System-on-Chip Design
Analysis of Control Data Flow

Hao Zheng
Comp Sci & Eng
U of South Florida

Overview

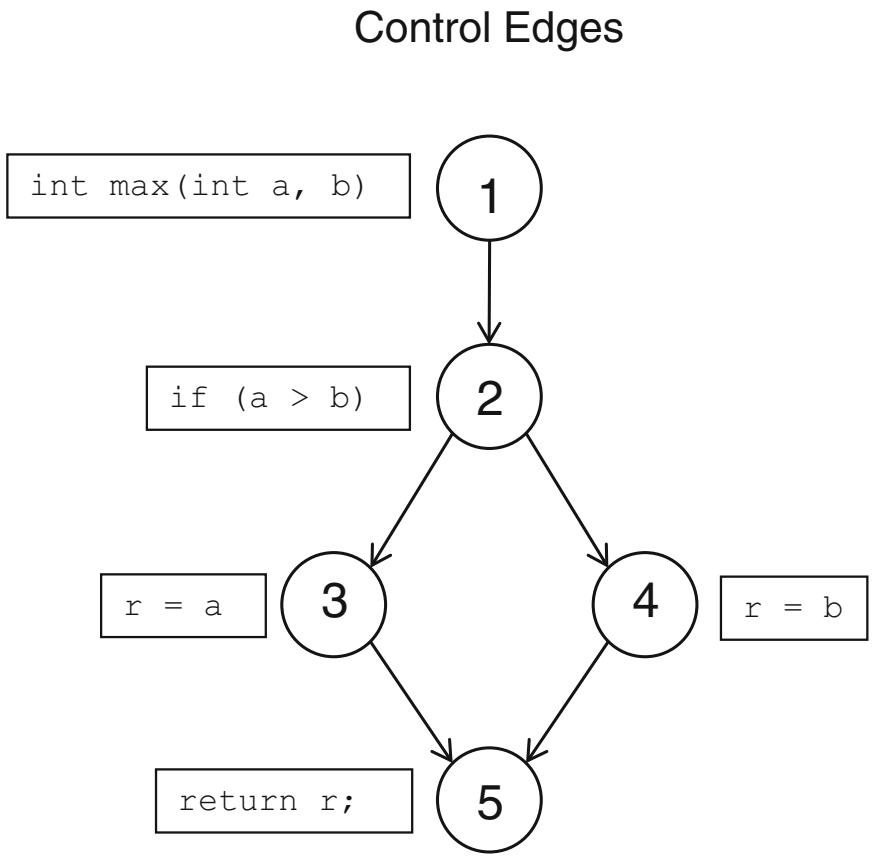
- DF models describe concurrent computation at a very high level
  - Each actor describes non-trivial computation.
- Each actor is often described in C.
  - Can be mapped to either HW or SW
- Will look at issues in mapping C to HW.

Data & Control Edges of C Programs

- C is used as a modeling as well as an implementation language.
- Mapping C programs to HW is hard.
  - HW is parallel while C is sequential.
  - need to understand the structure of C programs.
- Relations between operations in C programs
  - **Data edges**: data moved from one op. to another.
  - **Control edge**: no data xfer.

Control Flow Graph

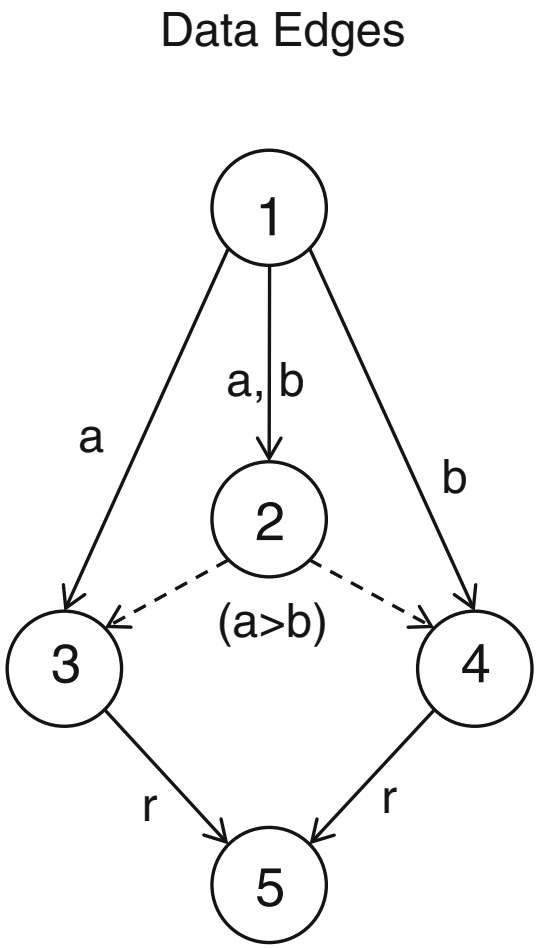
```
1   int x(a, b) {
      int r;
2     if (a > b)
3       r = a;
      else
4       r = b;
5     return r;
    }
```



Control edges are often labeled with conditions whose satisfaction dictates if a control can be taken.

Data Flow Graph

```
1   int max(int a, b) {
      int r;
2     if (a > b)
3       r = a;
      else
4       r = b;
5     return r;
    }
```



Data edges are labeled with variables upon which one operation depends on another

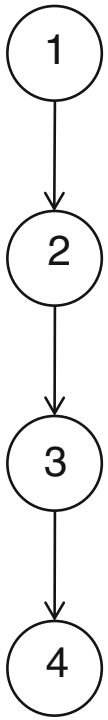
Implementing Control/Data Edges

- A data edge => flow of information
  - Must be implemented.
- A control edge => result of semantics of program language
  - Maybe ignore or changed if the behavior remains the same.

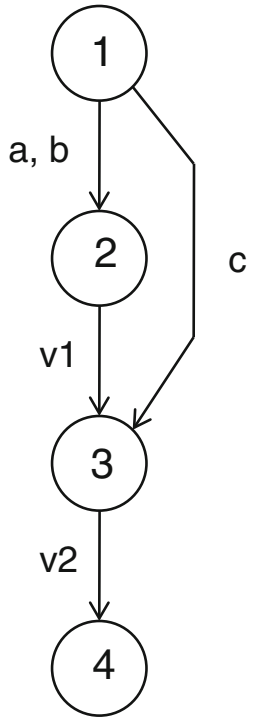
Implementing Control/Data Edges

```
int sum(int a, b, c) {
    int v1;
    v1 = a + b;   // op 2
    v2 = v1 + c;  // op 3
    return v2;
}
```

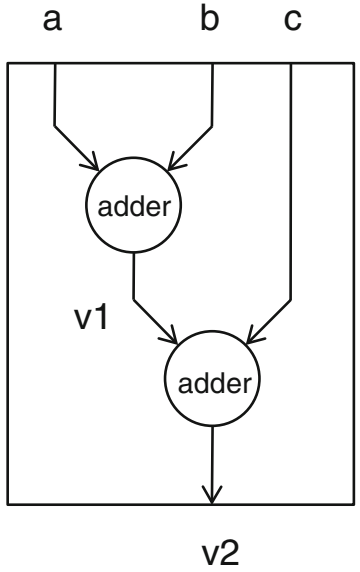
Control Edges



Data Edges



Hardware Implementation



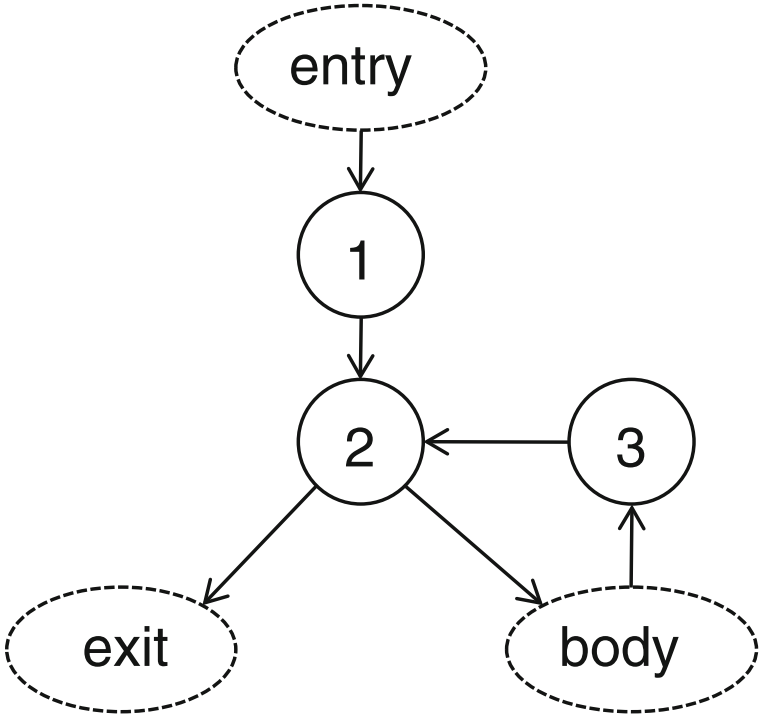
Control edges are meaningless as HW is parallel.

Control/Data Edges – Example

```
int sum(int a, b, c, d) {// op 1
    int v1;
    v1 = a + b;          // op 2
    v2 = c + d;          // op 3
    return v1 + v2;      // op 4
}
```

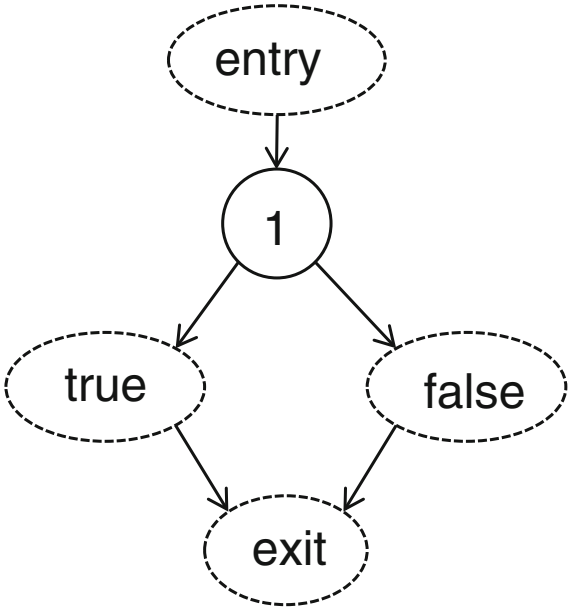
Basic Elements of CFG

```
      (1)      (2)      (3)
for (i=0; i < 20; i++) {
    // body of the loop
}
```



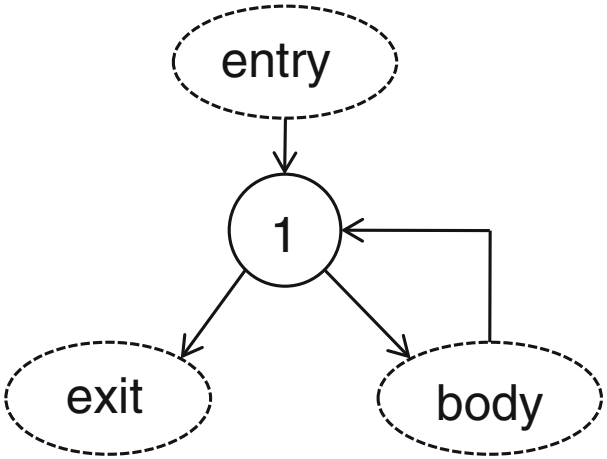
Construction of CFG

```
     (1)
if(a < b) {
    // true branch
} else {
    // false branch
}
```



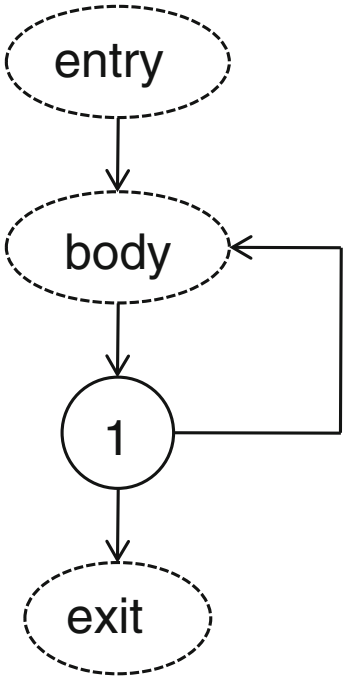
Construction of CFG

```
          1
while (a < b) {
    // loop body
}
```



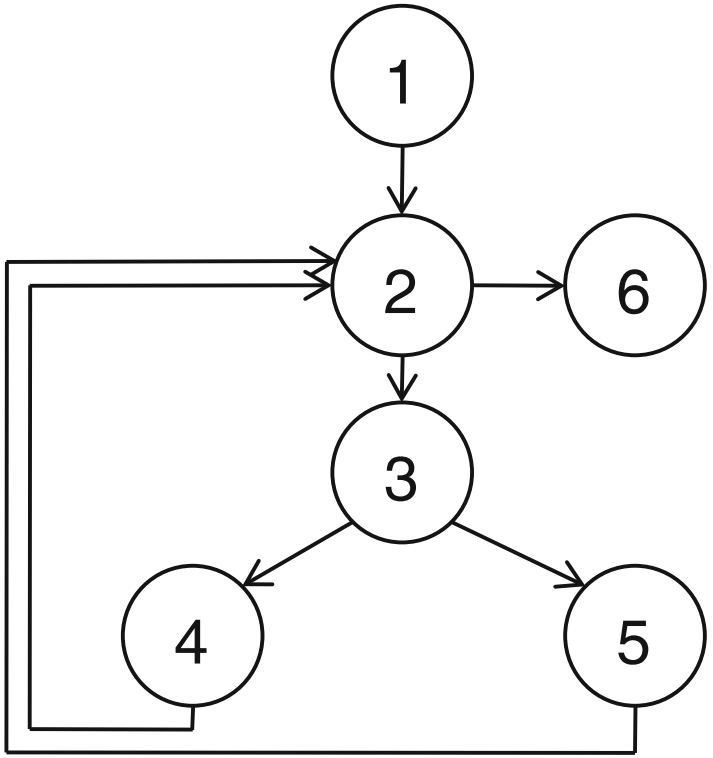
Construction of CFG

```
do {
   // loop body
} while (a<b)
```



Construction of CFG: GCD

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```

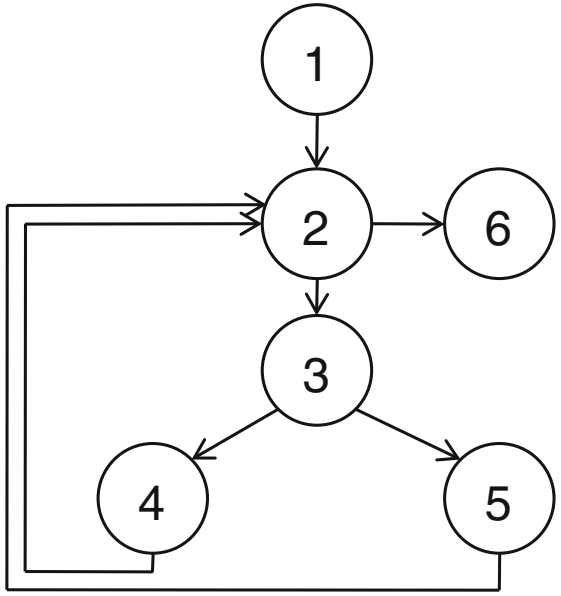


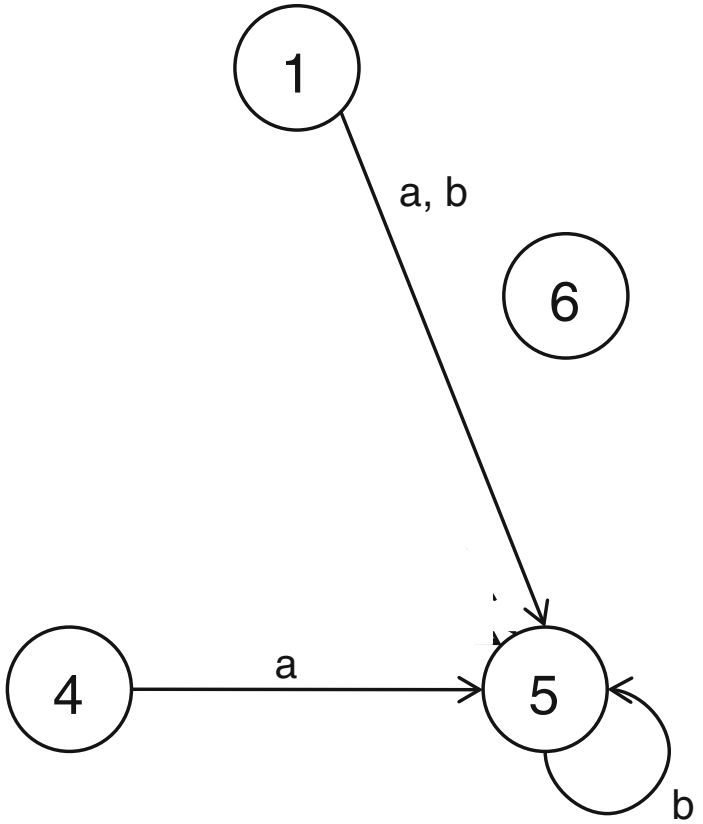
A ***control path*** in CFG corresponds to a sequence of executions of statements

Construction of DFG: GCD

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```

CFG

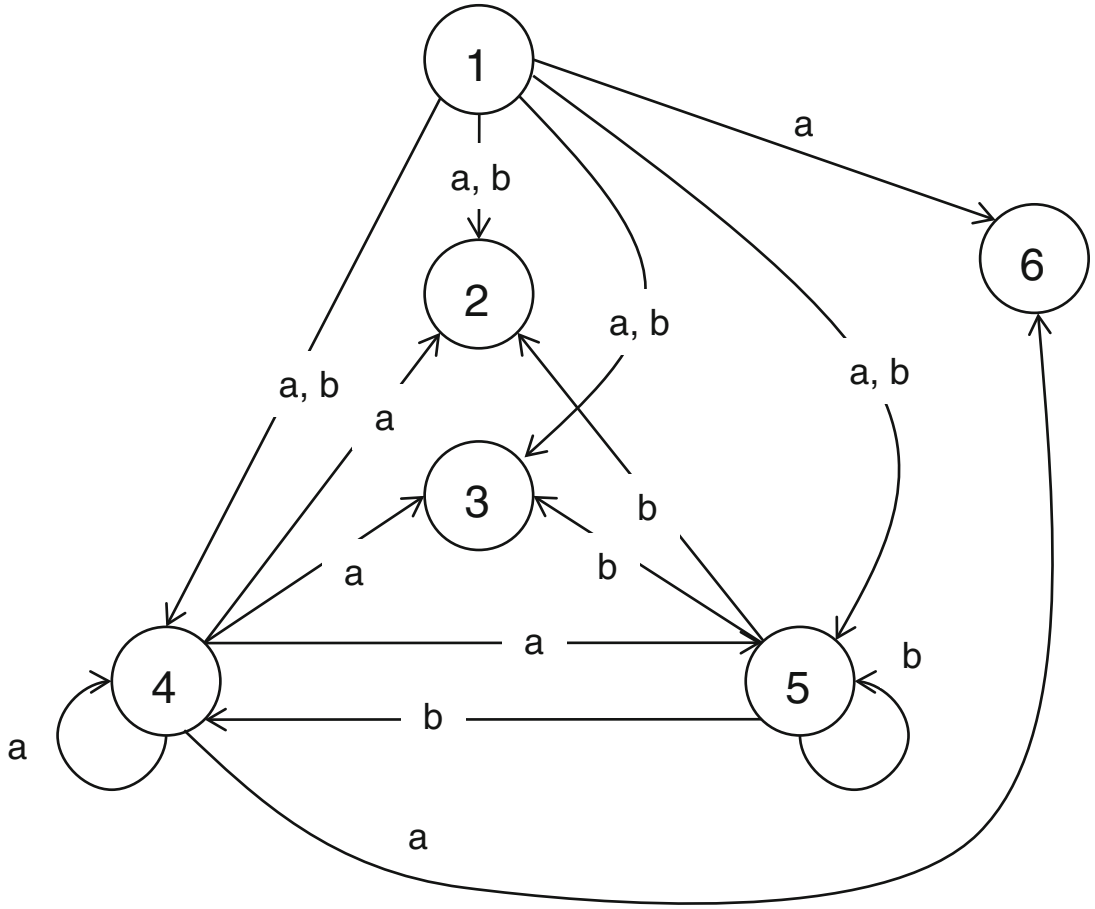




Partial DFG

Construction of DFG: GCD

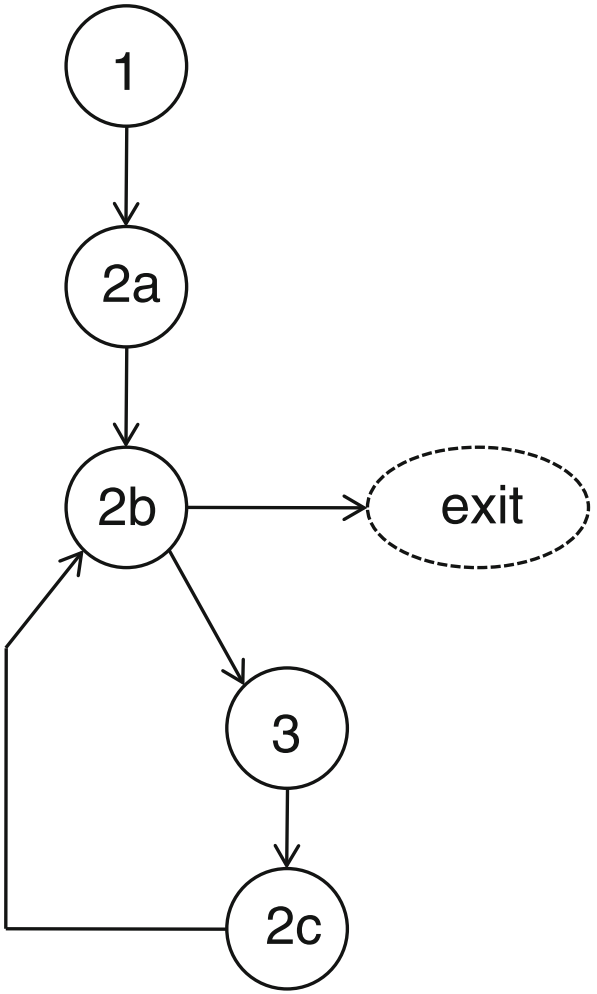
```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```



Construction of CFG/DFG

```
1: int L[3] = {10, 20, 30};
                 2a    2b   2c
2:    for (int i=1; i<3; i++)
3:      L[i] = L[i] + L[i-1];
```

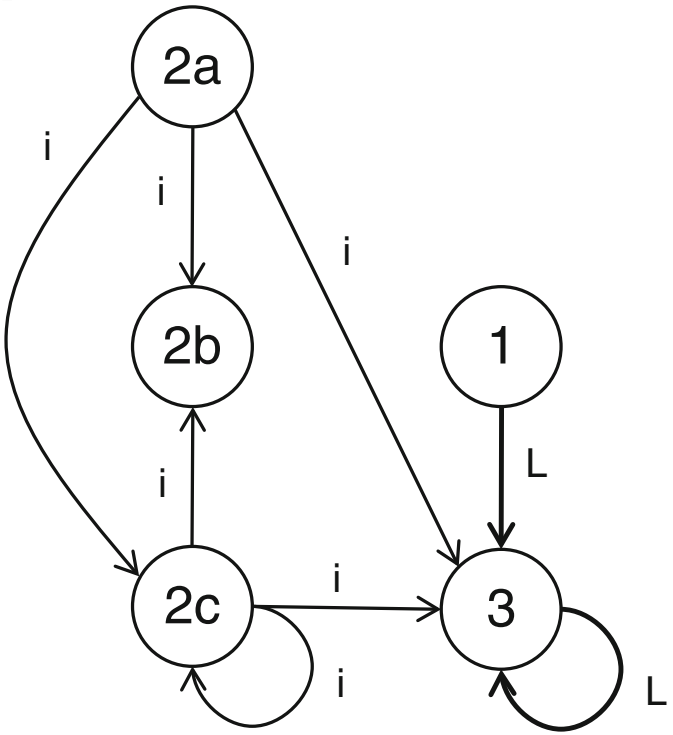
How to treat indexed variables in DFG construction?



CFG

Construction of CFG/DFG

a

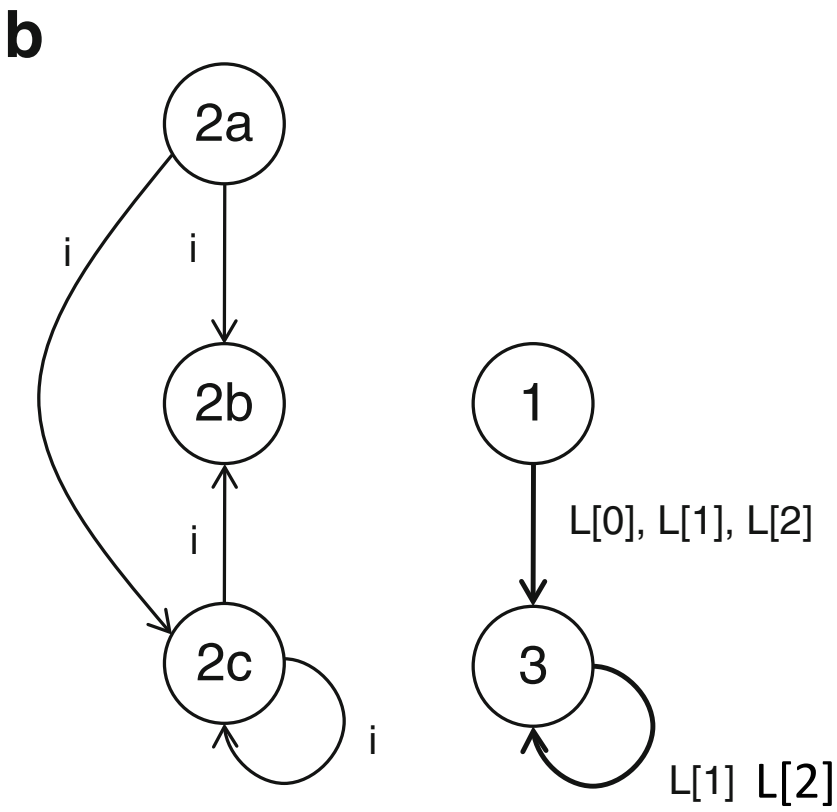



```
1: int L[3] = {10, 20, 30};
                 2a    2b    2c
2:    for (int i=1; i<3; i++)
3:      L[i] = L[i] + L[i-1];
```

Treat *L* as a single monolithic variable

Construction of CFG/DFG

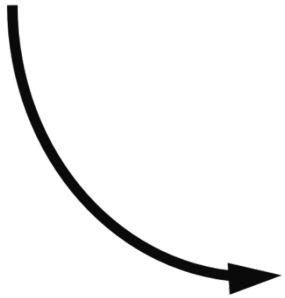
```
1: int L[3] = {10, 20, 30};
                 2a    2b   2c
2:    for (int i=1; i<3; i++)
3:      L[i] = L[i] + L[i-1];
```



Locations of *L* are treated individually

DFG Analysis – Loop Unrolling

```
1: int L[3] = {10, 20, 30};
                 2a    2b    2c
2:    for (int i=1; i<3; i++)
3:      L[i] = L[i] + L[i-1];
```



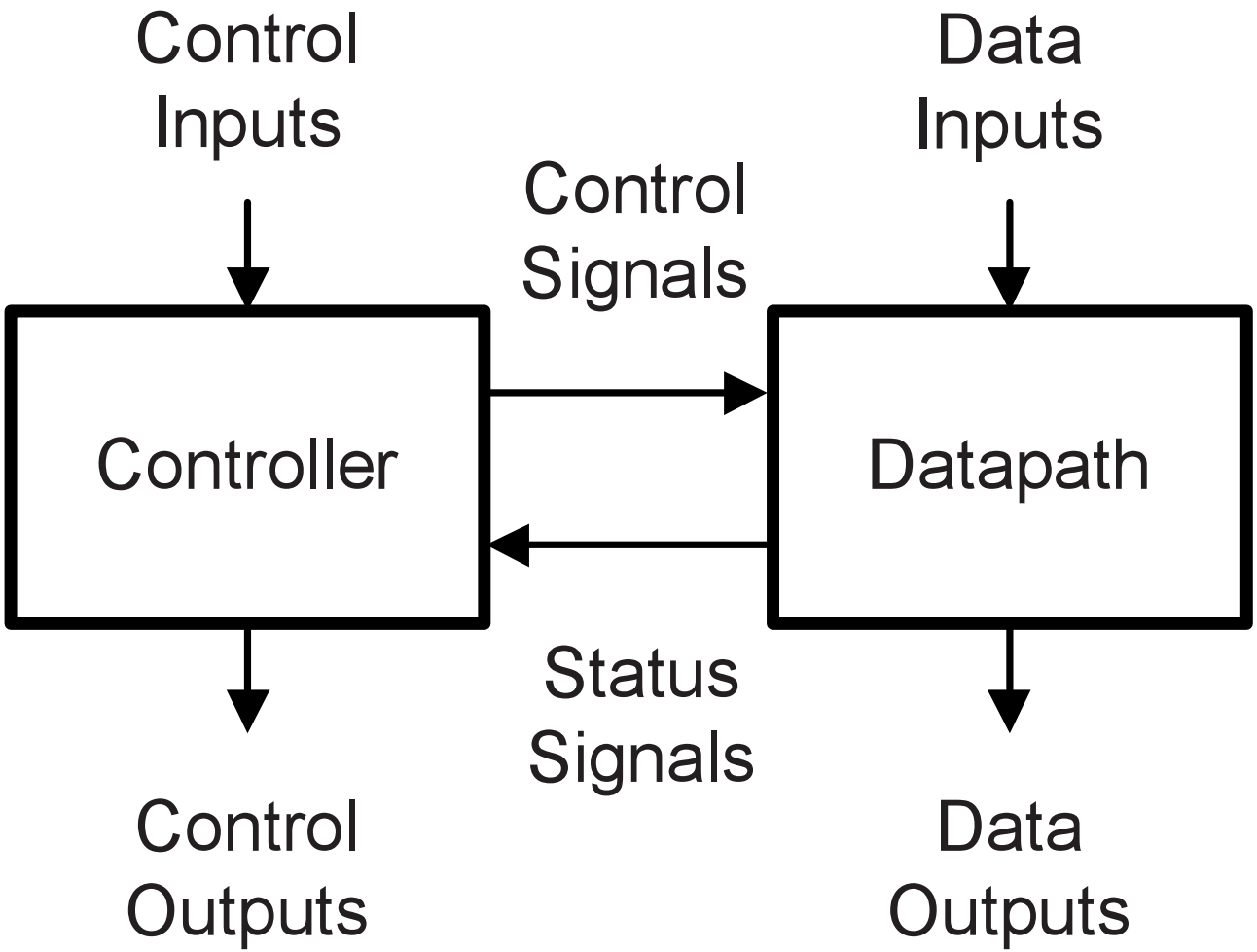
```
int L[3] = {10, 20, 30};

L[1] = L[1] + L[0];
L[2] = L[2] + L[1];
```

Translating C to HW

- Assumptions:
  - Scalar C programs – no pointers and arrays
  - Implement each statement in a clock cycle.
- Basic Idea
  - Construct CFG and DFG
  - CFG => controller (control edge -> control sig.)
  - DFG => datapath (data edges -> comp conn.)
- Not very efficient – exist many optimization opportunities

HW RTL Architecture

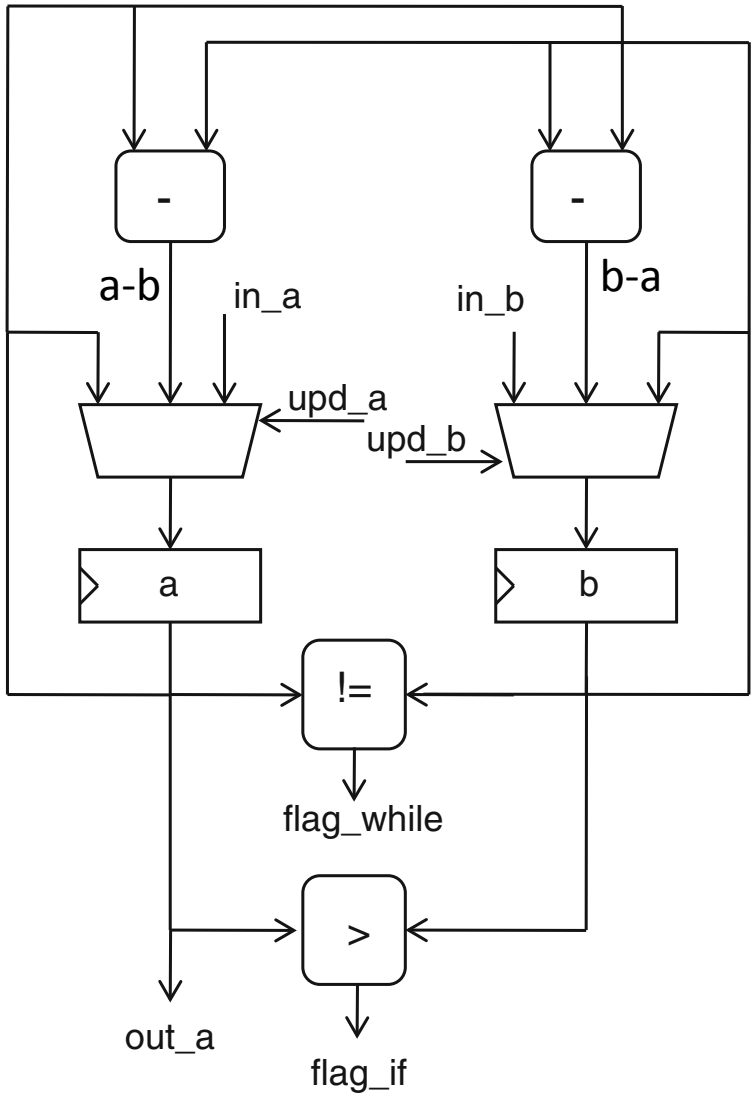


Translating C to HW: Building Datapath

- Each variable => a register
- MUX is used if a variable is updated in multiple statements.
- Each expression => a combinational logic
  - Conditional expressions => flags to controller
- Datapath circuits and registers are connected according to data edges in DFG.

Translating C to HW: Building Datapath

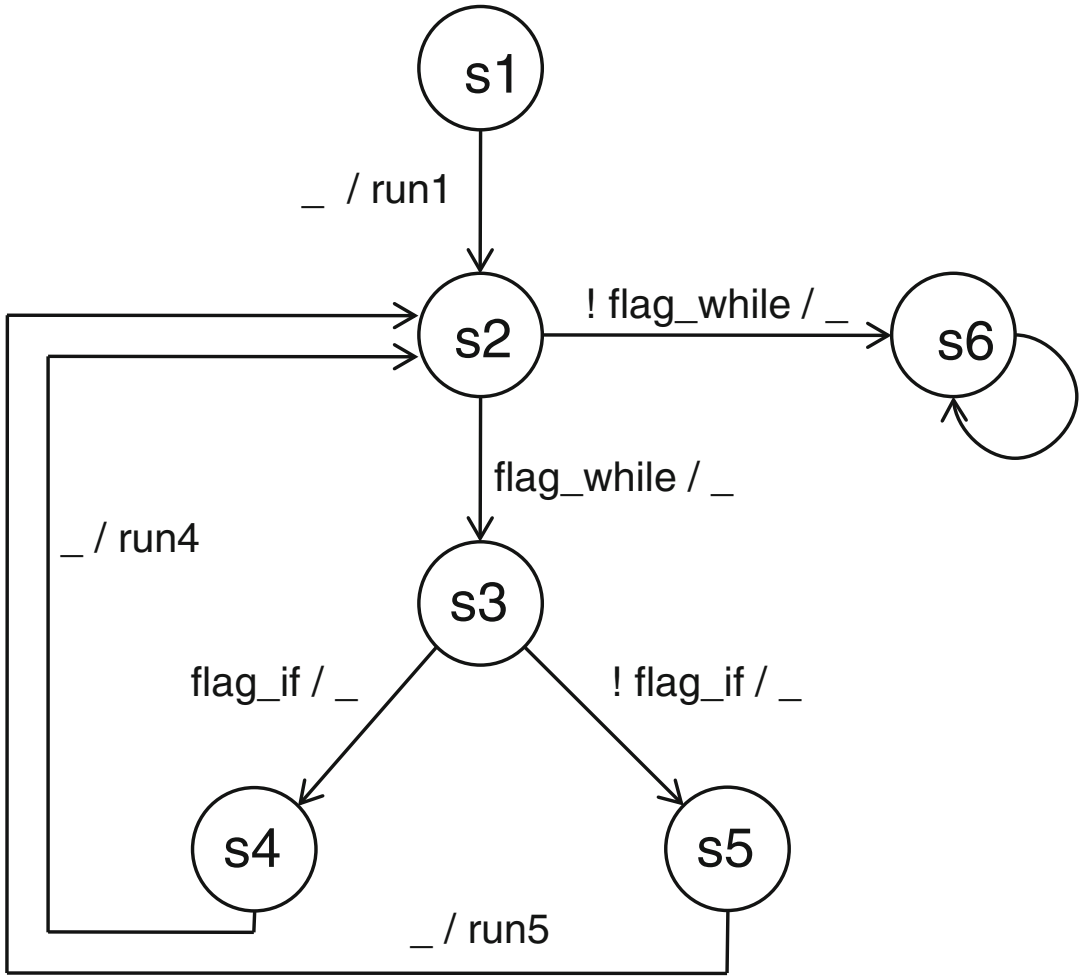
```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```



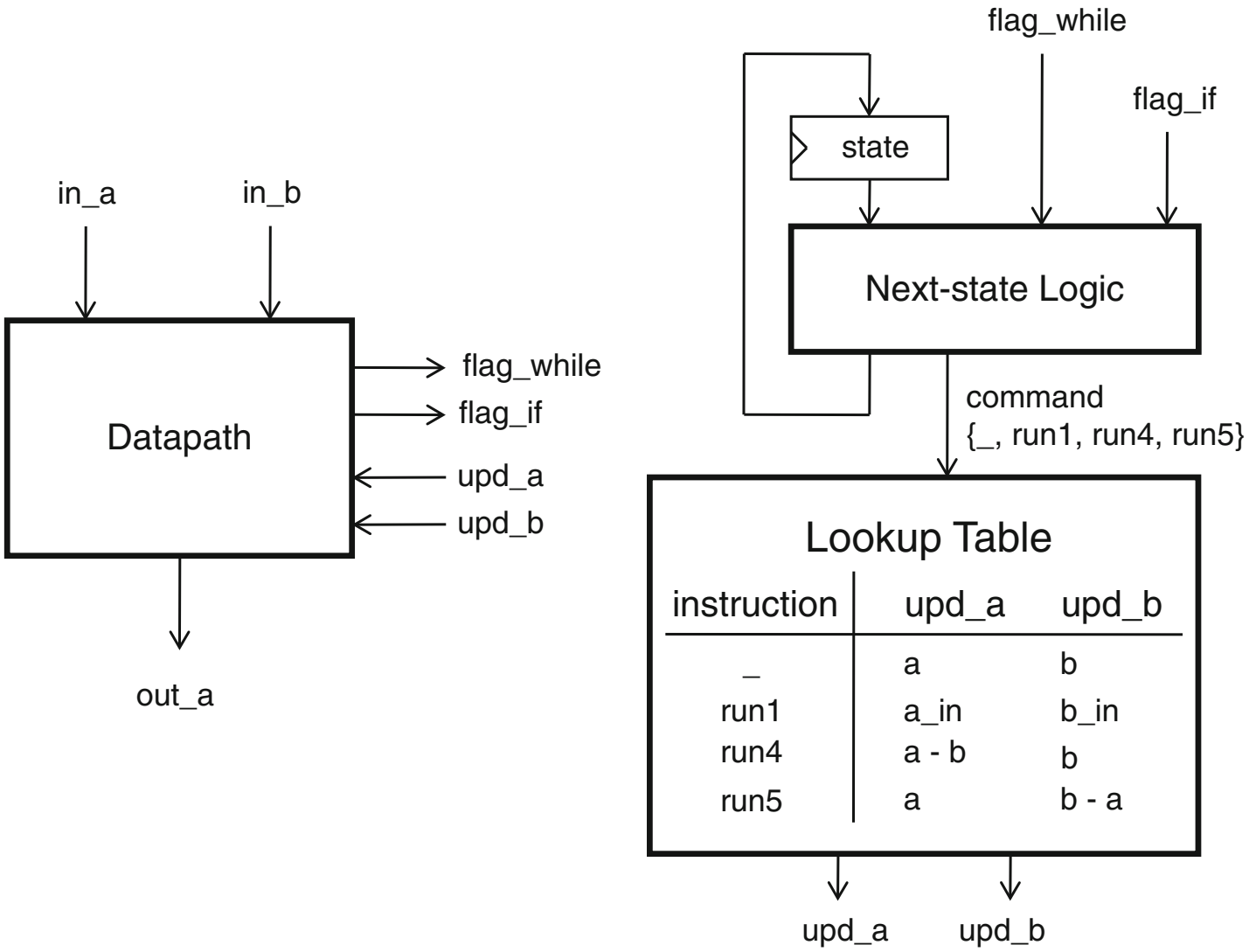
Translating C to HW: Building Controller

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```

Label CFG edges with flags from datapath and actions that DP should perform, and implement CFG as FSM.



Translating C to HW: Building Controller



Limitations

- Each variable mapped to a register.
- A functional unit is allocated to every operator.
- Performance bottleneck as a single statement is executed in a single clock cycle.
  - Processor is already doing this.
  - Can multiple statements be executed in a cycle?

Translating C to HW: Single-Assignment Form

- Each variable is assigned exactly once.
- To improve efficiency of the HW implementation.

```
a = a + 1;          a2 = a1 + 1;
a = a * 3;    ⟹    a3 = a2 * 3;
a = a – 2;          a4 = a3 – 2;
```

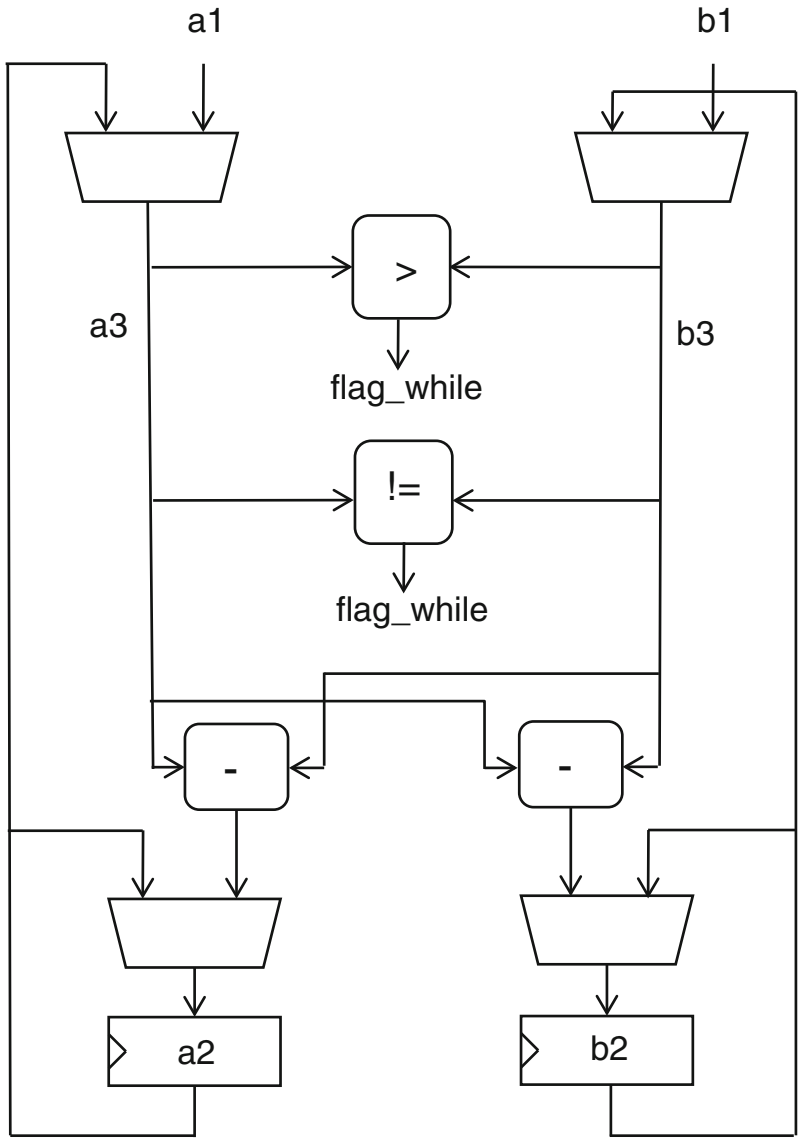
Translating C to HW: Single-Assignment Form

```
int gcd(int a, b) {
  while (a != b) {
    if (a > b)
      a = a – b;
    else
      b = b – a; }
  return a; }
```

```
int gcd(int a1, b1) {
  while (merge(a1, a2) != merge(b1, b2)) {
    a3 = merge(a1, a2);
    b3 = merge(b1, b2);
    if (a3 > b3)
      a2 = a3 – b3;
    else
      b2 = b3 – a3; }
  return a; }
```

Translating C to HW: Single-Assignment Form

```
int gcd(int a1, b1) {
  while (merge(a1, a2) != merge(b1, b2)) {
      a3 = merge(a1, a2);
      b3 = merge(b1, b2);
      if (a3 > b3)
        a2 = a3 – b3;
      else
        b2 = b3 – a3; }
  return a; }
```



Reading Guide

- Chapter 4, the *CoDesign* book.