High-Level Synthesis

Franco Fummi

UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

Version 1.1

Contents

- High-Level (Architectural) Synthesis
  - complete example
  - behavioral synthesis
  - graphs
- Scheduling: Heuristic Algorithms
  - DFG
  - ASAP / ALAP
- Allocation:
  - resource binding
  - sharing

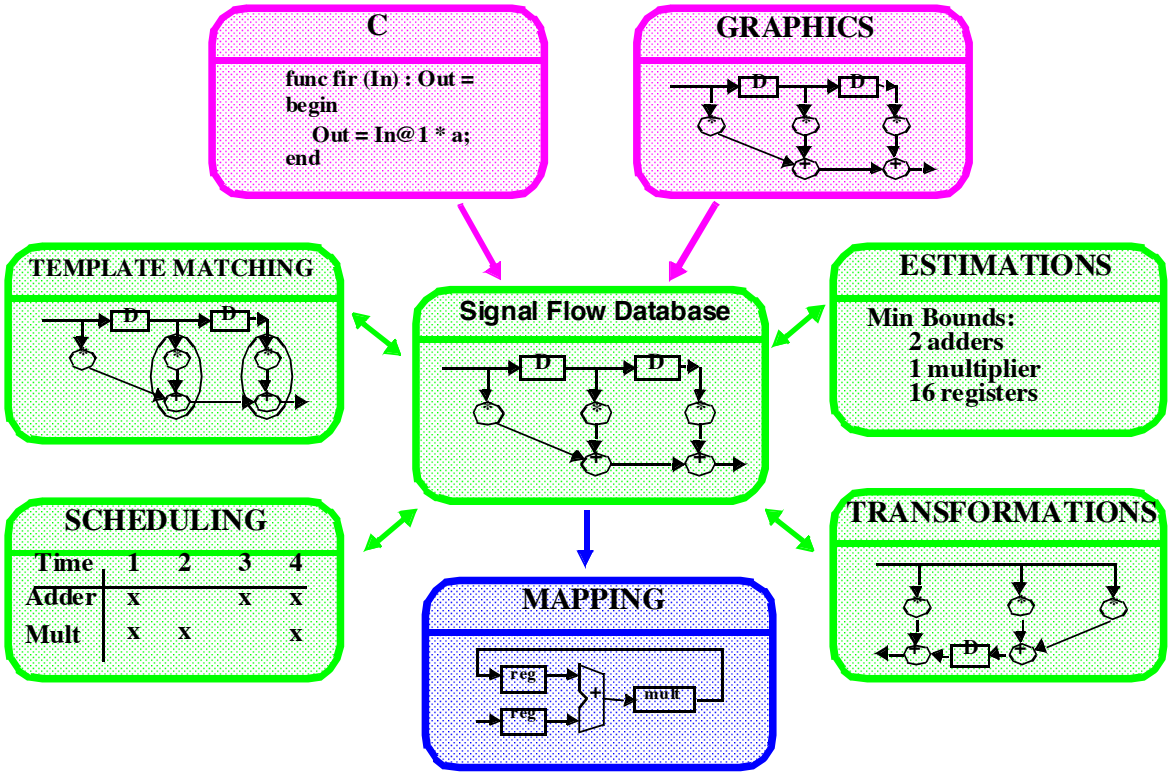
High-Level (Architectural) Synthesis

High Level Synthesis (HLS)

- The process of converting a high-level description of a design to RTL
  - Input:
    - High-level languages (C, System C, system Verilog)
    - Behavioral hardware description languages (Verilog, VHDL)
    - Structural HDLs (VHDL, Verilog)
    - State diagrams / logic networks
  - Tools:
    - Parser
    - Library of modules
  - Constraints:
    - Resource constraints (no. of modules of a certain type)
    - Timing constraints (Latency, delay, clock cycle)
  - Output:
    - Operation scheduling (time) and binding (resource)
    - Control generation and RTL architecture

TLM => No automatic synthesis existing up to now

High Level Synthesis



Example – Digital Filter design

- A second-order digital filter => The starting point is the algorithm

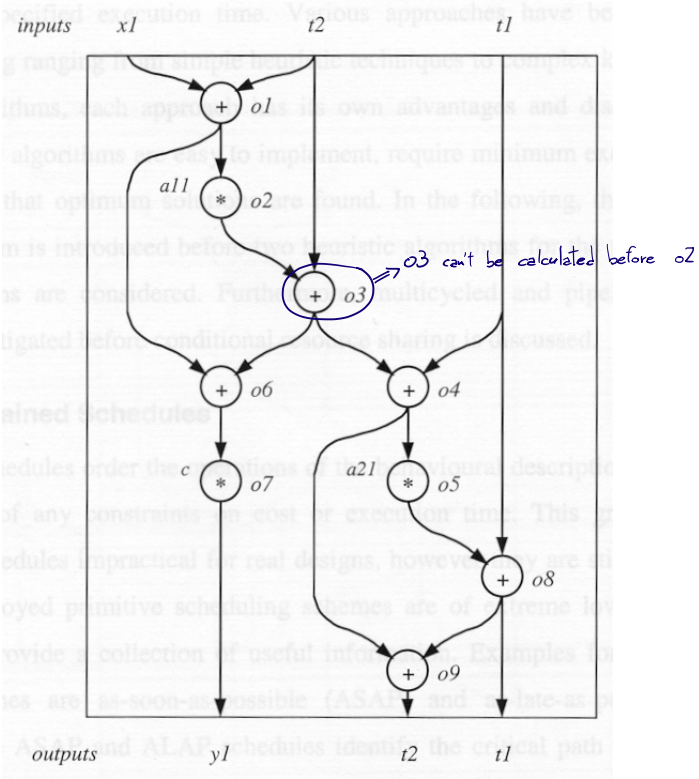
Algorithm:

$$y_1(kh+h) = c.(r_1 + r_2)$$
$$r_1 = x_1(kh) + t_2(kh)$$
$$r_2 = r_1.a_{11} + t_2(kh)$$
$$t_1(kh+h) = r_3$$
$$r_3 = r_4.a_{21} + t_1(kh)$$
$$r_4 = r_2 + t_1(kh)$$
$$t_2(kh+h) = r_3 + r_4$$

Verilog code:

```
/* A behavioral description of a digital filter

module digital_filter(x1,y1);

input x1;
output y1;
wire [7:0] r1,r2,r3,r4,t1,t2,c,a11,a21;

      assign r1 = x1 + t2;
      assign r2 = r1 * a11 + t2;
      assign r4 = r2 + t1;
      assign r3 = r4 + a21 + t1;

      assign y1 = c* (r1 + r2);
      assign t1 = r3;
      assign t2 = r3 + r4;

endmodule
```

Example – Unscheduled DFG



$$y_1(kh+h) = c.(r_1 + r_2)$$

$$r_1 = x_1(kh) + t_2(kh)$$

$$r_2 = r_1.a_{11} + t_2(kh)$$

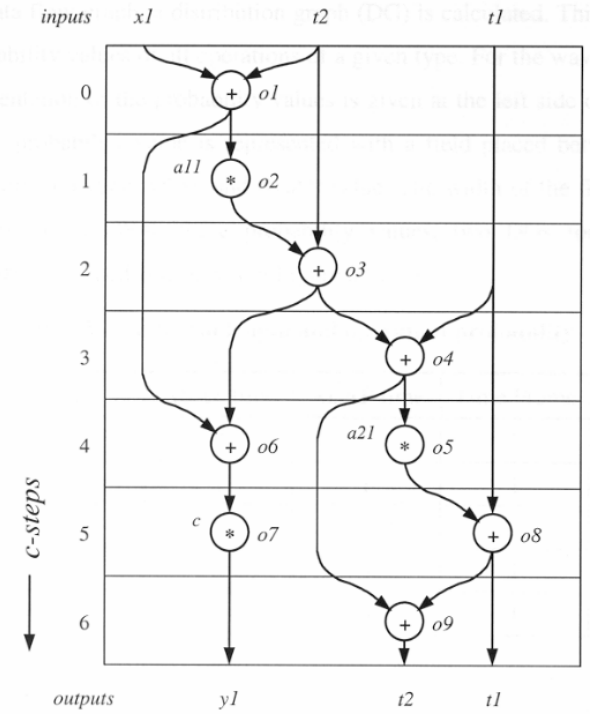
$$t_1(kh+h) = r_3$$

$$r_3 = r_4.a_{21} + t_1(kh)$$

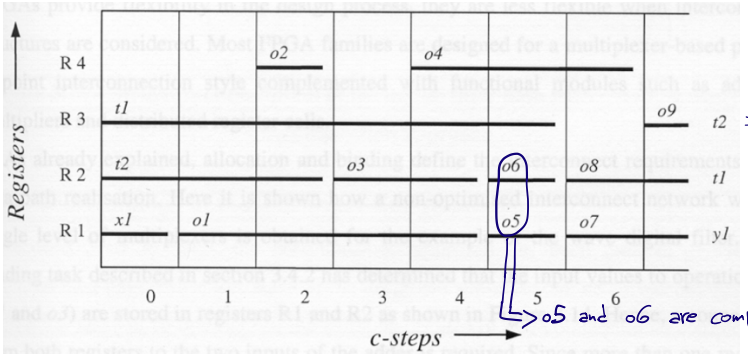
$$r_4 = r_2 + t_1(kh)$$

$$t_2(kh+h) = r_3 + r_4$$

Example – Scheduling and Regs mapping



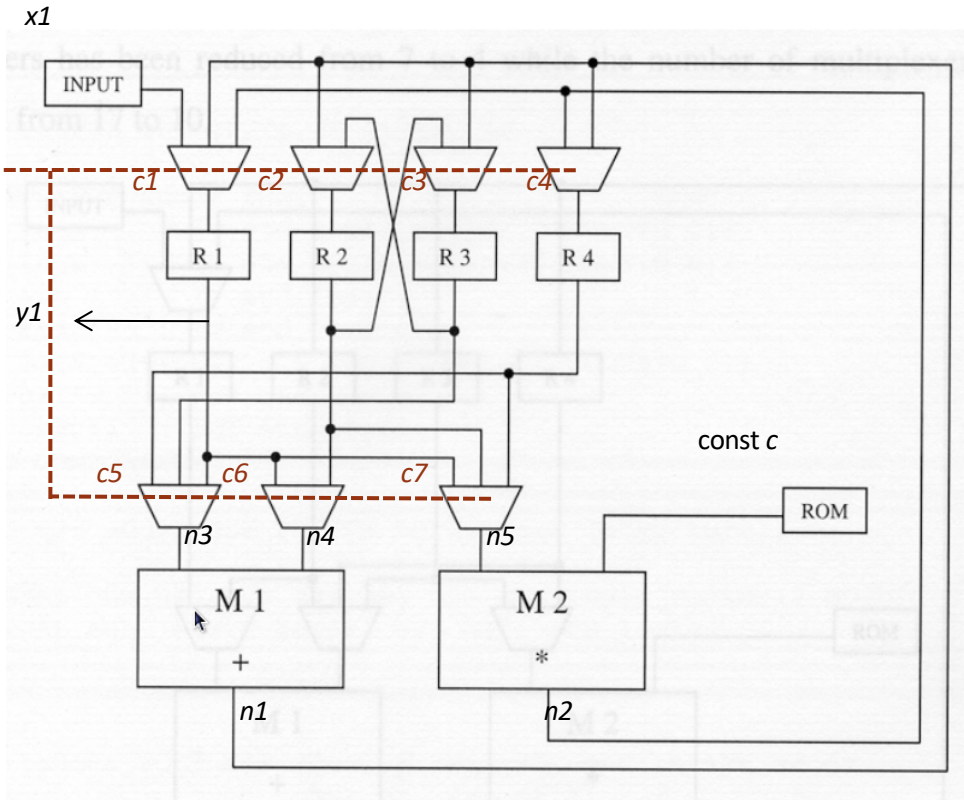
Resource-constraint scheduling (1 adder , 1 multiplier)



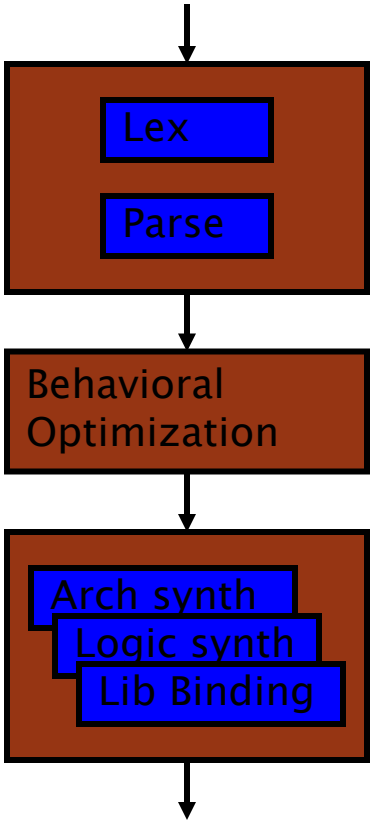
Register mapping (left-edge algorithm)

Example – Final Architecture

FSM controller



High-Level Synthesis Compilation Flow

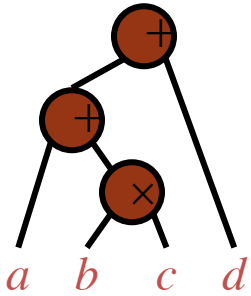


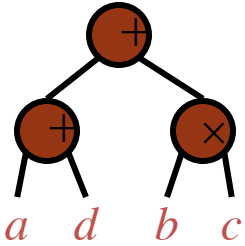
Compilation front-end

Intermediate form

HLS backend

$$x = a + b \times c + d$$



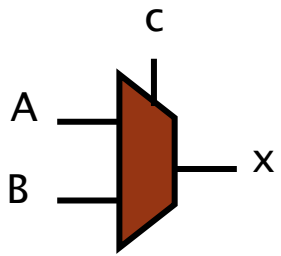


Behavioral Optimization

- Techniques used in software compilation
  - Expression tree height reduction
  - Constant and variable propagation
  - Common sub-expression elimination
  - Dead-code elimination
  - Operator strength reduction (e.g., *4 → << 2)
- Typical Hardware transformations
  - Conditional expansion
    - If (c) then x=A else x=B
      → compute A and B in parallel, x=(C)?A:B
  - Loop unrolling
    - Instead of k iterations of a loop, replicate the loop body k times

tricky ⇒ if a variable is computed once but used many times, we propagate the computation

⇒ if I have to multiply by a power of 2 I can shift

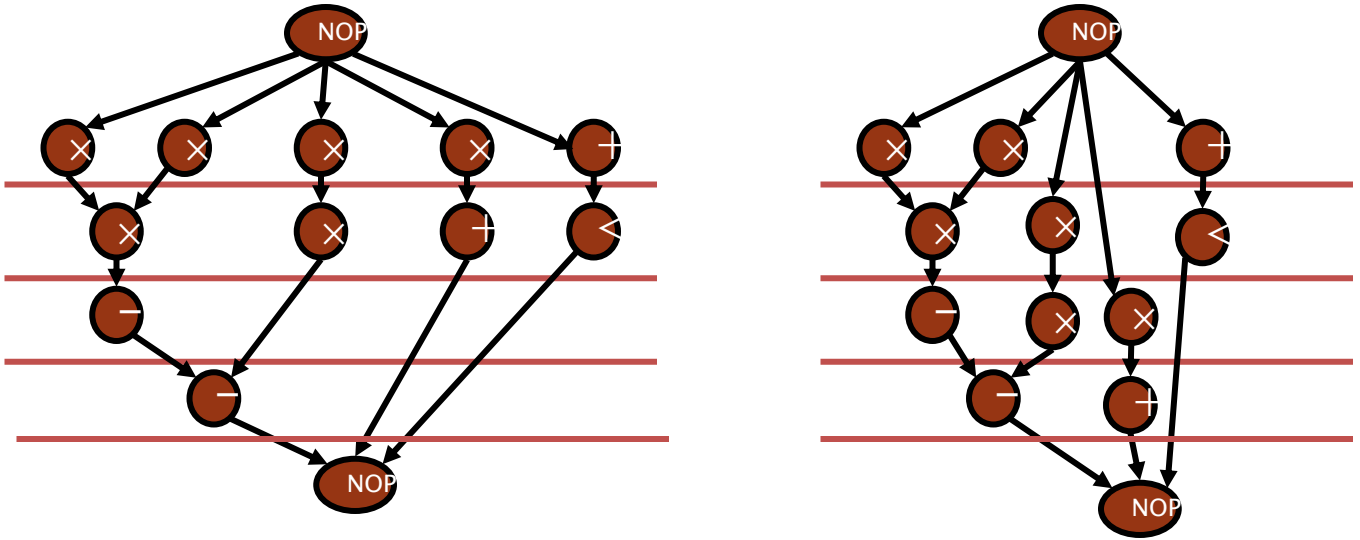


```
x=0;
...
if(x>0){
  ...
}
...
```

=> this is DEAD CODE => the pgm will never execute it

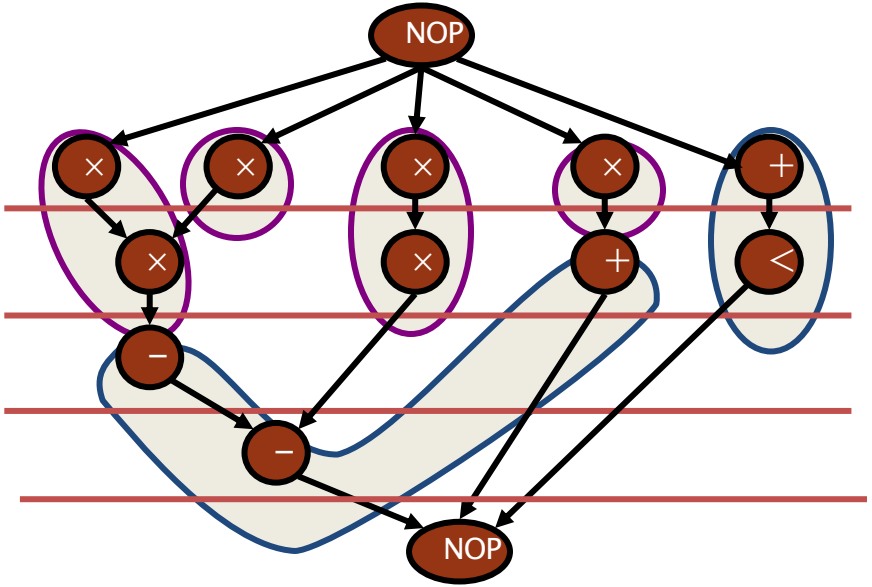
Optimization in Temporal Domain

- Scheduling and binding can be done in different orders or together
- Schedule:
  - Mapping of operations to time slots (cycles)
  - A scheduled sequencing graph is a labeled graph



Scheduling in Spatial Domain

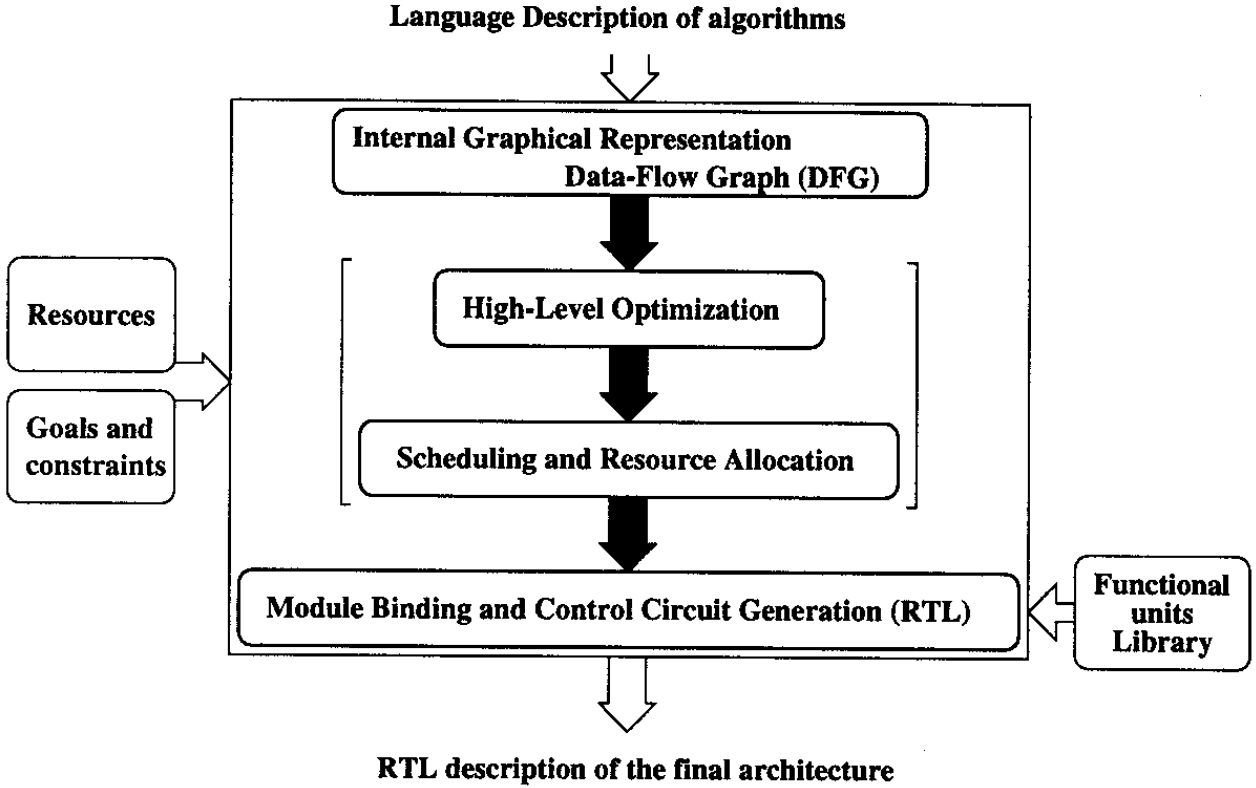
- Resource sharing
  - More than one operation bound to same resource
  - Operations have to be serialized
  - Can be represented using hyperedges (define vertex partition)



Architectural Optimization

- Optimization in view of design space flexibility
- A multi-criteria optimization problem:
  - Determine schedule f and binding b.
  - Under area A, latency L and cycle time t objectives
- Find non-dominated points in solution space
- Solution space tradeoff curves:
  - Non-linear, discontinuous
  - Area / latency / cycle time (more?)
- Evaluate (estimate) cost functions
- Unconstrained optimization problems for resource dominated circuits:
  - Min area: solve for minimal binding
  - Min latency: solve for minimum L scheduling

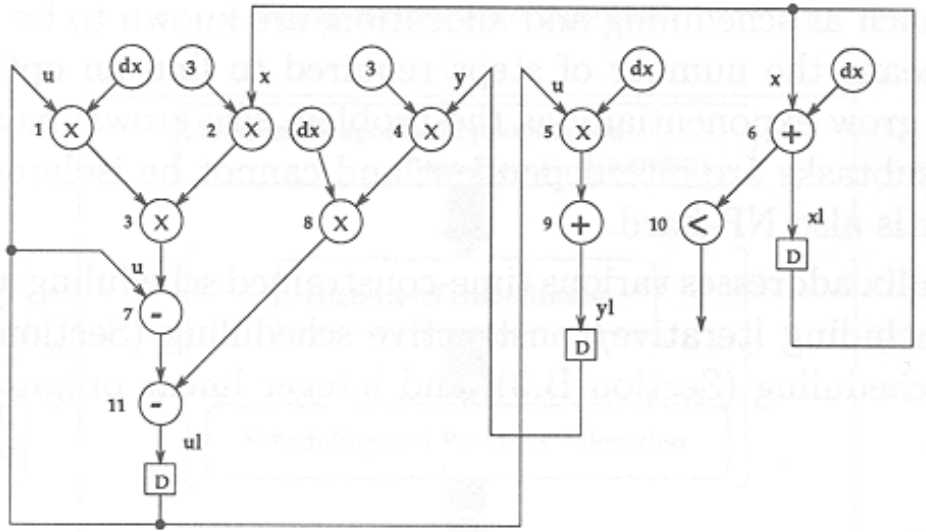
Typical High-Level Synthesis System



Algorithm Description

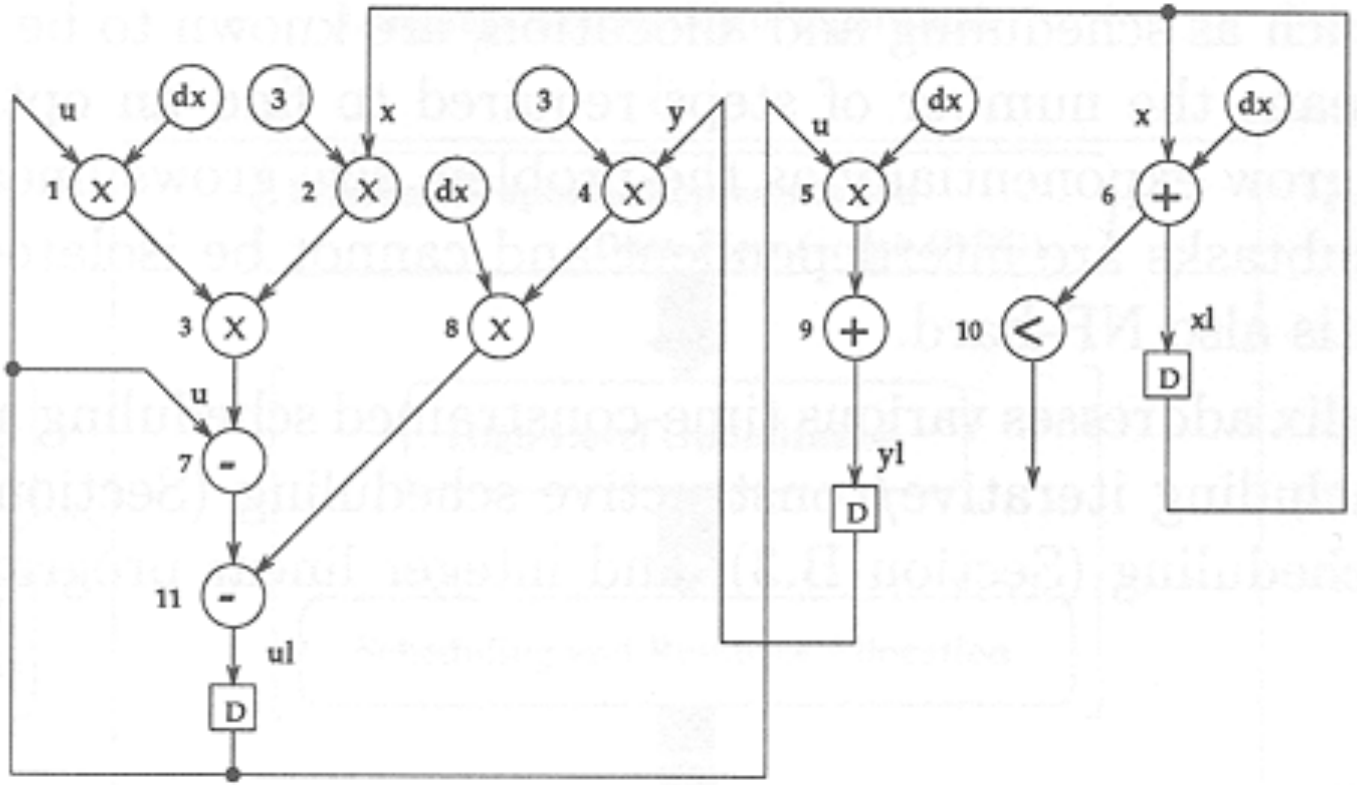
$$y'' + 3xy' + 3y = 0 \qquad u = y' = \frac{dy}{dx}$$

$$\frac{du}{dx} = y'' = \frac{d^2y}{dx^2} = -3xy' - 3y = -3xu - 3y$$

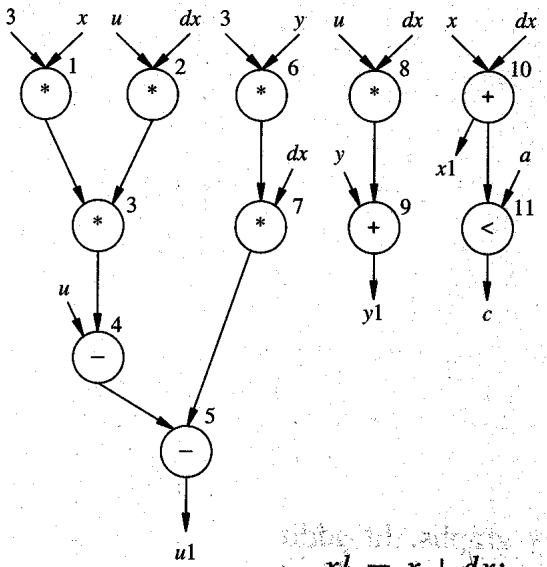


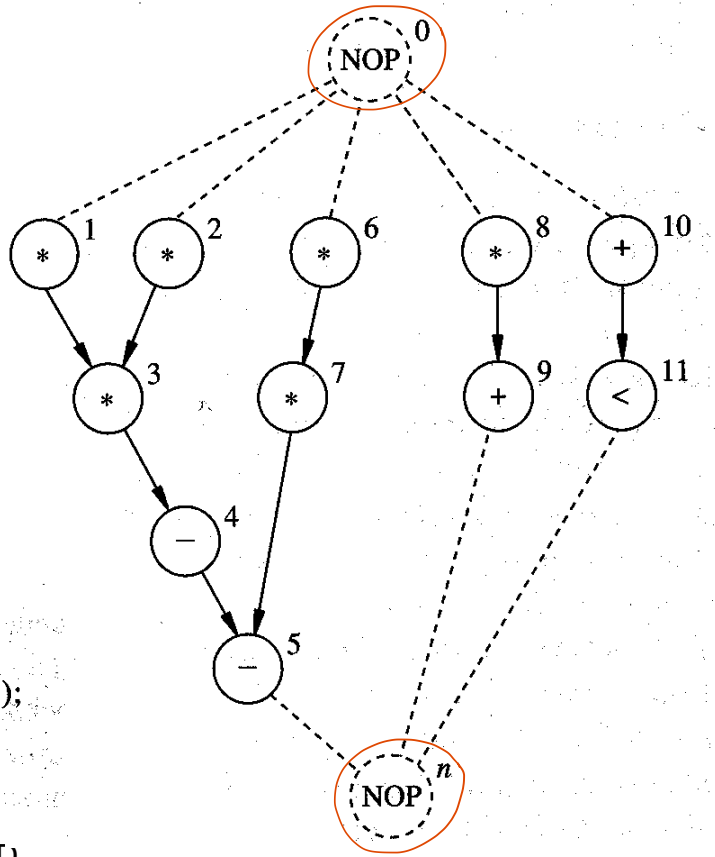

```
while (x < a) {
      xl = x + dx;
      ul = u - (3 * x * u * dx) - (3 * y * dx);
      yl = y + u * dx;
      x = xl; y = yl; u = ul;
}
```

Control Data Flow Graph (CDFG)



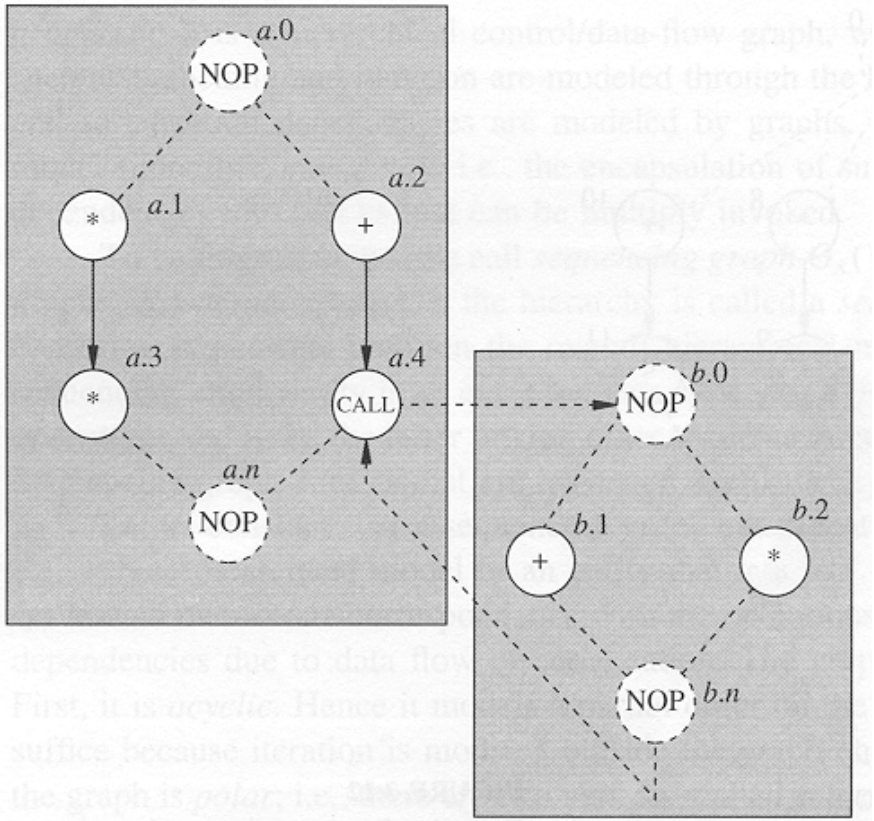
Sequence Graph: Start and End Nodes



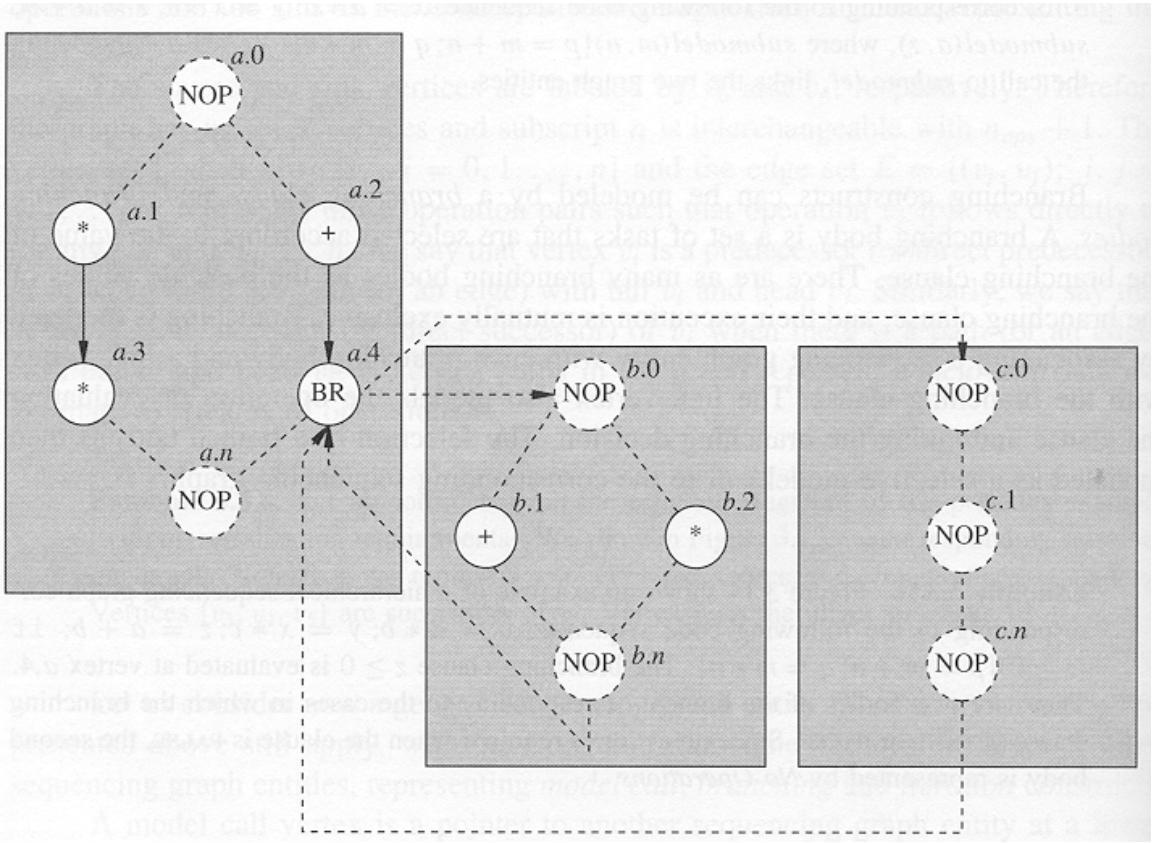


$$xl = x + dx;$$
$$ul = u - (3 * x * u * dx) - (3 * y * dx);$$
$$yl = y + u * dx;$$
$$c = xl < a;$$

Hierarchy in Sequence Graphs



Hierarchy in Sequence Graphs (contd.)



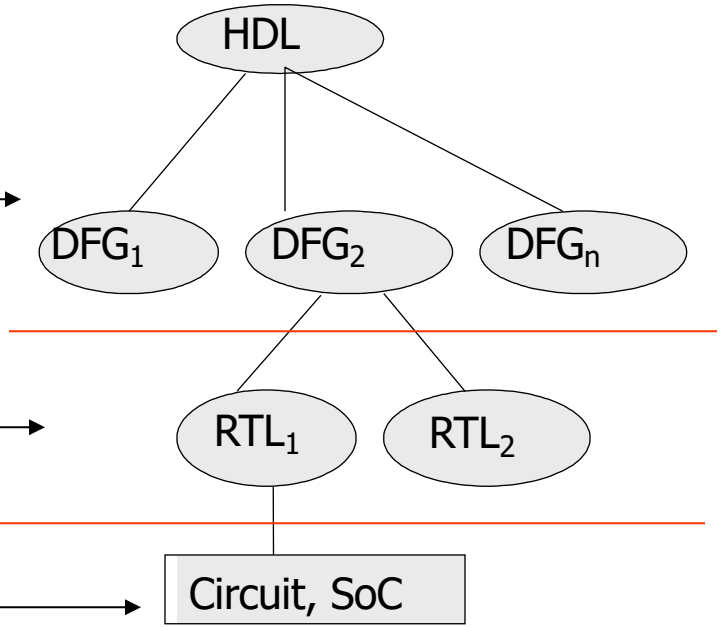
Synthesis Flow – mapping & optimization steps

Original specification →

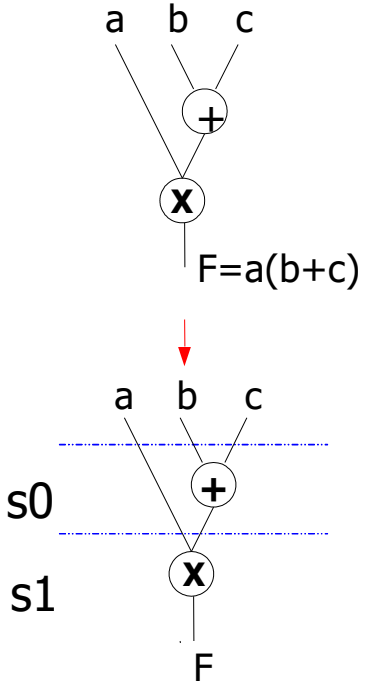
Un-scheduled DFG: one-to-one mapping from HDL spec. →

RTL optimized for particular objective →

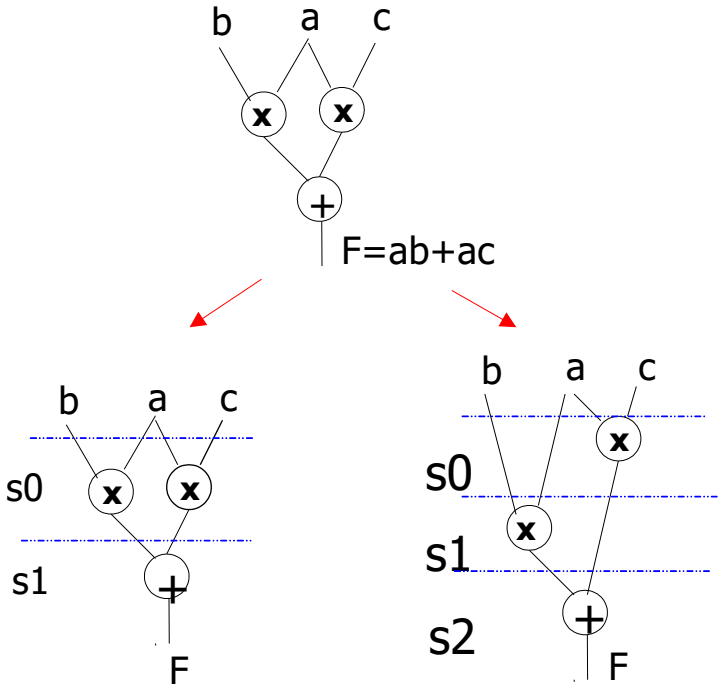
Physical Implementation →



DFG Scheduling



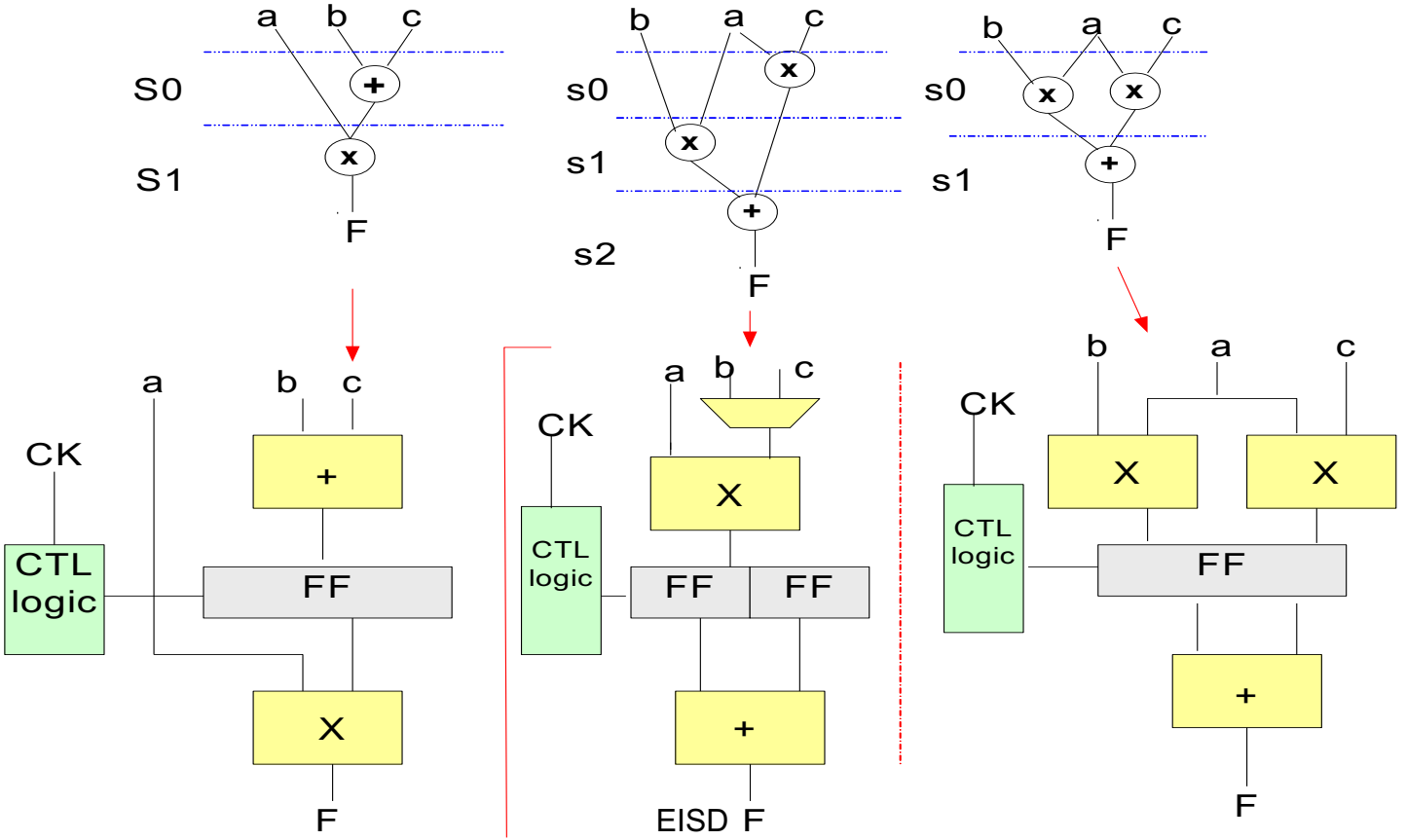
L=2, 1 Add, 1 Mult



L=2, 1 Add, 2 Mult

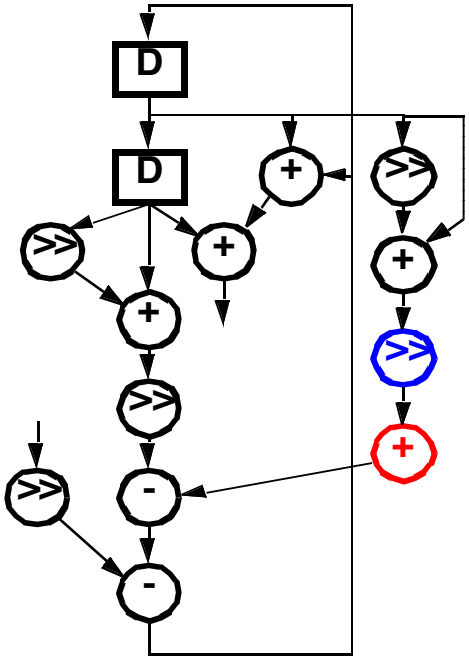
L=3, 1 Add, 1 Mult

Scheduling affects RTL



SCHEDULING: HEURISTIC ALGORITHMS

Scheduling, Allocation and Assignment



Allocation: How Much?

2 adders
1 shifter
24 registers

Assignment: Where?

Shifter 1

Schedule: When?

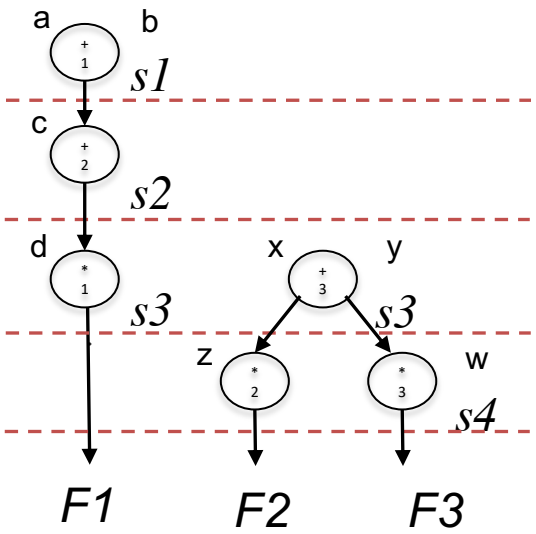
Time Slot 4

Techniques are well understood and mature

Scheduling and Assignment - Overview

$F1 = (a + b + c) * d$ $F2 = (x + y) * z$ $F3 = (x + y) * w$

Schedule 1



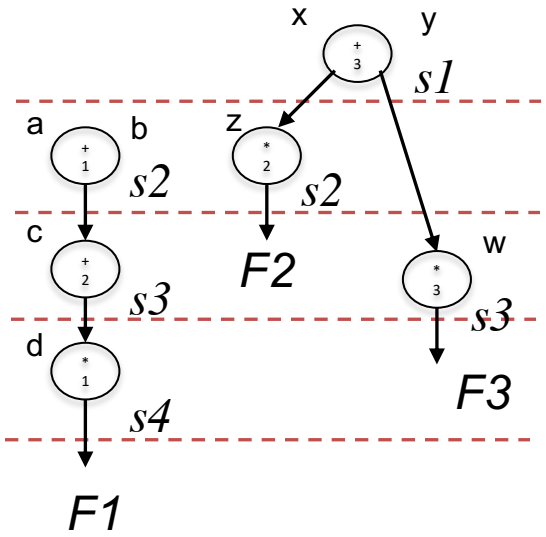
4 control steps, 1 Add, 2 Mult

Control Step	+	*	*
1	+1		
2	+2		
3	+3	*1	
4		*2	*3

Scheduling and Assignment - Overview

$F1 = (a + b + c) * d$ $F2 = (x + y) * z$ $F3 = (x + y) * w$

Schedule 2



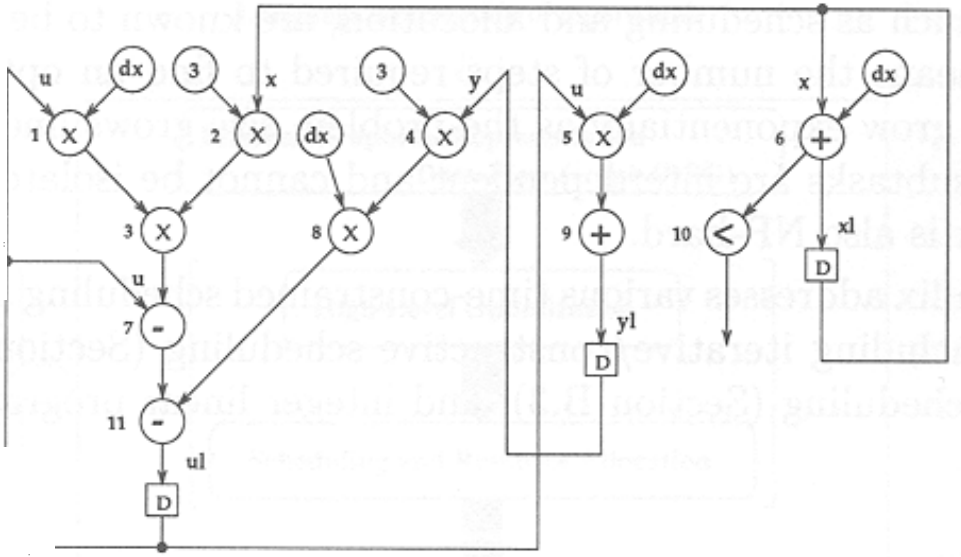
4 control steps, 1 Add, 1 Mult

Control Step	+	*
1	+3	
2	+1	*2
3	+2	*3
4		*1

Algorithm Description → Data Flow Graph

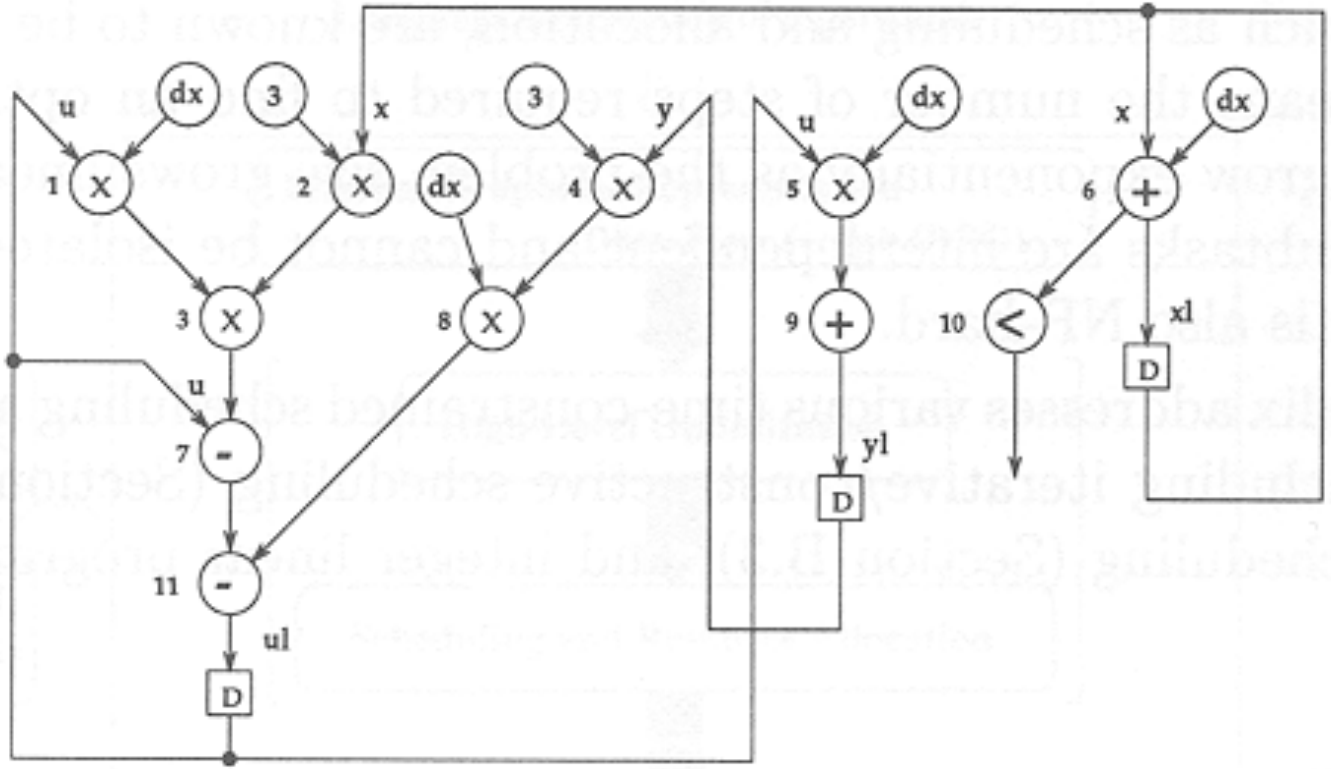
$$y'' + 3xy' + 3y = 0 \qquad u = y' = \frac{dy}{dx}$$

$$\frac{du}{dx} = y'' = \frac{d^2y}{dx^2} = -3xy' - 3y = -3xu - 3y$$



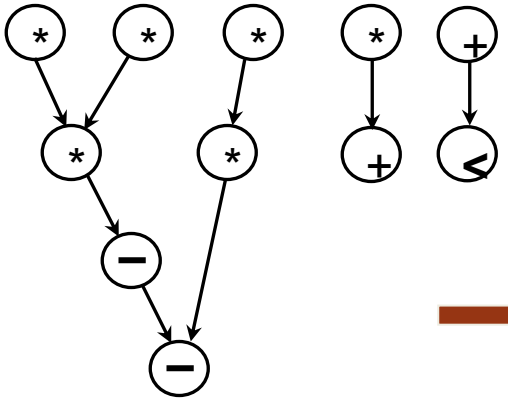

```
while (x < a) {
      xl = x + dx;
      ul = u - (3 * x * u * dx) - (3 * y * dx);
      yl = y + u * dx;
      x = xl; y = yl; u = ul;
}
```

Data Flow Graph (DFG)

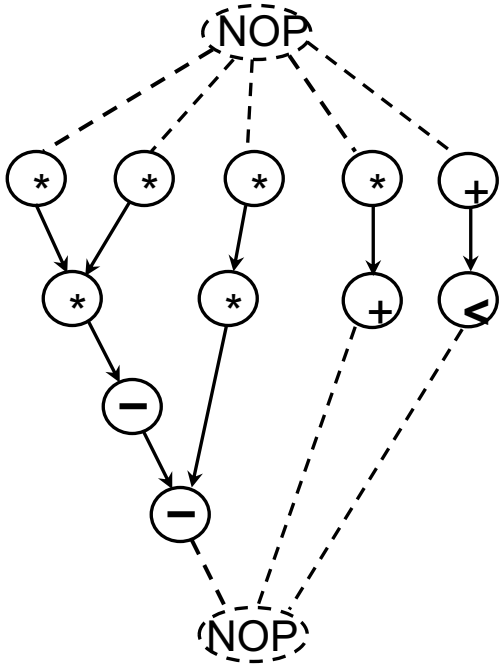


Sequencing Graph

- Add source and sink nodes (NOP) to the DFG



Data Flow Graph (DFG)



Sequencing Graph

ASAP Scheduling Algorithm

- As Soon as Possible scheduling
  - Unconstrained minimum latency scheduling
  - Uses topological sorting of the sequencing graph (polynomial time)
  - Gives optimum solution to scheduling problem
  - Schedule first the first node no → T1 until last node *nv* is scheduled

Input: DFG $G = (N, E)$.
Output: ASAP Schedule.

1. $TS_0 = 1$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   - 2.1 Select a node n_j whose predecessors have already been scheduled;
   - 2.2 Schedule node n_j to time step $TS_j = \max\{ TS_i + (C_i)\}$ $\forall\ n_i \rightarrow n_j$;

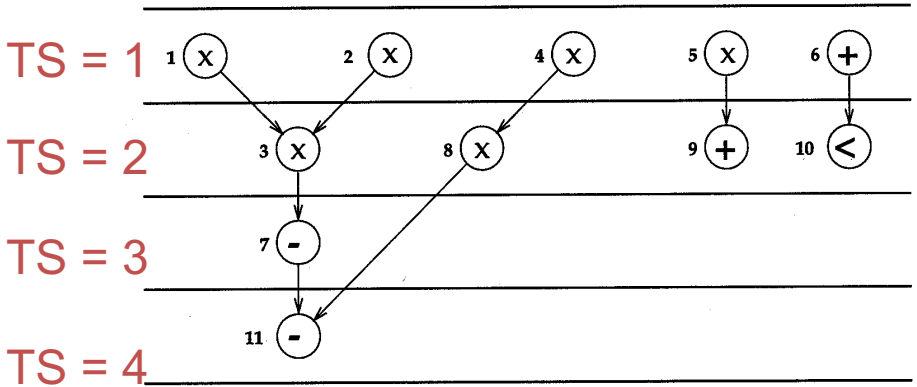
}

ASAP Scheduling Algorithm - Example

Input: DFG $G = (N, E)$.
Output: ASAP Schedule.

1. $TS_0 = 1$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   - 2.1 Select a node n_j whose predecessors have already been scheduled;
   - 2.2 Schedule node n_j to time step $TS_j = \max\{TS_i + (C_i)\}$ $\forall\ n_i \rightarrow n_j$;

}



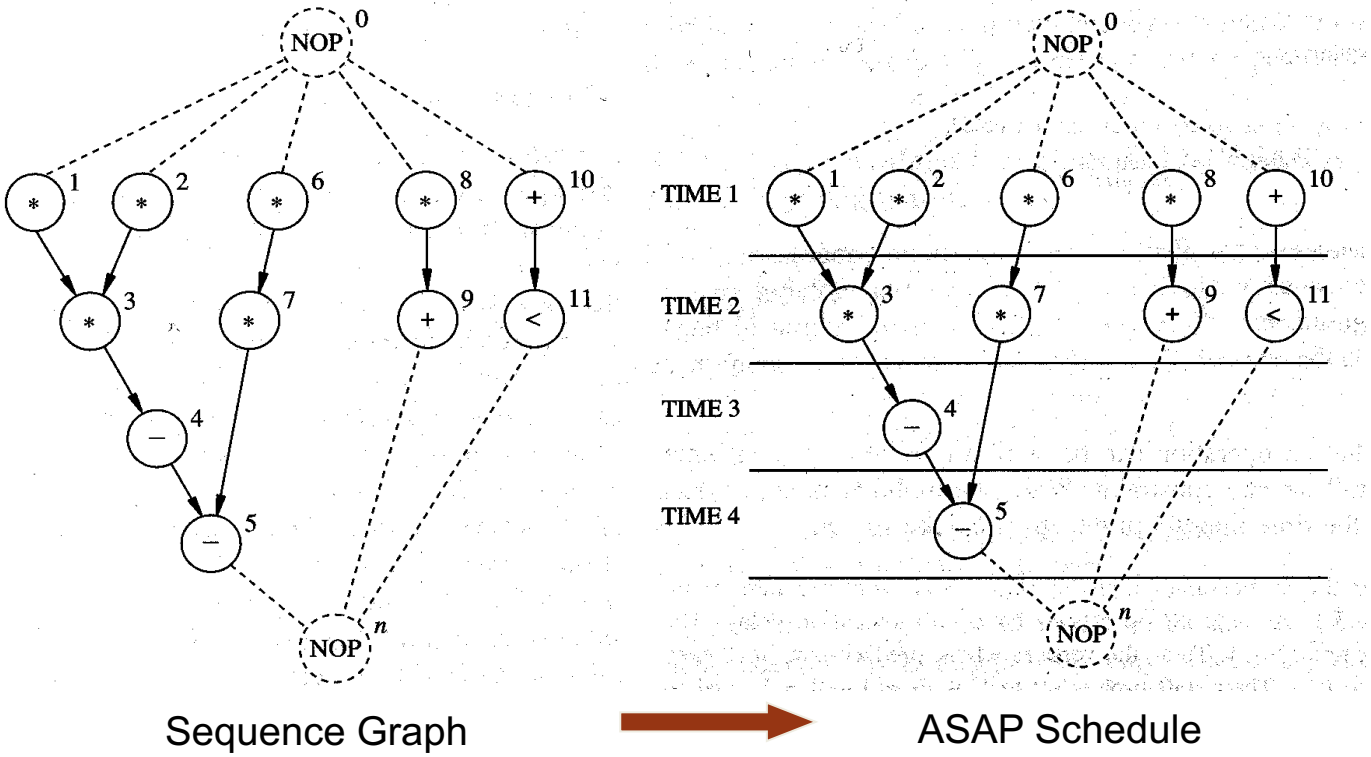
start

finish

I need:
4 mult
1 plus
1 minus

Optimum only for latency, but not for resources

ASAP Scheduling Example



ALAP Scheduling Algorithm

- As late as Possible scheduling
  - Latency-constrained scheduling (latency is fixed)
  - Uses reversed topological sorting of the sequencing graph
  - If over-constrained (latency too small), solution may not exist
  - Schedule first the last node *nv* → T, until first node n0 is scheduled

Produce the same latency of ASAP, postponing each operation as late as possible

Input: DFG $G = (N, E)$, $IterationPeriod = T$. *(Latency)*
Output: ALAP Schedule.
1. $TS_0 = T$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   2.1 Select a node n_i whose successors have already been scheduled;
   2.2 Schedule node n_i to time step $TS_i = \min\{ TS_j - (C_i)\}$ $\forall\ n_i \rightarrow n_j$;
   }

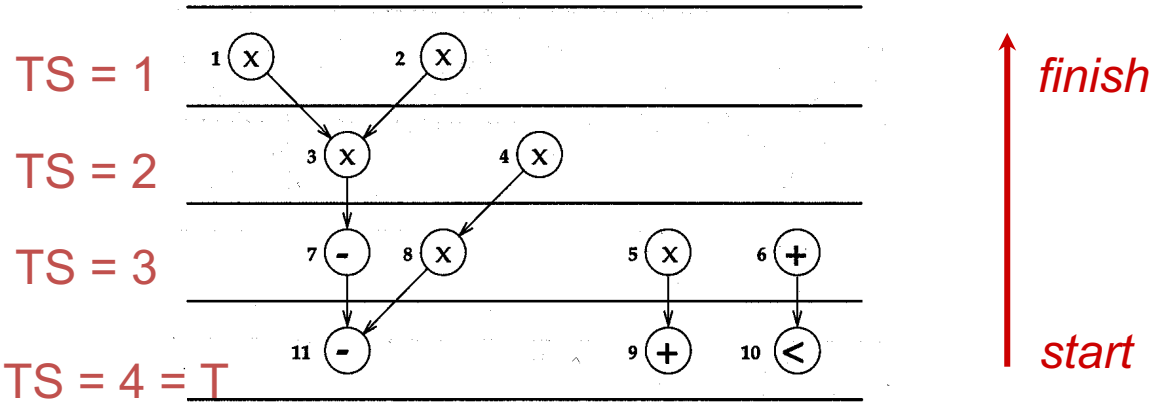
ALAP Scheduling Algorithm - example

Input: DFG $G = (N, E)$, $IterationPeriod = T$.
Output: ALAP Schedule.

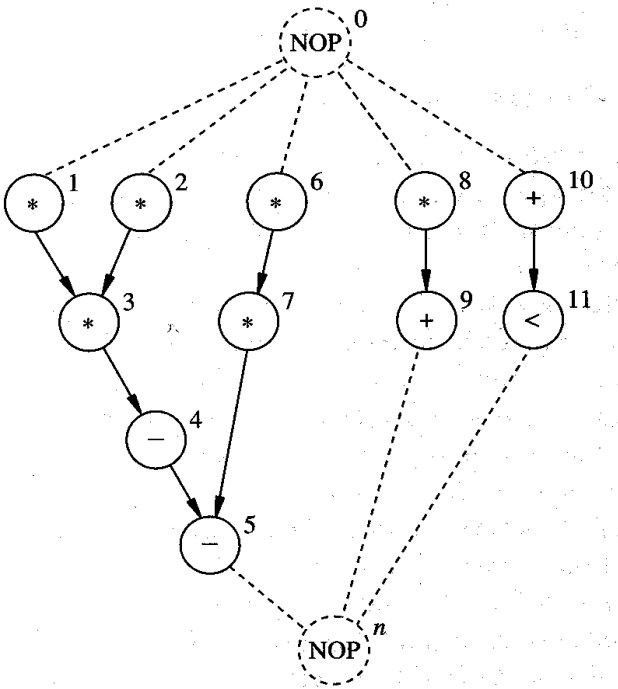
1. $TS_0 = T$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   - 2.1 Select a node n_i whose successors have already been scheduled;
   - 2.2 Schedule node n_i to time step $TS_i = \min\{ TS_j - (C_i)\}$ $\forall\ n_i \rightarrow n_j$;

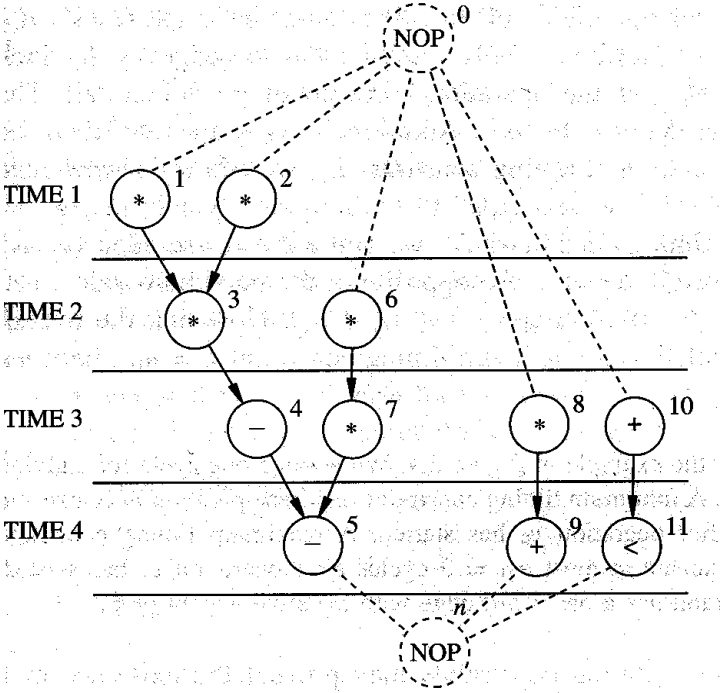
}

Optimal for latency
But it's not optimal for latency because that's not its target
In this case we need
2 mults
2 adders
1 minus
1 component less than ASAP but this is a case



ALAP Scheduling Example





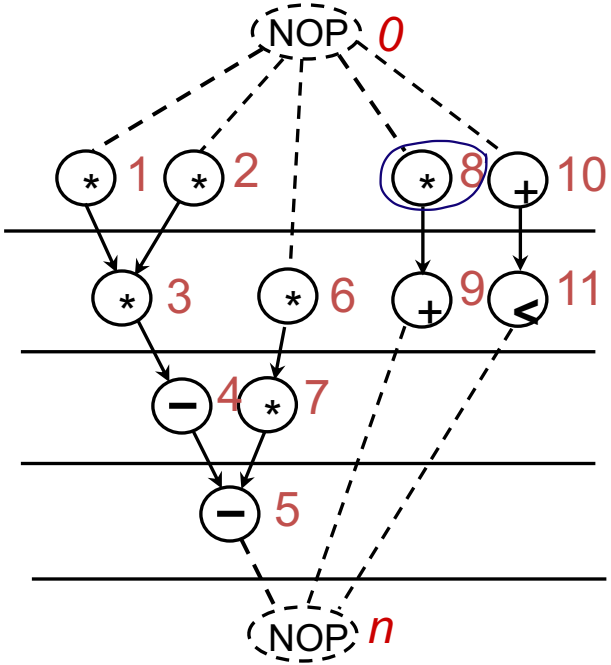
Sequence Graph

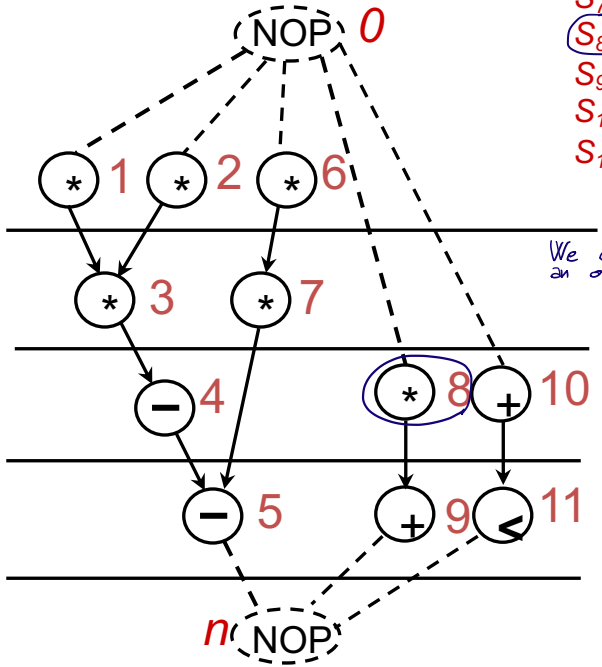
EISD

ALAP Schedule
(latency constraint = 4)

Computing Slack (mobility)

- Slack of Operator i:
  - Si = TSiALAP – TSiASAP





$S_6 = 2-1 = 1$
$S_7 = 2-1 = 1$
$S_8 = 3-1 = 2$
$S_9 = 4-2 = 2$
$S_{10} = 3-1 = 2$
$S_{11} = 4-2 = 2$

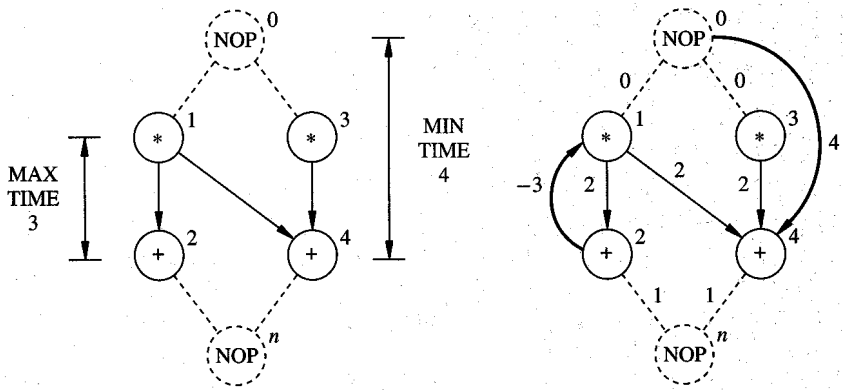
We can move the nodes to loon for an optimal solution
⇒ Does an algo exist?

Multipliers :
move node 6 at T=2
move node 7 at T=3
So we can have 2 multipliers

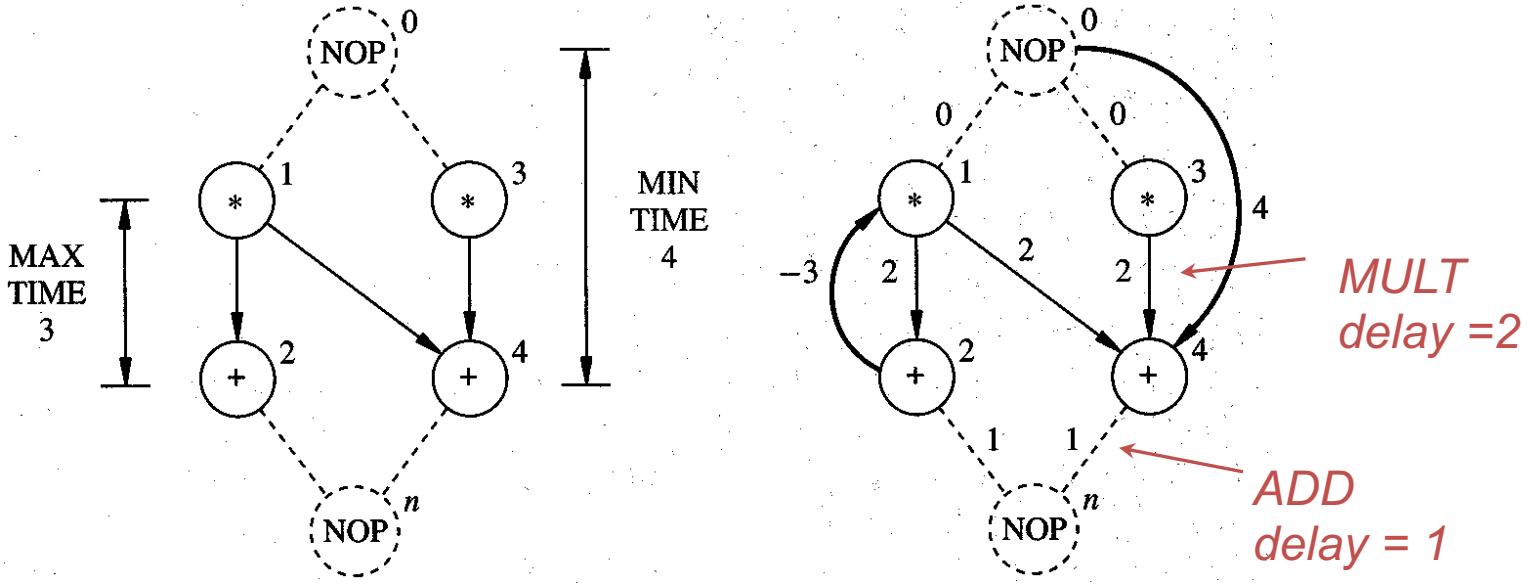
Timing Constraints

- Time measured in cycles or control steps
- Imposing relative timing constraints between operators i and j
  - max & min timing constraints

A **minimum** timing constraint $l_{ij} \geq 0$ requires: $t_j \geq t_i + l_{ij}$.
A **maximum** timing constraint $u_{ij} \geq 0$ requires: $t_j \leq t_i + u_{ij}$.



Constraint Graph Gc(V,E)



Scheduling – a Combinatorial Optimization Problem

- NP-complete Problem
- Optimal solutions for special cases and ILP
- Heuristics - iterative Improvements
- Heuristics – constructive
- Various versions of the problem
  - Unconstrained, minimum latency
  - Resource-constrained, minimum latency
  - Timing-constrained, minimum latency
  - Latency-constrained, minimum resource

Allocation: Resource Binding & Sharing

Binding and Sharing Problem

- Given: scheduled sequencing graph
  - Operation concurrency well defined
- Consider operation types independently
  - Problem decomposition (natural)
  - Perform analysis for each resource type
- Operation compatibility
  - Same type
  - Non-concurrent
- Conflicting operations
  - Concurrent, different types
  - Dual to compatibility

Allocation (Binding)

- Allocation = resource binding
  - Spatial mapping between operations and resources
  - Operators can be dedicated or generic (shared)
  - Operators and registers need to be allocated
- Sharing
  - Assignment of a resource to more than one operation

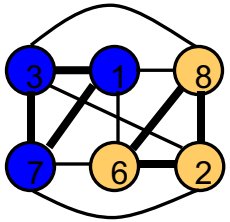
- Constrained resource binding
  - Resource-dominated circuits
  - Fixed number and type of resources available
- NP-complete problem – need heuristics

Binding in Resource-Dominated Circuits

- Resource Compatibility Graph G+(V,E)
  - V represents operations
  - E represents compatible operation pairs
- Compatible operations
  - (vi, vj) are compatible if they are not concurrent and can be implemented by resources of same type
  - Note: concurrency depends on schedule
- Partition the graph into minimum number of cliques in G+(V,E)
  - Clique = maximal complete subgraph
  - Partition the graph into minimum number of cliques, or
  - Clique cover number, $\kappa(G+(V,E))$

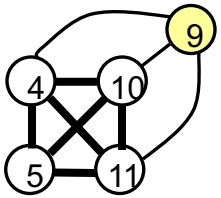
Compatibility Graph G+(V,E)

- Minimum Clique covers in G+(V,E)



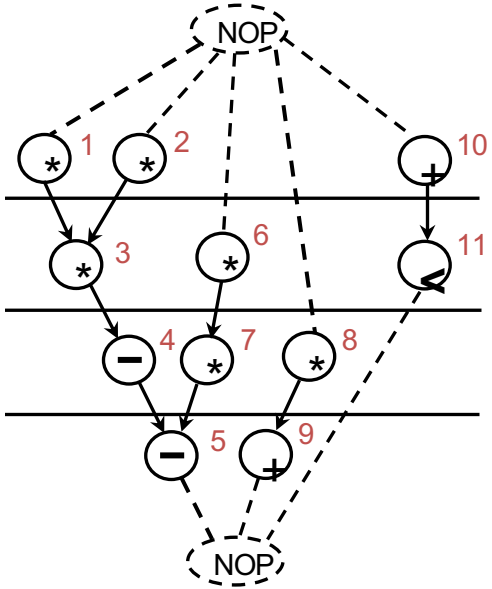
MULT

$\kappa(G_+(V,E)) = 2$



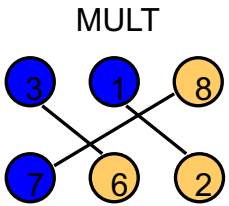
ALU

$\kappa(G_+(V,E)) = 2$

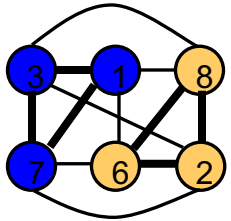


Conflict Graph G-(V,E)

- Resource Conflict Graph G-(V,E)
  - V represents operations
  - E represents conflicting operation pairs
- Conflicting operations
  - Two operations are conflicting if they are not compatible
- Complementary to compatibility graph
- Find independent set of G-(V,E)
  - A set of mutually compatible operations
  - Coloring with minimum number of colors
  - Chromatic number χ(G-(V,E))



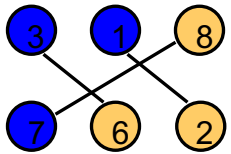
Conflict graph $G_{-}(V,E)$



Compatibility graph $G_{+}(V,E)$

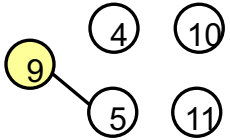
Conflict Graph G-(V,E) - Example

- Chromatic numbers in G-(V,E)



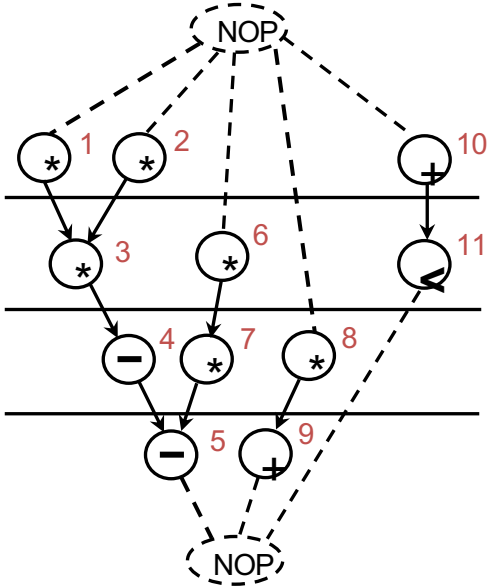
MULT

$\chi(G_-(V,E)) = 2$



ALU

$\chi(G_-(V,E)) = 2$

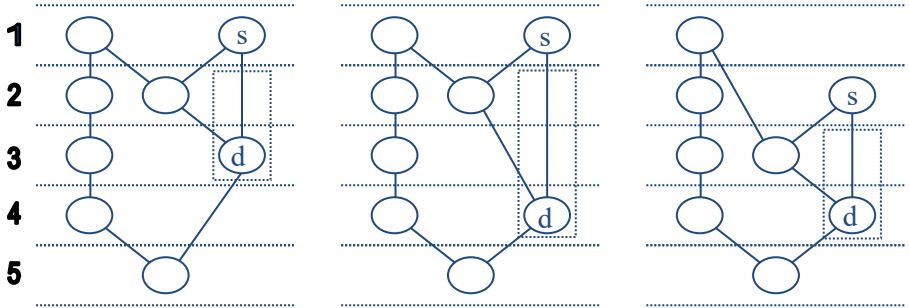


Register Binding Problem

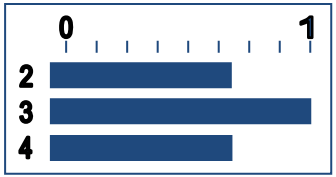
- Registers are storage resources, holding variable values across control steps
- Given a schedule, generate:
  - Lifetime intervals for variables
  - Lifetime overlaps
- Construct a conflict graph (interval graph)
  - Vertices V : variables (operations)
  - Edges E: overlaps
  - Build an interval graph
- Compatibility graph (comparability graph)
  - Complement of conflict graph

Minimization of Register Costs

- Given a scheduled sequencing graph
  - Minimum set of registers required is given by the largest number of data arcs crossing a C-step boundary
- Create storage operations, at output of any operation that transfers a value to a destination in a later C-step
- Generate Storage DG for these “operations”
- Length of storage operation depends on final schedule



ASAP Lifetime MAX Lifetime ALAP Lifetime



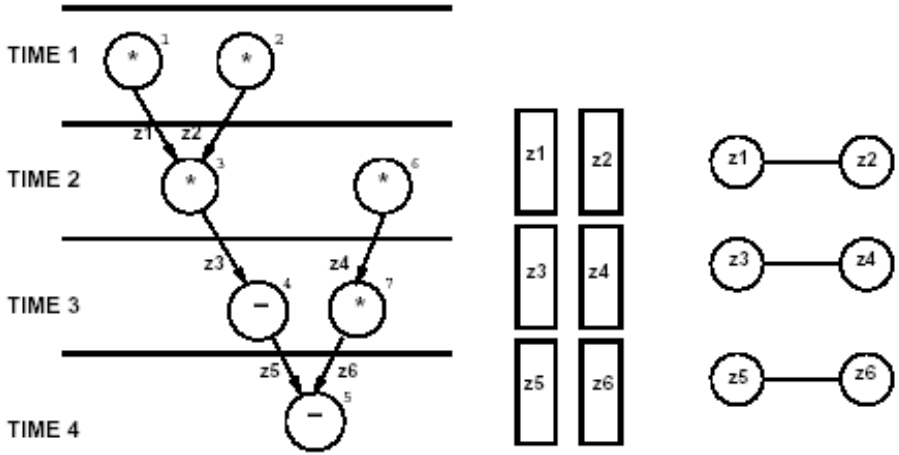
Storage distribution for S

Register Binding Problem

- Given
  - Variable lifetime conflict graph
- Find
  - Minimum number of registers storing all variables
- Simple case
  - Non-iterative designs: Interval graph
    - Solve using left-edge algorithm (polynomial time)

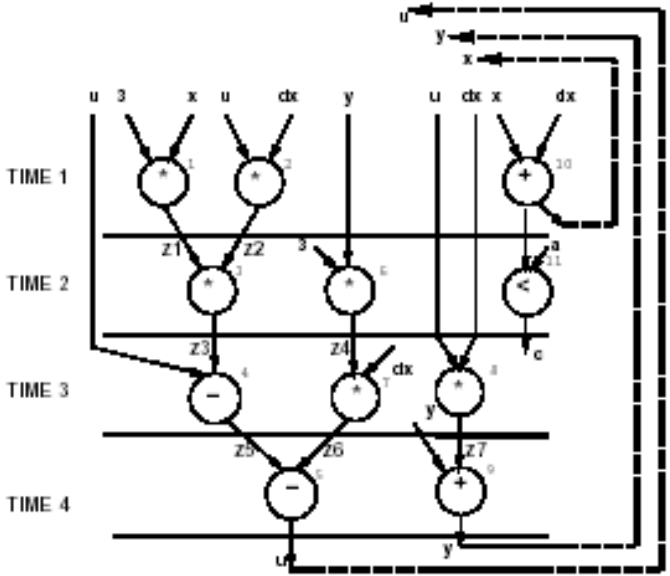
Register Binding Problem – Example 1

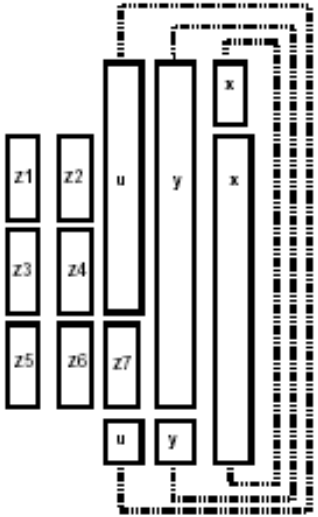
- Non-iterative designs
  - Create variable compatibility graph or conflict graph
  - Use left-edge algorithm to minimize the number of registers



Register Binding – Example 2

- Iterative designs
  - Sequencing graph and variable lifetimes





Register Sharing – General Case

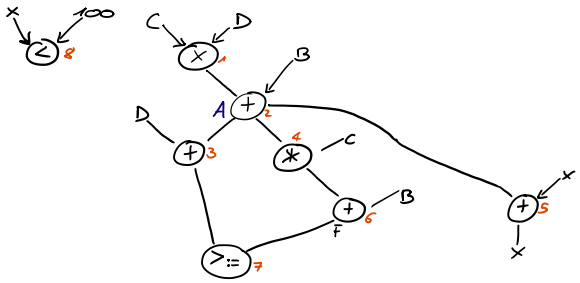
- Iterative constructs
  - Preserve values across iterations
  - Circular-arc conflict graph (not simple intervals)
    - Coloring is intractable
- Hierarchical graphs:
  - General conflict graphs
    - Coloring is intractable
- Heuristic algorithms required

Summary

- Resource sharing and binding is reducible to coloring or clique covering
- Simple for flat (non-hierarchical) graphs
- Intractable in general case, but still easy in practice for other graphs
- More complicated for non resource-dominated circuits
- Extension: module selection

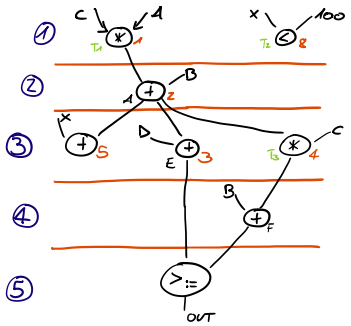
EXAMPLE

```
while (x<100) {
   A=B+C*D;
   E=A+D;
   F=A*C+B;
   IF(F>E)
      OUT=E;
   ELSE
      OUT=F;
   x=x+A;
}
```



1) Identify possible behavioural optimizations (not for this example)
   E.g.: E=B+C*D+D ⟺ E=B+D+C*D
2)

ASAP ALLOCATION



Last = 5
(<) => 1
(*) => 1
(+) => 2
(>:=) => 1

ALAP is the best solution => 1+ and 1*

T_1	T_2	T_3	A	B	C	D	E	F
1								
2								
3								
4								
5								

$R_2 = \{T_1, A, T_3, F\}$
$R_3 = \{E\}$