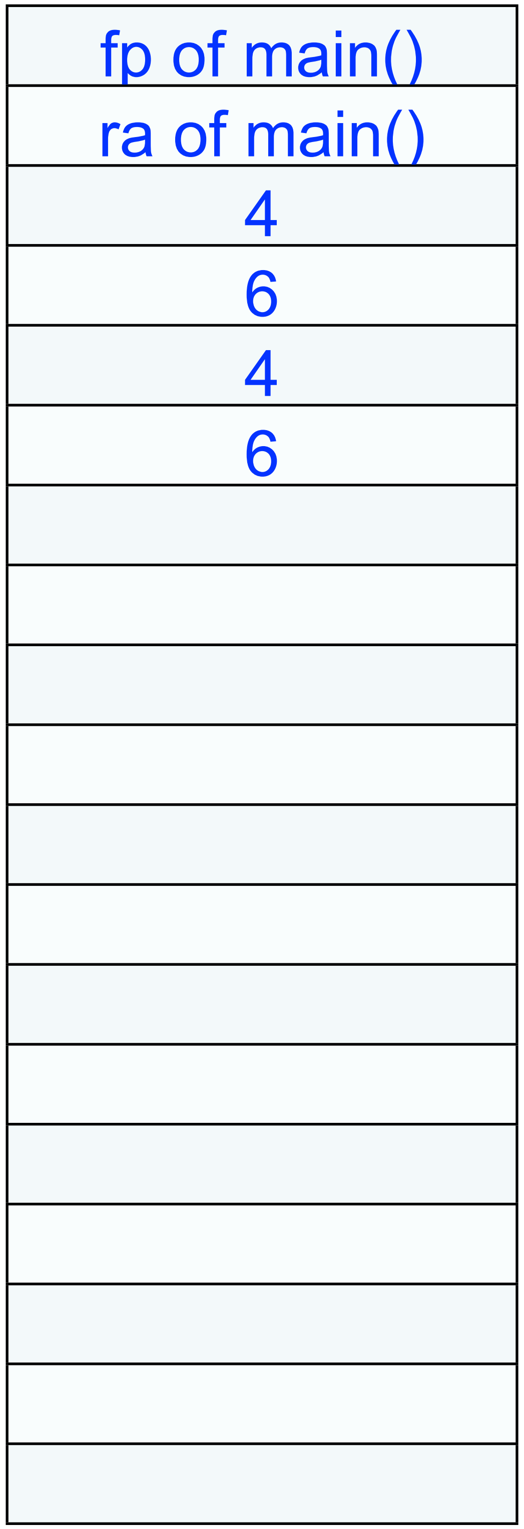
```
15 main: // main()
16    BeginFunc 8;
17    _t4 = 4;
18    _t5 = 6;
19    PushParam _t4;
20    PushParam _t5;
21    Lcall _gcd;
22    PopParams 8
23    EndFunc;
```

Activation Records

- Draw the stack when Line 7 is executed

```
1 _gcd: // gcd(x,y)
2   BeginFunc 16;
3   _t0 = x % y;
4   _t1 = 0;
5   _t2 = _t0 == _t1;
6   IfZ _t2 Goto _L1;
7   Return y;
8 _L1:
9   PushParam _t0;
10  PushParam y;
11  _t3 = LCall _gcd;
12  PopParams 8;
13  Return _t3;
14  EndFunc;
```

```
15 main: // main()
16   BeginFunc 8;
17   _t4 = 4;
18   _t5 = 6;
19   PushParam _t4;
20   PushParam _t5;
21   Lcall _gcd;
22   PopParams 8
23   EndFunc;
```

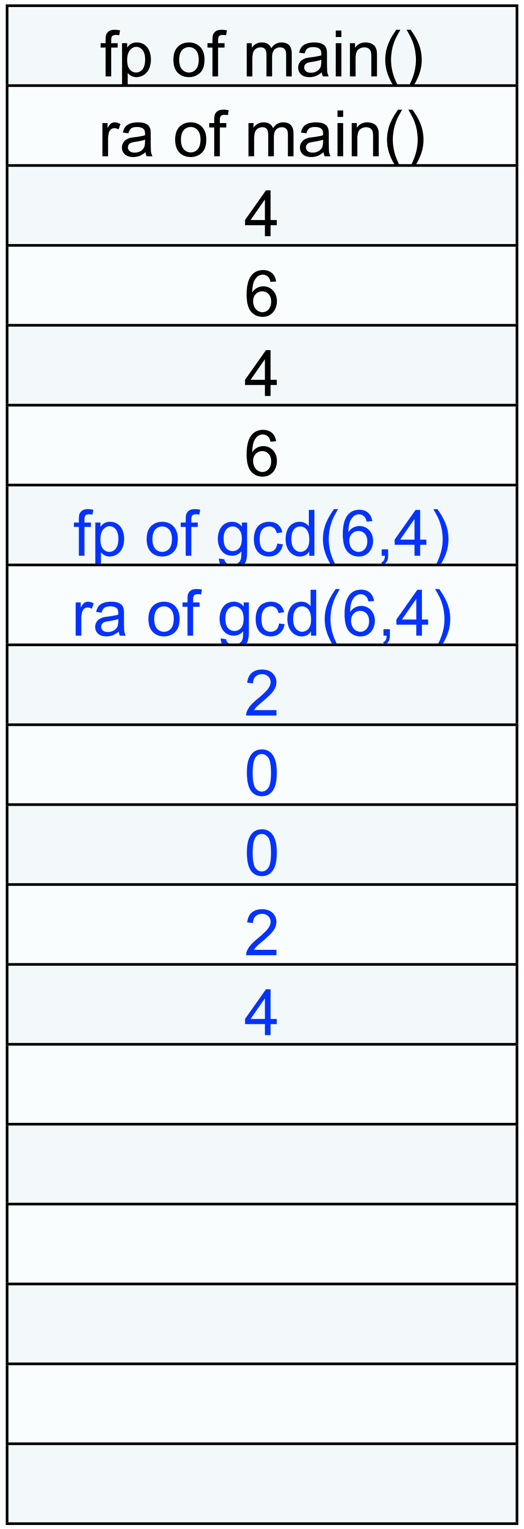

fp of main()
ra of main()
4
6
4
6

Activation Records

- Draw the stack when Line 7 is executed

```
 1 _gcd: // gcd(x,y)
 2    BeginFunc 16;
 3    _t0 = x % y;
 4    _t1 = 0;
 5    _t2 = _t0 == _t1;
 6    IfZ _t2 Goto _L1;
 7    Return y;
 8 _L1:
 9    PushParam _t0;
10    PushParam y;
11    _t3 = LCall _gcd;
12    PopParams 8;
13    Return _t3;
14    EndFunc;
```

```
15 main: // main()
16    BeginFunc 8;
17    _t4 = 4;
18    _t5 = 6;
19    PushParam _t4;
20    PushParam _t5;
21    Lcall _gcd;
22    PopParams 8
23    EndFunc;
```



fp of main()
ra of main()
4
6
4
6
fp of gcd(6,4)
ra of gcd(6,4)
2
0
0
2
4

Activation Records

- Draw the stack when Line 7 is executed

```
 1 _gcd: // gcd(x,y)
 2    BeginFunc 16;
 3    _t0 = x % y;
 4    _t1 = 0;
 5    _t2 = _t0 == _t1;
 6    IfZ _t2 Goto _L1;
 7    Return y;
 8 _L1:
 9    PushParam _t0;
10    PushParam y;
11    _t3 = LCall _gcd;
12    PopParams 8;
13    Return _t3;
14    EndFunc;
```

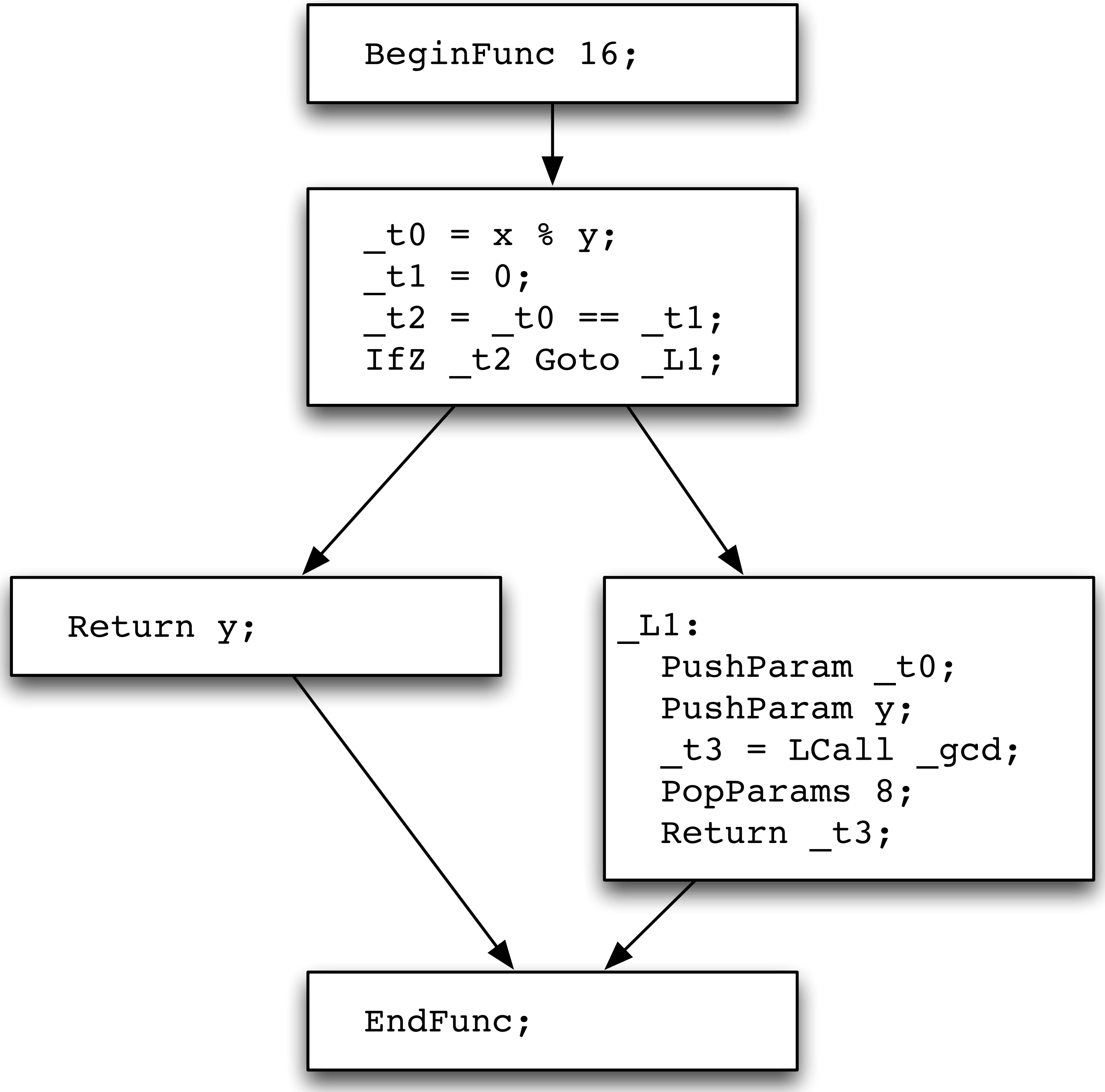
```
15 main: // main()
16    BeginFunc 8;
17    _t4 = 4;
18    _t5 = 6;
19    PushParam _t4;
20    PushParam _t5;
21    Lcall _gcd;
22    PopParams 8
23    EndFunc;
```

fp of main()
ra of main()
4
6
4
6
fp of gcd(6,4)
ra of gcd(6,4)
2
0
0
2
4
fp of gcd(4,2)
ra of gcd(4,2)
0
0
1

Dataflow Analysis (1/2)

- Apply the liveness analysis on the following function

```
_gcd:
   BeginFunc 16;
   _t0 = x % y;
   _t1 = 0;
   _t2 = _t0 == _t1;
   IfZ _t2 Goto _L1;
   Return y;
_L1:
   PushParam _t0;
   PushParam y;
   _t3 = LCall _gcd;
   PopParams 8;
   Return _t3;
   EndFunc;
```



```
BeginFunc 16;
```

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

```
Return y;
```

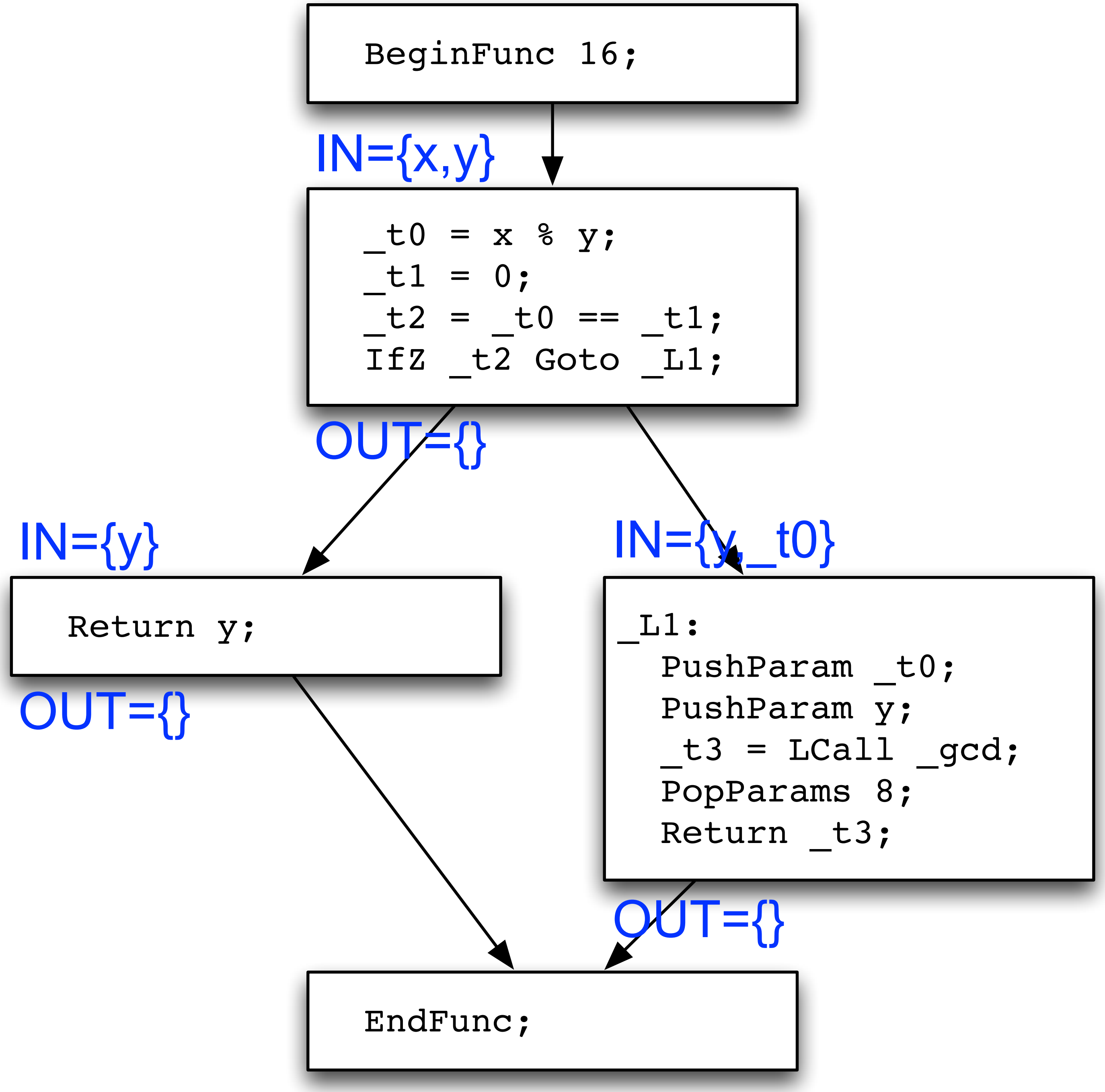
```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

```
EndFunc;
```

Dataflow Analysis (1/2)

- Apply the liveness analysis on the following function

```
_gcd:
  BeginFunc 16;
  _t0 = x % y;
  _t1 = 0;
  _t2 = _t0 == _t1;
  IfZ _t2 Goto _L1;
  Return y;
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
  EndFunc;
```



```
BeginFunc 16;
```

IN={x,y}

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

OUT={}

IN={y}

```
Return y;
```

OUT={}

IN={y,_t0}

```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

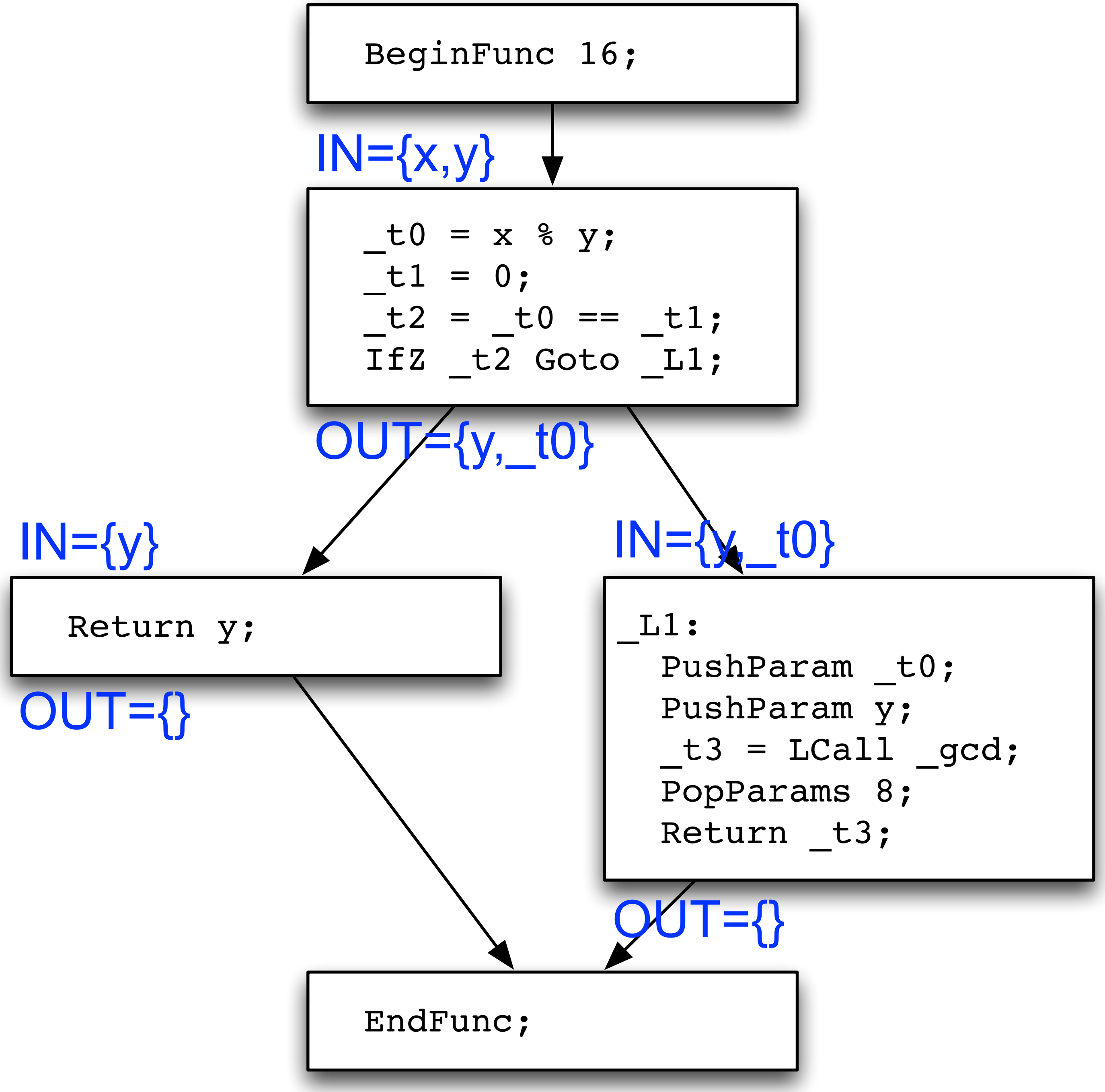
OUT={}

```
EndFunc;
```

Dataflow Analysis (1/2)

- Apply the liveness analysis on the following function

```
_gcd:
   BeginFunc 16;
   _t0 = x % y;
   _t1 = 0;
   _t2 = _t0 == _t1;
   IfZ _t2 Goto _L1;
   Return y;
_L1:
   PushParam _t0;
   PushParam y;
   _t3 = LCall _gcd;
   PopParams 8;
   Return _t3;
   EndFunc;
```



```
BeginFunc 16;
```

IN={x,y}

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

OUT={y,_t0}

IN={y}

```
Return y;
```

OUT={}

IN={y,_t0}

```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

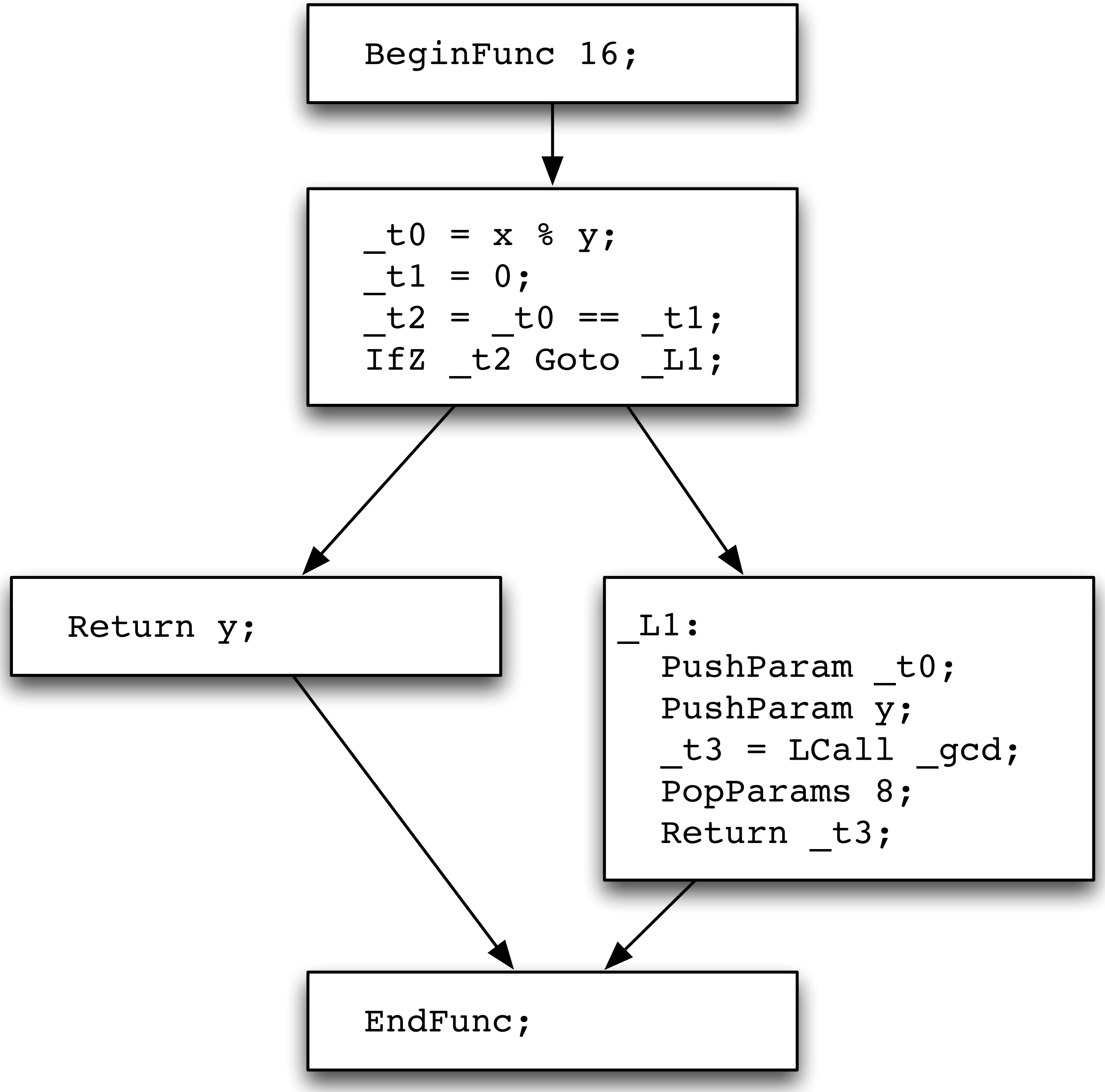
OUT={}

```
EndFunc;
```

Dataflow Analysis (2/2)

- Apply the available expression analysis on the following function

```
_gcd:
    BeginFunc 16;
    _t0 = x % y;
    _t1 = 0;
    _t2 = _t0 == _t1;
    IfZ _t2 Goto _L1;
    Return y;
_L1:
    PushParam _t0;
    PushParam y;
    _t3 = LCall _gcd;
    PopParams 8;
    Return _t3;
    EndFunc;
```



```
BeginFunc 16;
```

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

```
Return y;
```

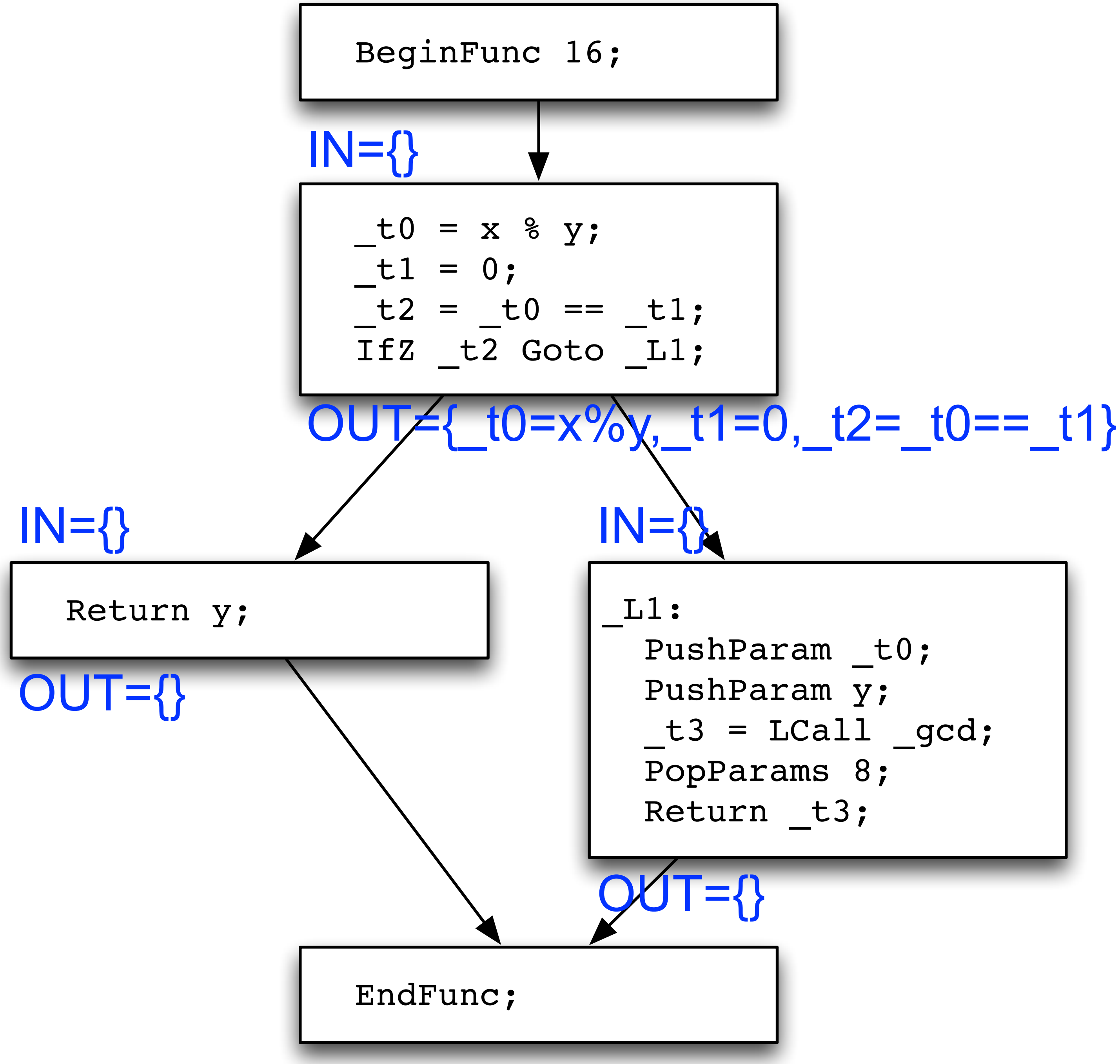
```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

```
EndFunc;
```

Dataflow Analysis (2/2)

- Apply the available expression analysis on the following function

```
_gcd:
    BeginFunc 16;
    _t0 = x % y;
    _t1 = 0;
    _t2 = _t0 == _t1;
    IfZ _t2 Goto _L1;
    Return y;
_L1:
    PushParam _t0;
    PushParam y;
    _t3 = LCall _gcd;
    PopParams 8;
    Return _t3;
    EndFunc;
```



```
BeginFunc 16;
```

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

```
Return y;
```

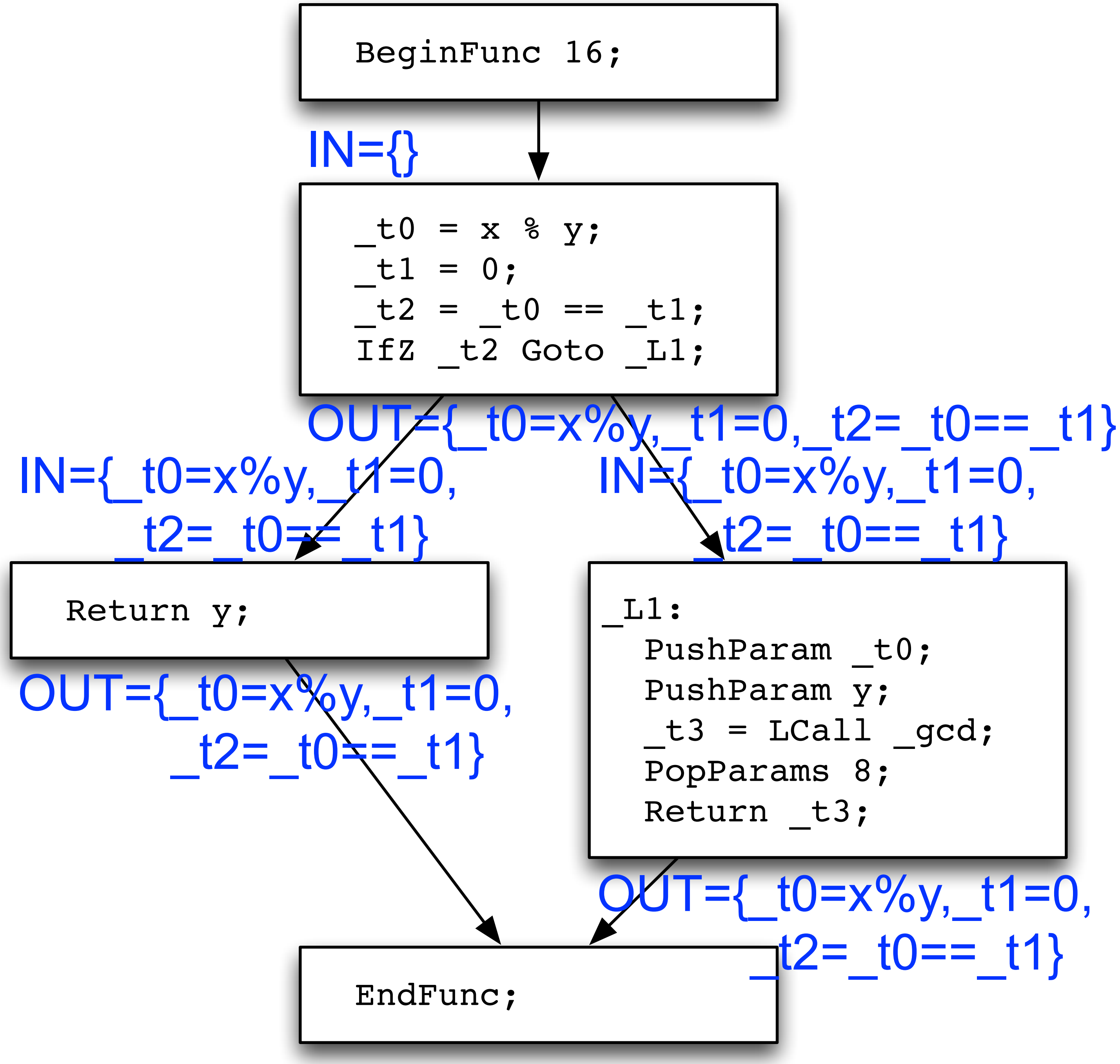
```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

```
EndFunc;
```

Dataflow Analysis (2/2)

- Apply the available expression analysis on the following function

```
_gcd:
    BeginFunc 16;
    _t0 = x % y;
    _t1 = 0;
    _t2 = _t0 == _t1;
    IfZ _t2 Goto _L1;
    Return y;
_L1:
    PushParam _t0;
    PushParam y;
    _t3 = LCall _gcd;
    PopParams 8;
    Return _t3;
    EndFunc;
```



```
BeginFunc 16;
```

IN={}

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

OUT={_t0=x%y,_t1=0,_t2=_t0==_t1}

IN={_t0=x%y,_t1=0, _t2=_t0==_t1}

```
Return y;
```

OUT={_t0=x%y,_t1=0, _t2=_t0==_t1}

IN={_t0=x%y,_t1=0, _t2=_t0==_t1}

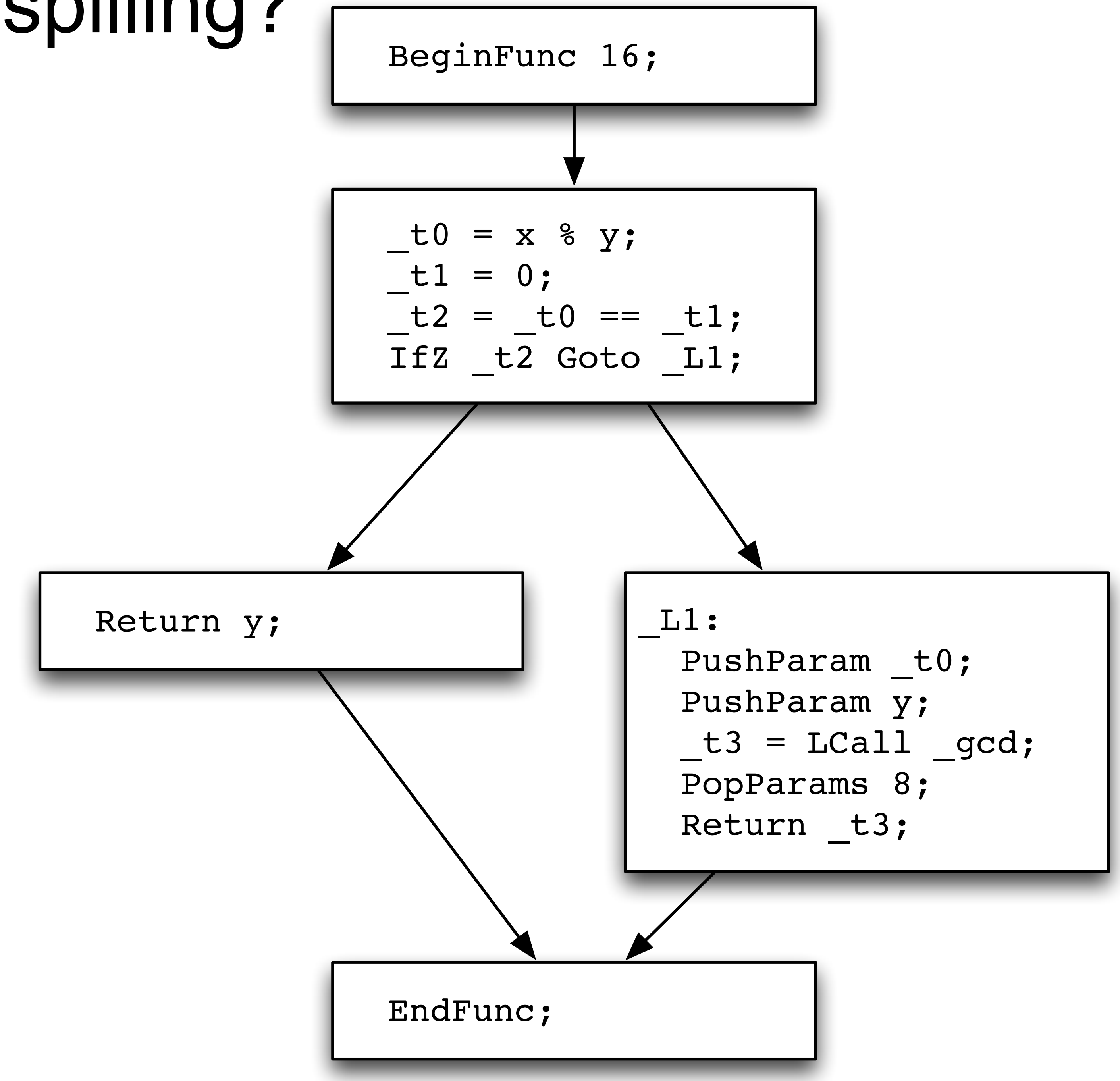
```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

OUT={_t0=x%y,_t1=0, _t2=_t0==_t1}

```
EndFunc;
```

Register Allocation (1/2)

- What is the minimum number of registers without spilling?



```
BeginFunc 16;
```

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

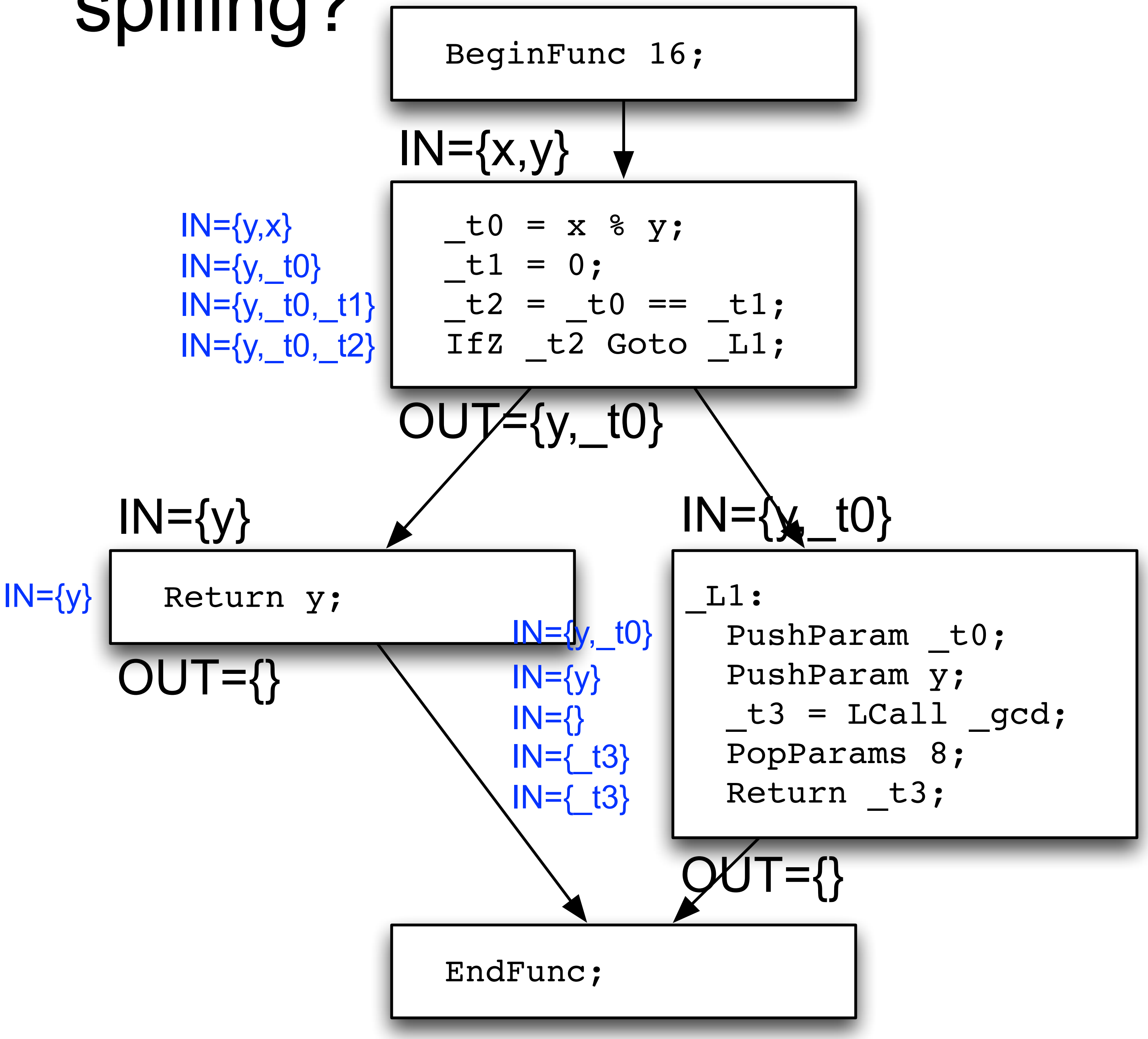
```
Return y;
```

```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

```
EndFunc;
```

Register Allocation (1/2)

- What is the minimum number of registers without spilling?



```
BeginFunc 16;
```

IN={x,y}

IN={y,x}
IN={y,_t0}
IN={y,_t0,_t1}
IN={y,_t0,_t2}

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

OUT={y,_t0}

IN={y}

IN={y}

```
Return y;
```

OUT={}

IN={y,_t0}

IN={y,_t0}
IN={y}
IN={}
IN={_t3}
IN={_t3}

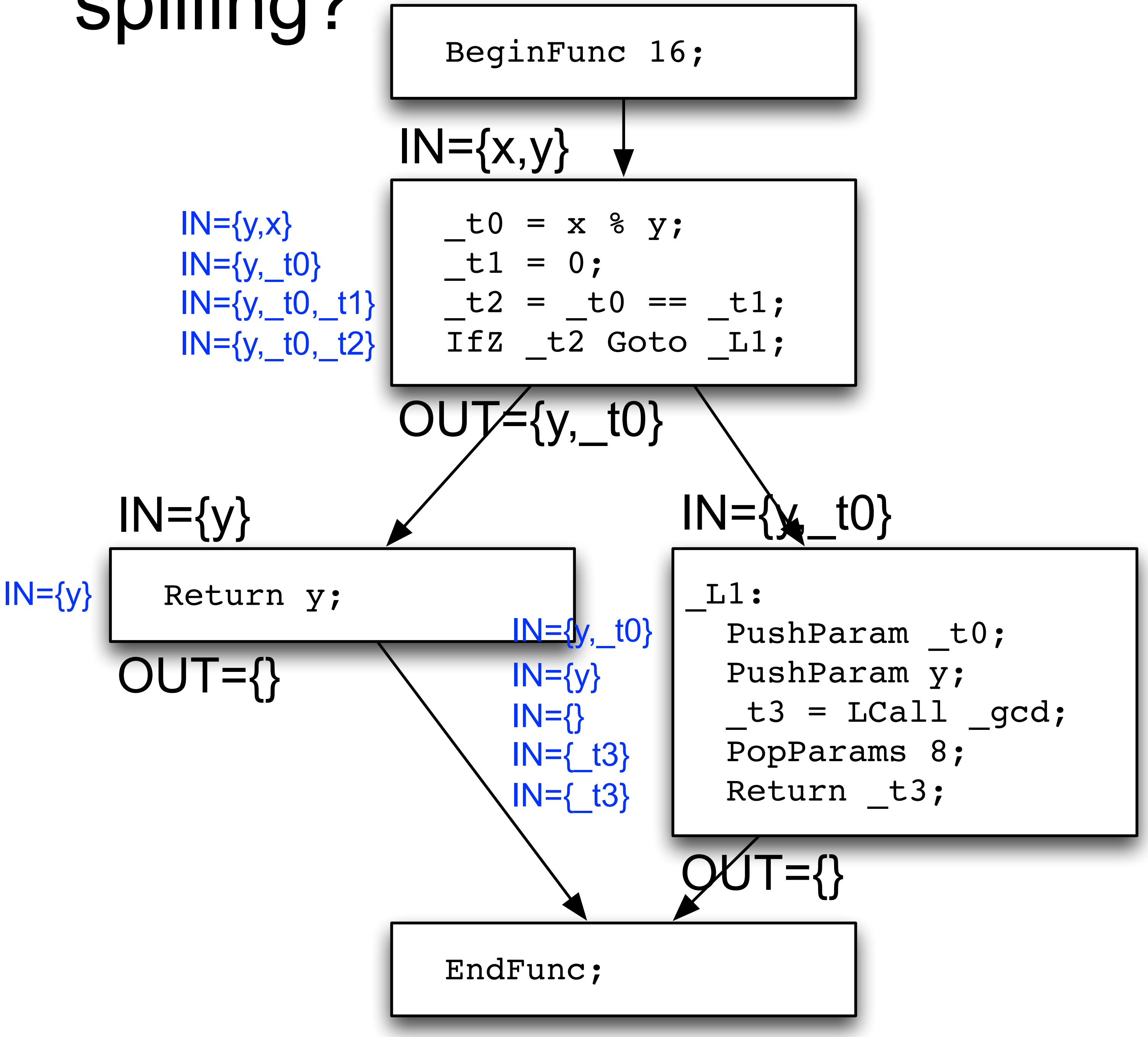
```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

OUT={}

```
EndFunc;
```

Register Allocation (1/2)

- What is the minimum number of registers without spilling?



```
BeginFunc 16;
```

IN={x,y}

IN={y,x}
IN={y,_t0}
IN={y,_t0,_t1}
IN={y,_t0,_t2}

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

OUT={y,_t0}

IN={y}

IN={y}

```
Return y;
```

OUT={}

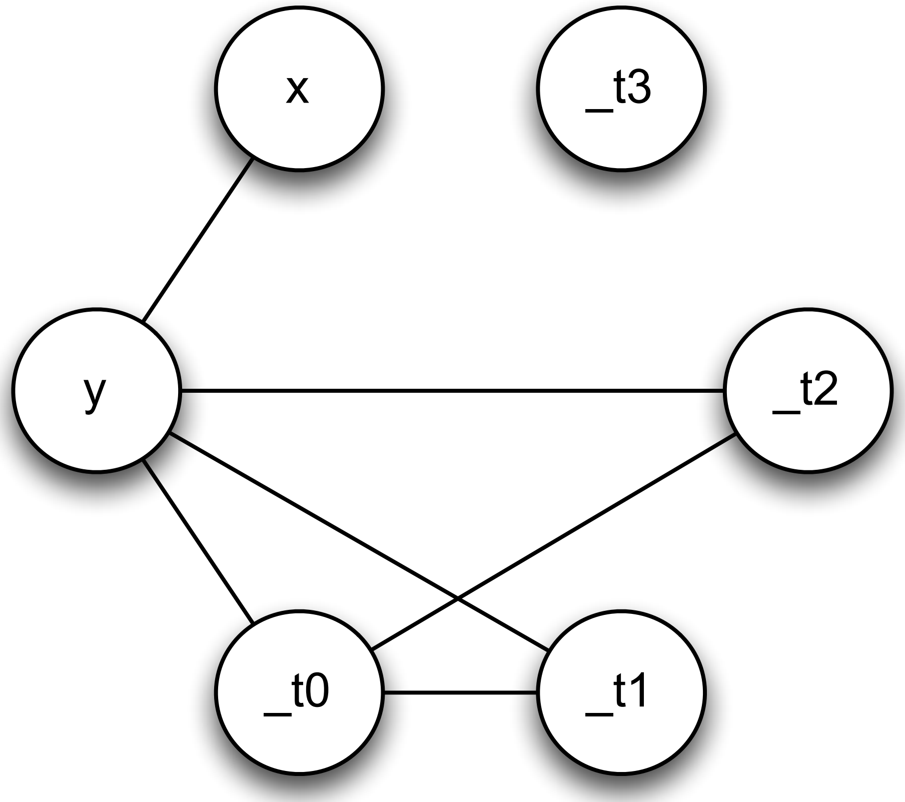
IN={y,_t0}

IN={y,_t0}
IN={y}
IN={}
IN={_t3}
IN={_t3}

```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

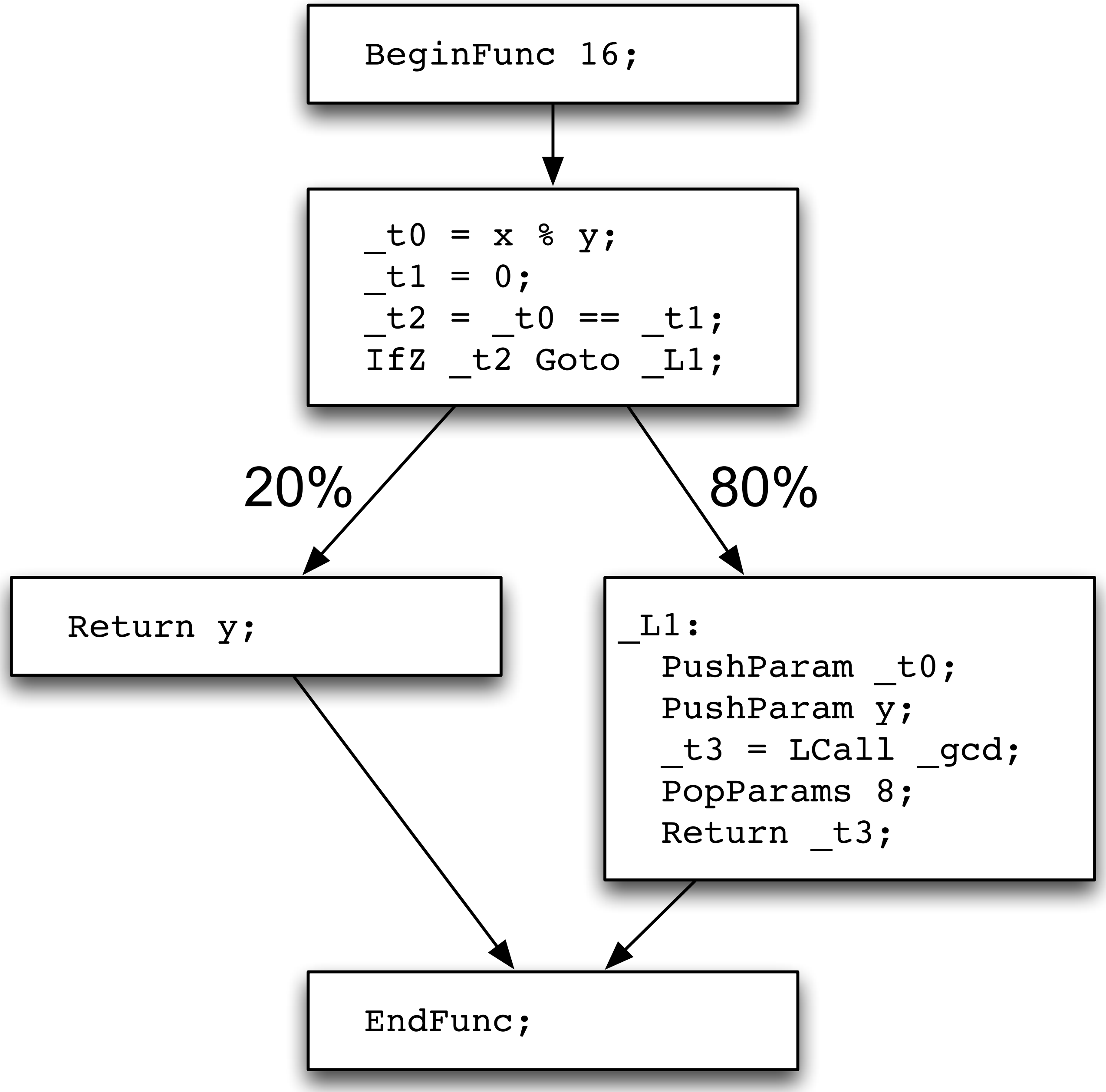
OUT={}

```
EndFunc;
```



Register Allocation (2/2)

- How to spill with minimum cost when there are only 2 registers, given the runtime profile?



```
BeginFunc 16;
```

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

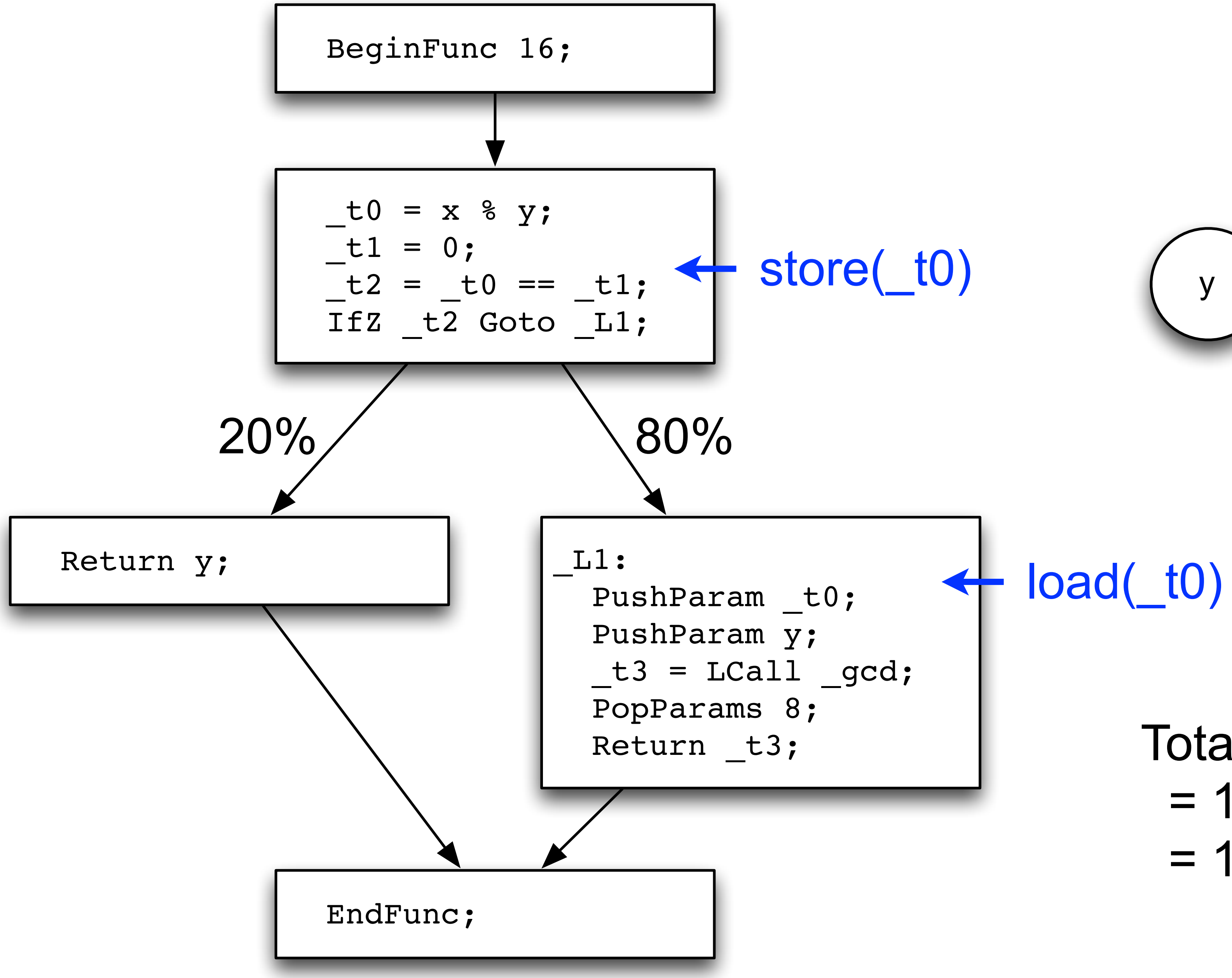
```
Return y;
```

```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

```
EndFunc;
```

Register Allocation (2/2)

- How to spill with minimum cost when there are only 2 registers, given the runtime profile?



```
BeginFunc 16;
```

```
_t0 = x % y;
_t1 = 0;
_t2 = _t0 == _t1;
IfZ _t2 Goto _L1;
```

store(_t0)

20%

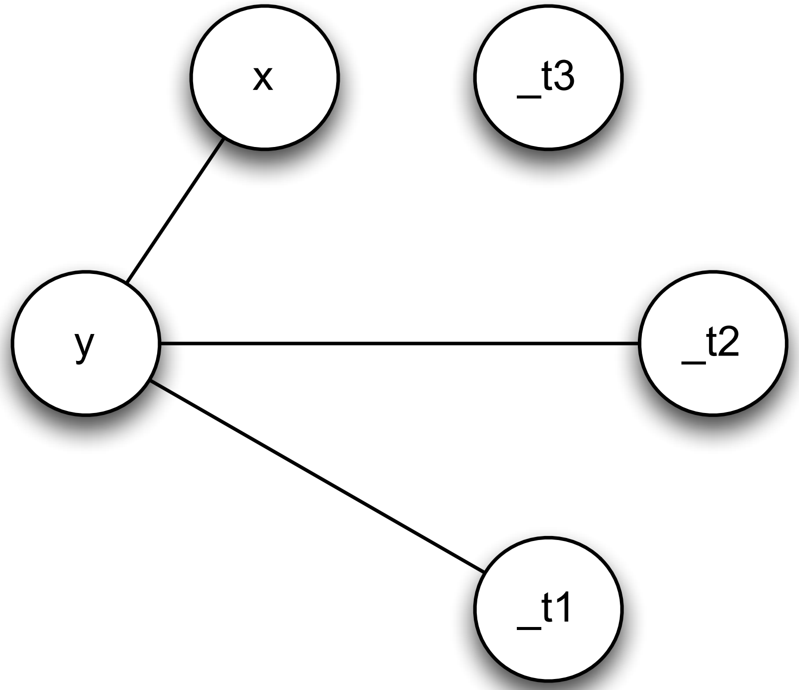
80%

```
Return y;
```

```
_L1:
  PushParam _t0;
  PushParam y;
  _t3 = LCall _gcd;
  PopParams 8;
  Return _t3;
```

load(_t0)

```
EndFunc;
```



Total cost
= 10 cycles + 10 cycles * 80%
= 18 cycles

SSA Conversion

- See the slides of the last discussion!
  - Compute dominator tree
  - Insert Phi nodes
  - Variable renaming

Thanks & good luck!