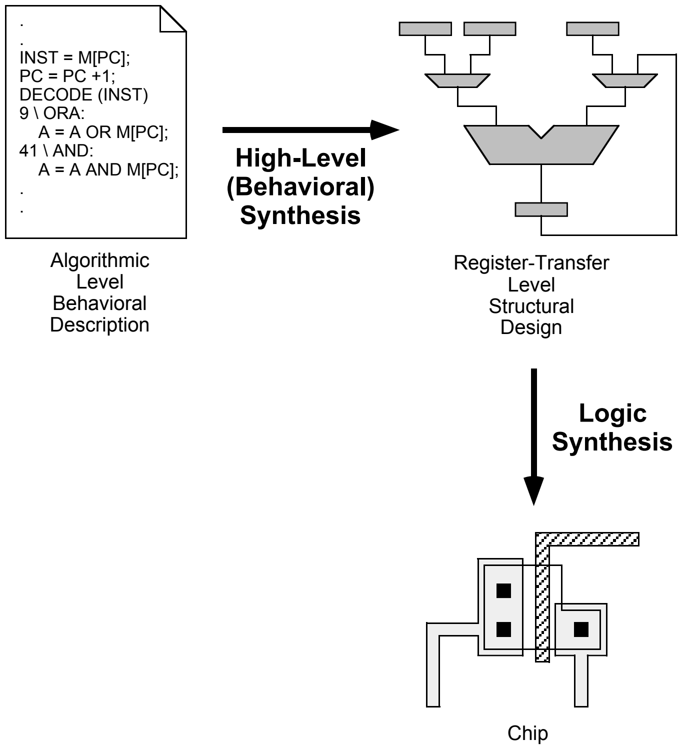
Automated Synthesis of a Digital System (Electronic Design Automation — EDA)



Behavioral Description of a *Very* Small Microprocessor

```
run {main} := BEGIN
  repeat BEGIN
    IR = M[PC];
    PC = PC + 1;
    DECODE IR => BEGIN
          9\ORA := BEGIN
                A = A OR M[PC];
                PC = PC + 1;
                END,
          41\AND := BEGIN
                A = A AND M[PC];
                PC = PC + 1;
                END
          END
    END
END
```


A Behavioral Description (Part of a Signal Processor, Perhaps)

A differential equation:

$$\frac{d^2y}{dx^2}+5\frac{dy}{dx}x+3y=0$$

The code to solve this equation could be written as:

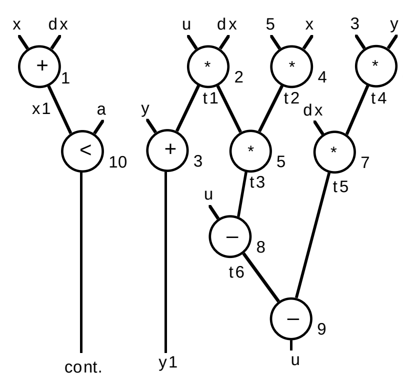
```
repeat
    x1 = x + dx;
    y1 = y + (u*dx);
    u = u – 5*x*(u*dx)
    – 3*y*dx;
    x = x1;  y = y1;
while (x1 < a)
```

That code might parse as:

1. x1 = x + dx
2. t1 = u * dx
3. y1 = y + t1
4. t2 = 5 * x
5. t3 = t2 * t1
6. t4 = 3 * y
7. t5 = t4 * dx
8. t6 = u – t3
9. u = t6 – t5
10. x1 < a

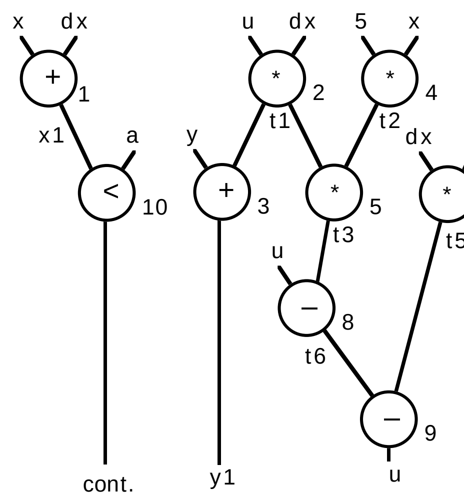
Building a Data Flow Graph (DFG)

1. x1 = x + dx
2. t1 = u * dx
3. y1 = y + t1
4. t2 = 5 * x
5. t3 = t2 * t1
6. t4 = 3 * y
7. t5 = t4 * dx
8. t6 = u – t3
9. u = t6 – t5
10. x1 < a



Scheduling

- ***Scheduling*** is the problem of determining the ***control step***, or state, in which each operation will execute



As-Soon-As-Possible (ASAP) Scheduling

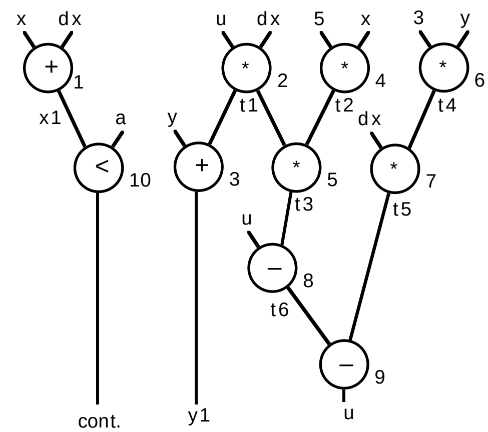
for each operation o_i

if o_i has no immediate predecessors

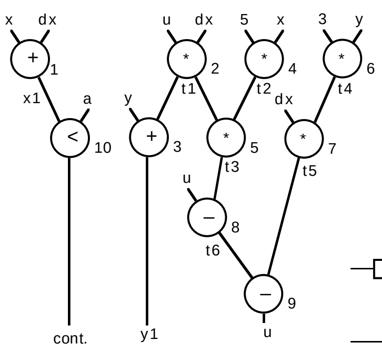
assign o_i to cstep 1

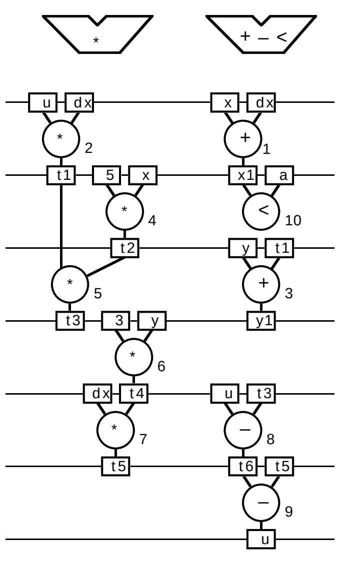
else

assign o_i to (maximum cstep of any of o_i's predecessors) + 1

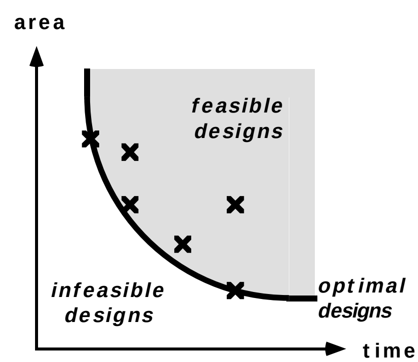


Another Possible Schedule (One Multiplier, One ALU (+,–,<))





The Design Space



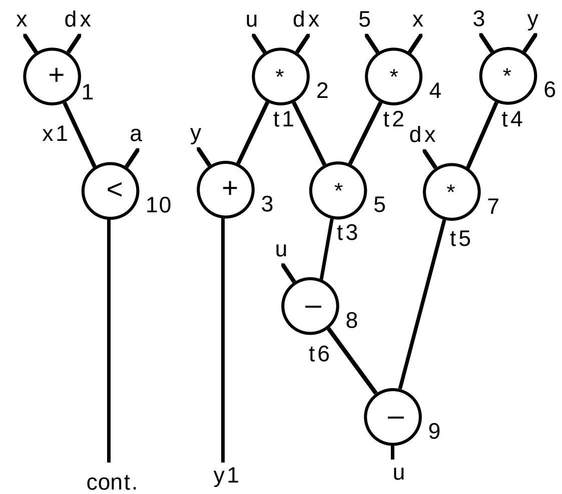
- For optimal designs, there is a tradeoff between:
  - *time* (schedule length), and
  - *area* (ideally total area, but usually simplified to functional unit area)
- We'd prefer to find *optimal* designs, but a heuristic (such as ASAP scheduling) only guarantees *feasible* designs

Three Scheduling Problems

- ***Scheduling*** is the problem of determining the ***control step***, or state, in which each operation will execute
- The scheduling problem is usually specified in one of three ways, depending on the desired goal:
  - ***Time-Constrained Scheduling*** (TCS) — for a fixed schedule length, minimize the number of resources (functional units)
  - ***Resource -Constrained Scheduling*** (RCS) — for a fixed number of resources (functional units), minimize the schedule length
  - ***Time- and Resource-Constrained Scheduling*** (TRCS) — for a fixed schedule length, and a fixed number of resources, find a feasible (or optimal) schedule

Example of Resource-Constrained Scheduling

- Schedule this DFG, assuming there are only 2 multipliers and 2 ALUs (+,–,<) available



- How could the ASAP algorithm be modified to solve this problem?

List Scheduling (To Solve the RCS Problem)

evaluate the priority of each operation

current-cstep = 1

while there are unscheduled operations

  current-cstep = current-cstep + 1

  place data-ready operations into the ready list

  sort the ready list in order of priority

  while there are data-ready operations in the ready list that meet the resource constraints

    choose the highest priority data-ready operation o_i from the ready list

    assign o_i to current-cstep

Notes on List Scheduling

- Solves the RCS problem
- Basic operation differs from ASAP:
  - ASAP — processes <u>operations</u> in a fixed order
  - List Scheduling — processes <u>csteps</u> in a fixed order
    - Fill one cstep, then go on to the next
- Uses a *ready list* to keep track of *data-ready operations* — those unscheduled operations that can be scheduled into the current cstep without violating:

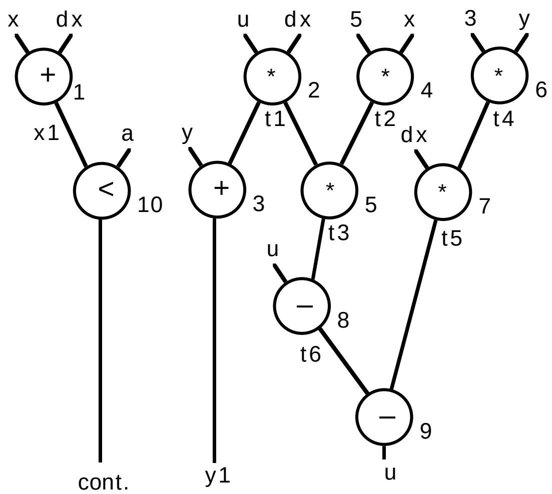
  precedence constraints (data dependencies)

  resource constraints

  - Pick operations from this ready list, and schedule them into the current cstep until it is full (i.e., other operations would violate the resource constraints)

List Scheduling Example

- Use list scheduling to schedule this DFG with a resource constraint of 2 multipliers, and 2 ALUs (+,–, <)

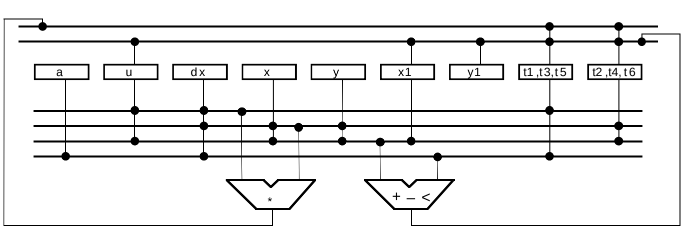


Notes on List Scheduling (cont.)

- As each cstep is processed, the data-ready operations are sorted according to *priority*
  - Data-ready operations are then removed from the ready list and scheduled into the current cstep based on their priority
- Common priority functions, giving increased priority to operations with:
  - Lower *mobility* — length of operation's *schedule interval* (ALAP – ALAP + 1)
  - Longer path to end of graph
  - Greater number of immediate successors
- We will use:
  - Primary priority function: highest priority to operations with lower mobility
  - Secondary priority function: highest priority to operations parsed earlier

Datapath Synthesis

- ***Datapath synthesis*** is the problem of:
  - Assigning operations to functional units (ALUs, adders, etc.)
  - Assigning values to storage elements (registers, etc.)
  - Allocating interconnections (multiplexors, buses, wires, etc.)
- A possible datapath for the 1 multiplier / 1 ALU schedule:

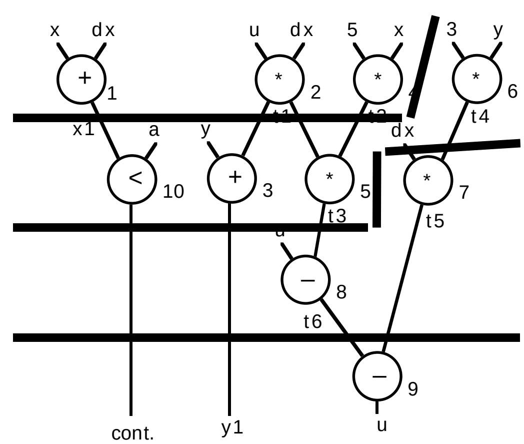


Constructive Datapath Synthesis

for each operation o_i

consider all possible bindings for o_i

select the binding that results in the smallest increase in cost

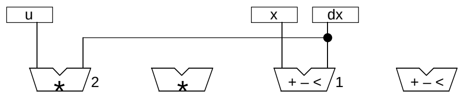


Sample costs:

New register = 100 New mux = 30

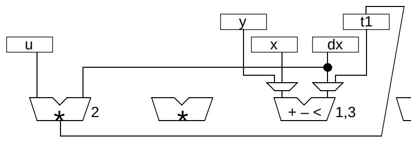
New wire / const = 5 New mux input = 20

Synthesizing a Datapath



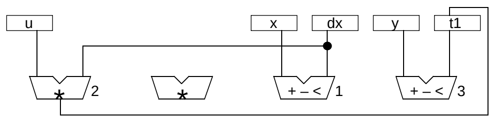
Assign operation 1
to an arbitrary ALU
(here, the one on the left)

Assign operation 2
to an arbitrary multiplier
(here, the one on the left)



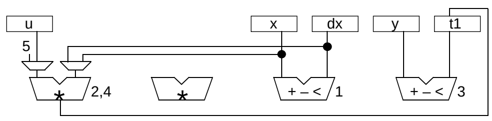
One alternative
for operation 3

3 on left ALU
cost = 2 reg, 2 mux, 5 wires
= 200 + 60 + 25
= 285



Another alternative
for operation 3

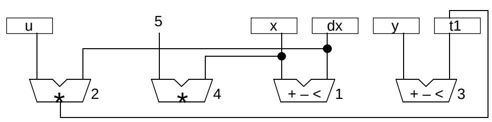
3 on right ALU
cost = 2 reg, 3 wires
= 200 + 15
= 215



One alternative
for operation 4

Not allowed!!

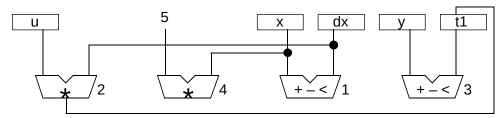
Same mult can't do
two operations (2 & 4)
in same control step!

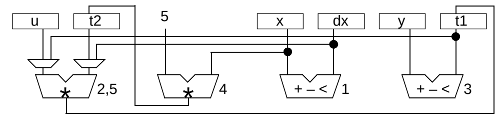


Another alternative
for operation 4

3 on right mult
cost = 2 wires/consts
= 10

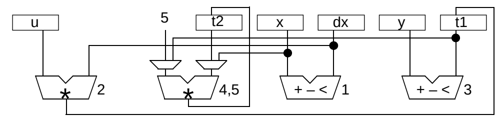
Synthesizing a Datapath (cont.)





One alternative
for operation 5

5 on left mult
cost = 1 reg, 2 mux, 5 wires
= 100 + 60 + 25
= 185



Another alternative
for operation 5

5 on right mult
cost = 1 reg, 2 mux, 5 wires
= 100 + 60 + 25
= 185

continue
this
process

- •
- •
- •