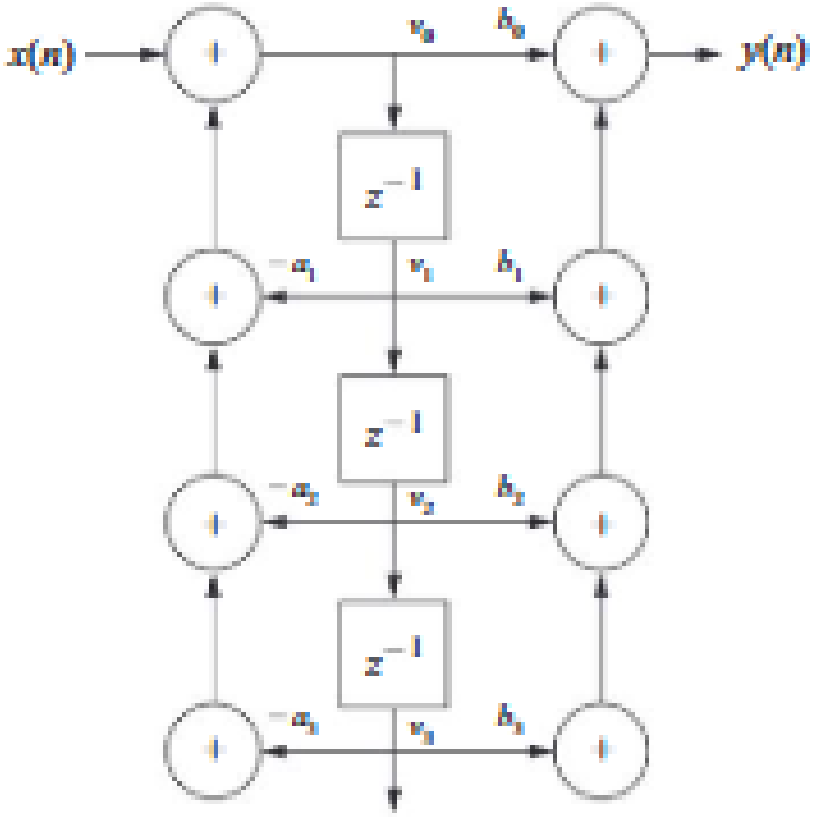
IIR filter example



Direct form II IIR filter

Pseudo code

```
// Define filter coefficients (example for a 3rd-order filter)
a1, a2, a3 = filter_coefficients_for_denominator  // Denominator coefficients
b0, b1, b2, b3 = filter_coefficients_for_numerator  // Numerator coefficients

// Initialize delay elements (previous input and output values)
x_prev_1 = 0  // Previous input value x[n-1]
x_prev_2 = 0  // Previous input value x[n-2]
x_prev_3 = 0  // Previous input value x[n-3]
y_prev_1 = 0  // Previous output value y[n-1]
y_prev_2 = 0  // Previous output value y[n-2]
y_prev_3 = 0  // Previous output value y[n-3]

// Initialize input stream (could be a list or stream of data)
input_stream = get_input_stream()  // Function to provide the input stream
output_stream = []  // To store or process the output
```

Pseudo code continue

```
// Start processing the input stream
while there are more samples in input_stream:
    x_n = next(input_stream)  // Get the next input sample x[n]

    // Calculate the new output y[n]
    y_n = b0 * x_n + b1 * x_prev_1 + b2 * x_prev_2 + b3 * x_prev_3
              - a1 * y_prev_1 - a2 * y_prev_2 - a3 * y_prev_3

    // Store or output the result
    output_stream.append(y_n)  // Save output for later processing

    // Update previous values for the next iteration
    x_prev_3 = x_prev_2  // Shift x[n-2] to x[n-3]
    x_prev_2 = x_prev_1  // Shift x[n-1] to x[n-2]
    x_prev_1 = x_n      // Current input x[n] becomes x[n-1]

    y_prev_3 = y_prev_2  // Shift y[n-2] to y[n-3]
    y_prev_2 = y_prev_1  // Shift y[n-1] to y[n-2]
    y_prev_1 = y_n      // Current output y[n] becomes y[n-1]
```

Models of programs

Source code is not a good representation for programs:

- Clumsy;
- Leaves much information implicit.

Compilers derive intermediate representations to manipulate and optimize the program.

A lot of details are present in the source code. Sometimes it is not the format that can help in optimizing the machine code generation.
Source code can come in different language from assembly code to C, Java, Python at various levels of abstractions. We need a uniform level of program models that can address all these abstractions.

Data flow graph

DFG: data flow graph.

- Does not represent control.
- Models basic block: code with no entry or exit.
- Describes the minimal ordering requirements on operations.

- It models the sequential flow of the program code. Does not account for the repetition and conditional structures seen in the code.

Single assignment form

```
x = a + b;
y = c - d;
z = x * y;
y = b + d;
```

original basic block

```
x = a + b;
y = c - d;
z = x * y;
y1 = b + d;
```

single assignment form

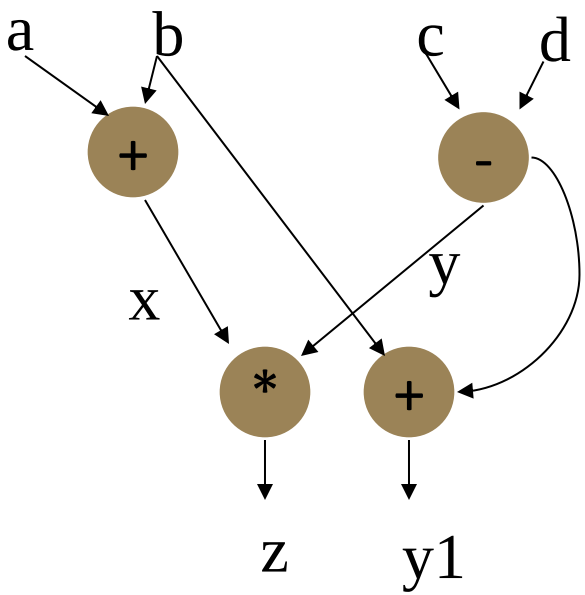
A single assignment form avoids the unnecessary generation of loops or cyclic graphs when the code is converted into intermediate form for optimization.

Data flow graph

An example of how the DFG is developed from the sequential code. Single assignment form of variables result in elimination of unnecessary cycles in the DFG.

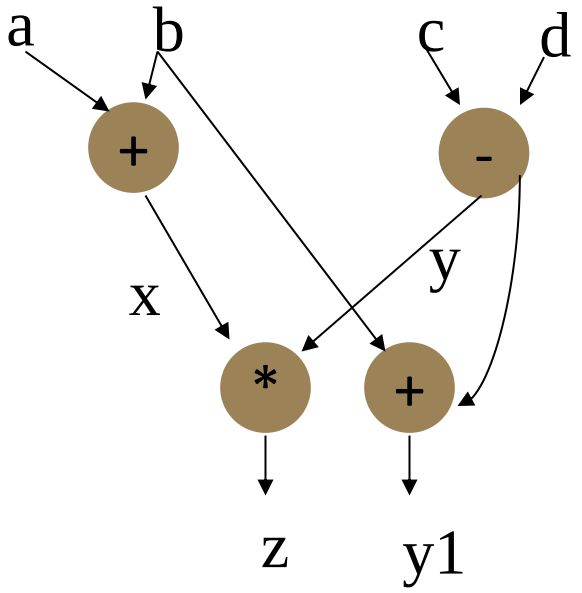
```
x = a + b;
y = c - d;
z = x * y;
y1 = b + d;
```

single assignment form



DFG

DFGs and partial orders



Partial order:

a+b, c-d;

b+d, x*y

Can do pairs of operations in any order.

Partial ordering helps in identifying which parts of the code can be run independently as their dependences are eliminated. This facilitates the compiler to take advantage of the hardware resources of the target processing system, For example running the instruction in parallel on available execution units and/or processing cores.

Control-data flow graph

CDFG: represents control and data.

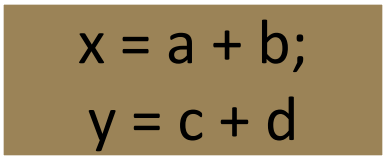
Uses **data flow graphs** as components.

Two types of nodes:

- decision;
- dataflow.

Expands DFG to account for decisions. Similar to flow chart or state chart modeling.

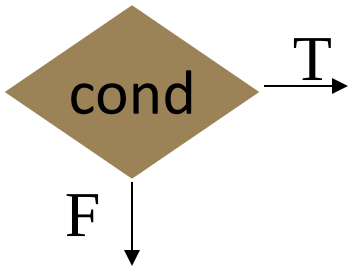
Encapsulates a data flow graph:

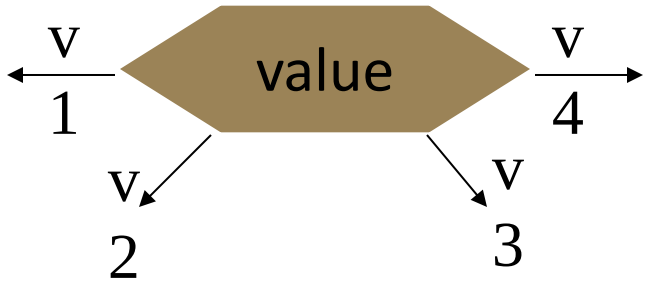


Data flow node

Write operations in basic block form for simplicity.

Control

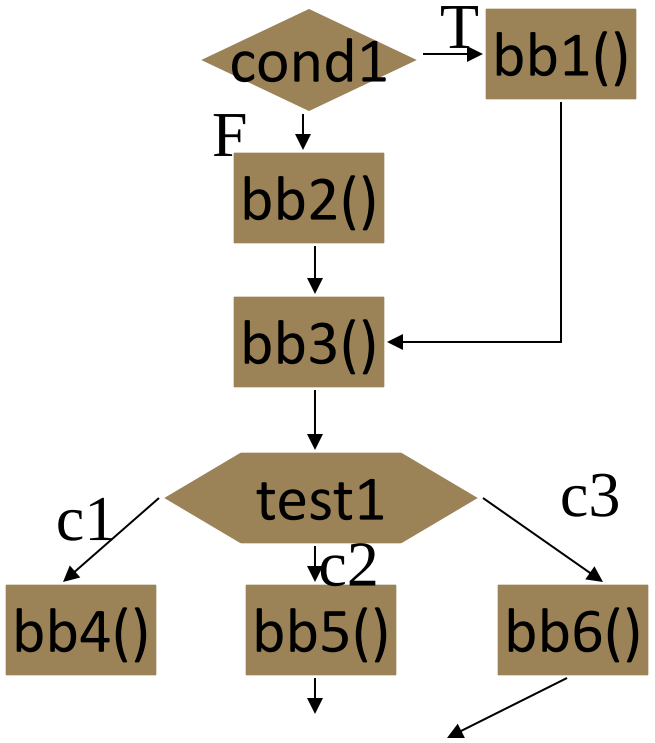




Equivalent forms

CDFG example

```
if (cond1) bb1();
 else bb2();
bb3();
switch (test1) {
  case c1: bb4(); break;
  case c2: bb5(); break;
  case c3: bb6(); break;
}
```



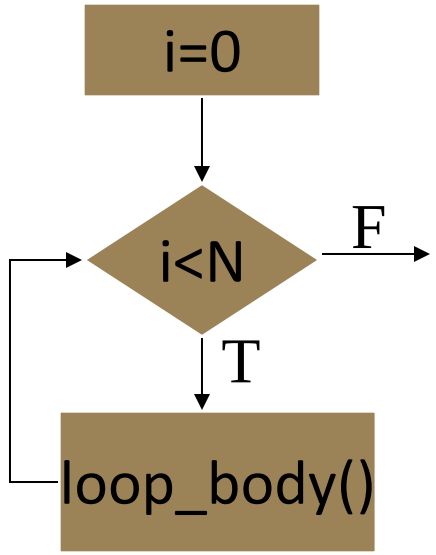
for loop

```
for (i=0; i<N; i++)
  loop_body();
```

for loop

Equivalent while loop:

```
i=0;
while (i<N) {
  loop_body(); i++; }
```



Control Data Flow Graph