CAD for VLSI

Binding

Outline

- Motivation and problem formulation
- Flat and hierarchical graphs
- Functional and memory resources
- Extension to module selection
- Data path generation
- Control synthesis

Allocation and Binding

- Allocation:
  - Number of resources available
- Binding:
  - Relation between operations and resources
- Sharing:
  - Many-to-one relation
- Optimum binding/sharing:
  - Minimize the resource usage

Binding

- Limiting cases:
  - Dedicated resources
    - One resource per operation
    - No sharing
  - One multi-task resource
    - ALU
  - One resource per type

Optimum Sharing Problem

- Scheduled sequencing graphs
  - Operation concurrency well defined
- Consider *operation types* independently
  - Problem decomposition
  - Perform analysis for each resource type

Compatibility and Conflicts

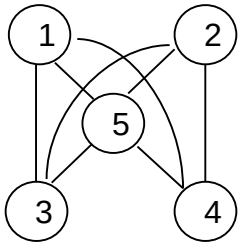
- Operation compatibility:
  - Same type
  - Non-concurrent

t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x-y	3	4
t3	z=a+t		5	

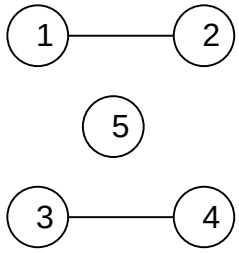
- *Compatibility* graph:
  - Vertices: operations
  - Edges: compatibility relation

- *Conflict* graph:
  - Complement of compatibility graph

Compatibility graph

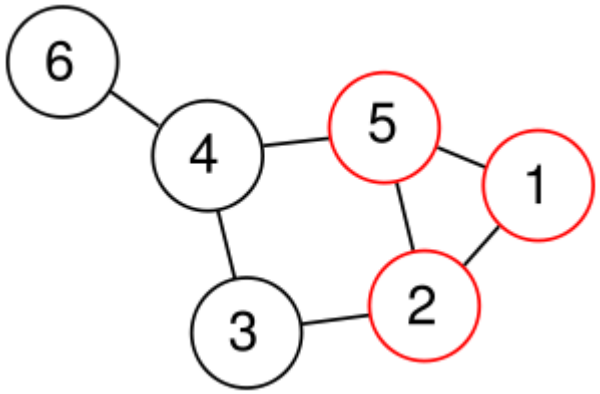


Conflict graph



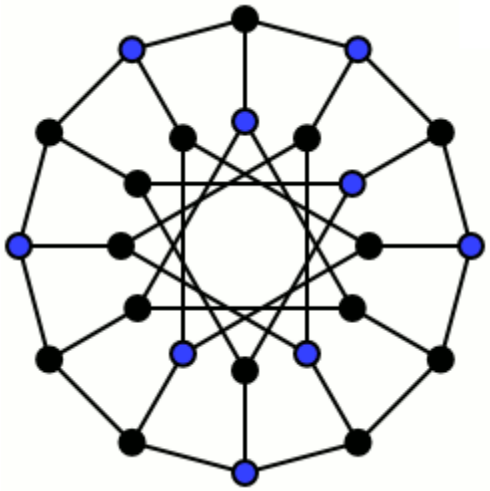
Clique

- Clique
  - A clique in an undirected graph *G* is a subset of vertices *V'* such that each pair of the vertices is connected by an edge. (i.e. a clique is a complete subgraph of *G*)



Independent Set

- Independent set (stable set)
  - An independent set of a graph *G* is a subset of vertices *V'* such that any two vertices are not connected to each other.

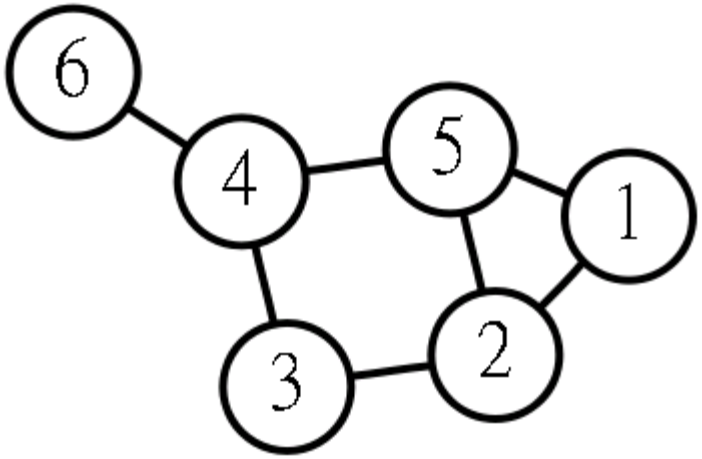


Graph Terminologies

- Clique number $\omega(G)$: the cardinality of the largest clique (maximum clique)
- Clique cover number $\kappa(G)$: the cardinality of a minimum clique cover
- Stability number $\alpha(G)$: The cardinality of the largest stable set (independent set)
- Chromatic number $\chi(G)$: The smallest number that can be the cardinality of stable set partition

Example

- Clique number $\omega(G)$
- Clique cover number $\kappa(G)$
- Stability number $\alpha(G)$
- Chromatic number $\chi(G)$



Graph Properties

- $\omega(G) \leq \chi(G)$: clique number ≤ chromatic number
- $\alpha(G) \leq \kappa(G)$: stability number ≤ clique cover number

- Perfect graph: "=" holds

Compatibility and Conflicts

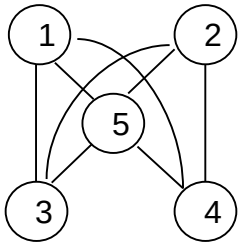
- Operation compatibility:
  - Same type
  - Non concurrent

t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x-y	3	4
t3	z=a+t		5	

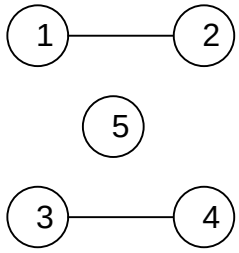
- *Compatibility* graph:
  - Vertices: operations
  - Edges: compatibility relation

- *Conflict* graph:
  - Complement of compatibility graph

Compatibility graph



Conflict graph



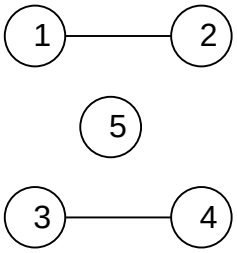
Compatibility and Conflicts

- Compatibility graph:
  - Partition the graph into a minimum number of cliques
  - Find clique cover number $\kappa(G_+)$
- Conflict graph:
  - Color the vertices by a minimum number of colors
  - Find the chromatic number $\chi(G_-)$
- NP-complete problems:
  - Heuristic algorithms

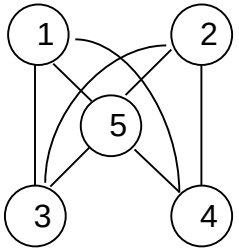
Example

t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x-y	3	4
t3	z=a+t		5	

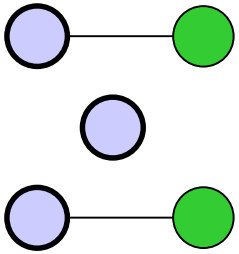
Conflict



Compatibility



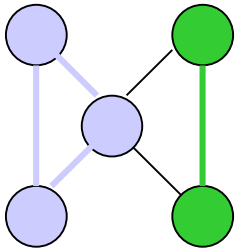
Coloring



ALU1: 1,3,5

ALU2: 2,4

Partitioning (Clique covering)



Special Perfect Graphs

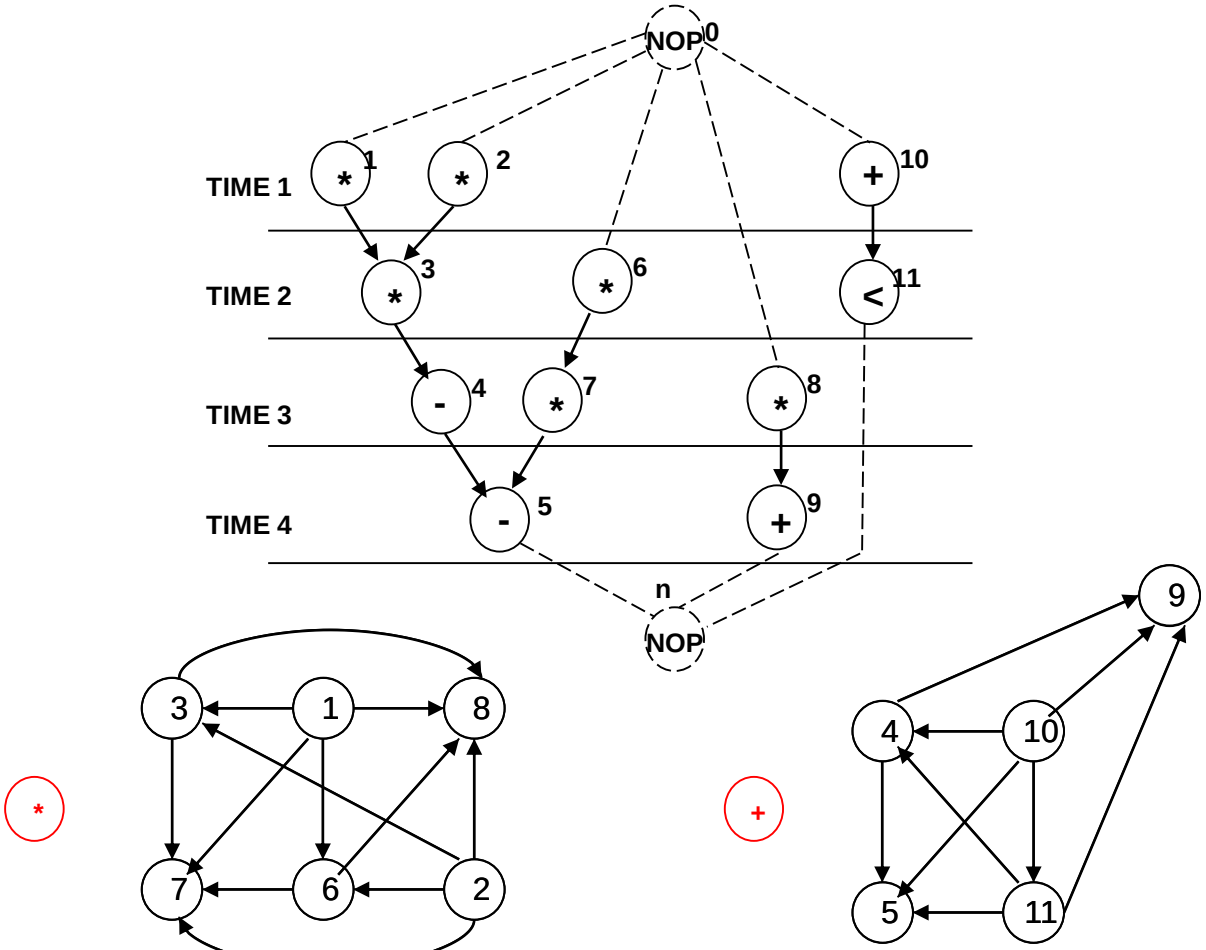
- *Comparability graph*:
  - Graph $G(V, E)$ has an orientation $G(V, F)$ with the transitive property

    $(v_i, v_j) \in F$ and $(v_j, v_k) \in F \rightarrow (v_i, v_k) \in F$
- *Interval graph*:
  - Vertices correspond to *intervals*
  - Edges correspond to interval intersection
  - Subset of *chordal* graphs
    - Every loop with more than three edged has a chord

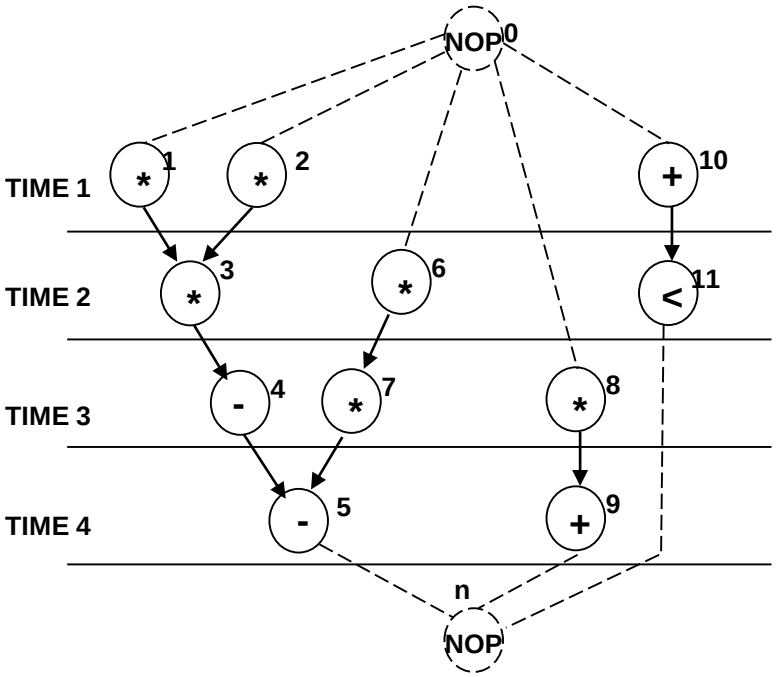
Data-flow Graphs

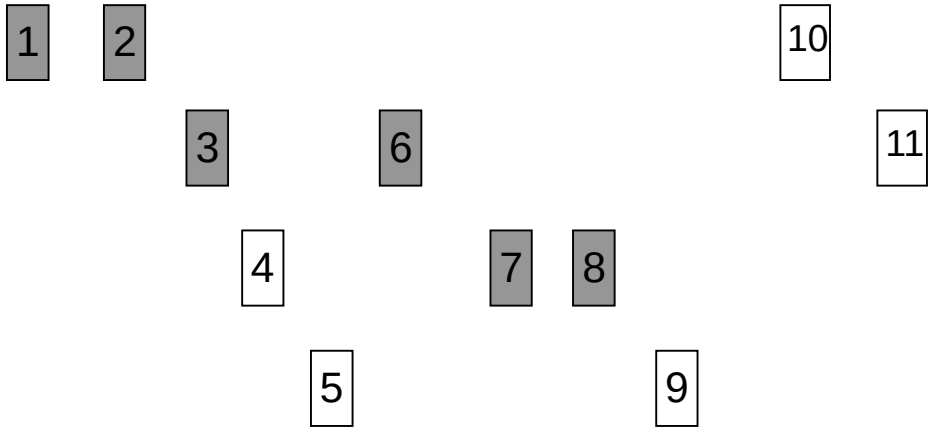
- The compatibility/conflict graphs have special properties:
  - Compatibility
    - Comparability graph
  - Conflict
    - Interval graph
- Polynomial time solutions:
  - Golumbic's algorithm
  - Left-edge algorithm

Comparability Graph Example



Interval Graph Example





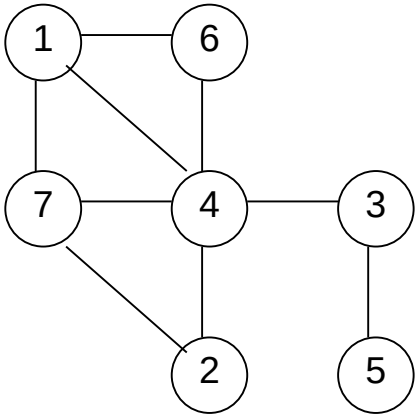
Left-edge Algorithm

- Input:
  - Set of intervals with *left* and *right edge*
  - A set of *colors* (initially one color)
- Rationale:
  - Sort intervals in a *list* by *left* edge
  - Assign non overlapping intervals to first color using the list
  - When possible intervals are exhausted, increase color counter and repeat

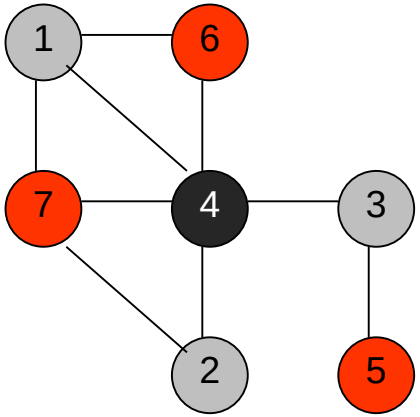
Left-edge Algorithm

```
LEFT_EDGE(I) {
    Sort elements of I in a list L in ascending order of l_i;
    c = 0;
    while (some interval has not been colored) do {
        S = Ø;
        r = 0;
        while ( exists s ∈ L such that l_s > r) do {
                s = First element in the list L with l_s > r;
                S = S U {s};
                r = r_s;
                Delete s from L;
        }
        c = c + 1;
        Label elements of S with color c;
    }
}
```

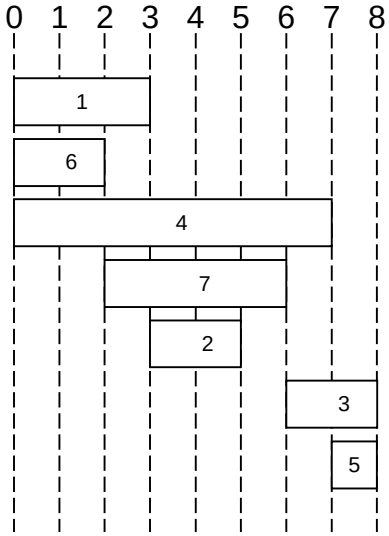
Example



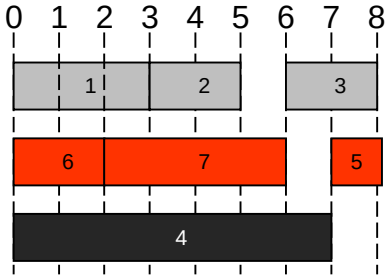
Conflict graph



Colored conflict graph



Intervals



Coloring

ILP Formulation of Binding

- Boolean variable b_{ir}
  - Operation i bound to resource r
- Boolean variables x_{il}
  - Operation i scheduled to start at step l

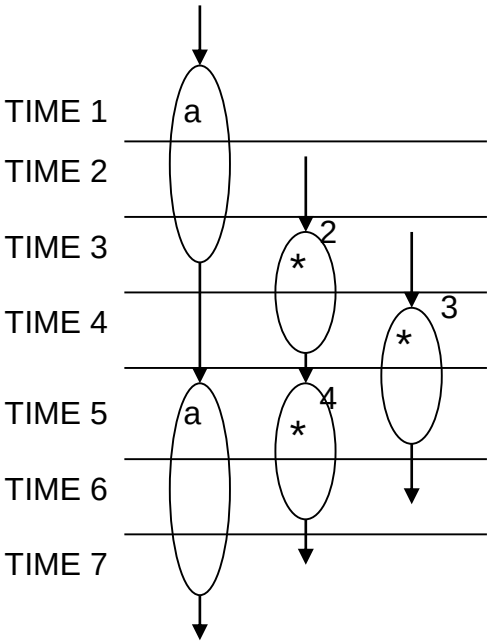
$$\sum_r b_{ir} = 1 \qquad \text{for all operations } i$$

$$\sum_i b_{ir} \sum_{m=l-di+1..l} x_{im} \leq 1 \qquad \text{for all steps } l \text{ and resources } r$$

Hierarchical Sequencing Graphs

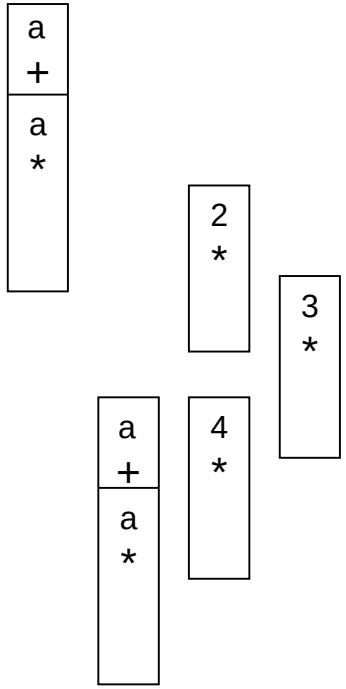
- Hierarchical conflict/compatibility graphs:
  - Easy to compute
  - Prevent sharing across hierarchy
- Flattened hierarchy:
  - Bigger graphs
  - Destroy nice properties

Example of Multiple Calls



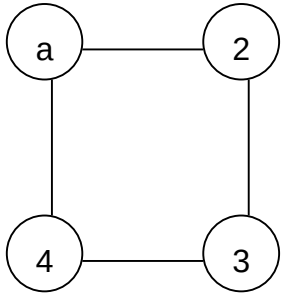
Multiple calls of "a"

(a)



Not an interval graph

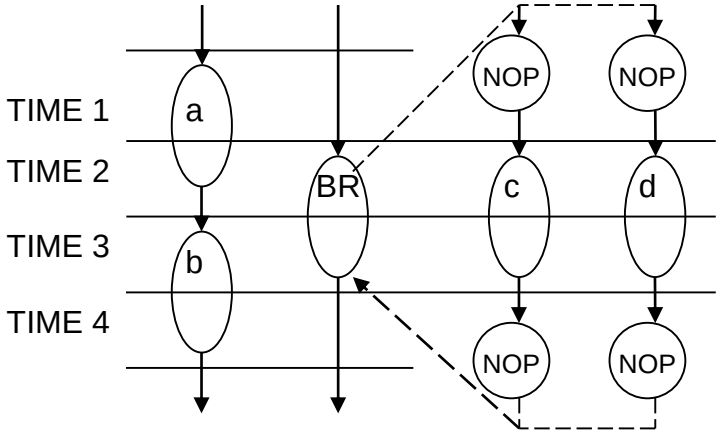
(b)



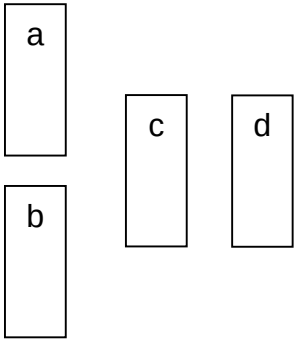
Non-chordal conflict graph

(c)

Example of Branching

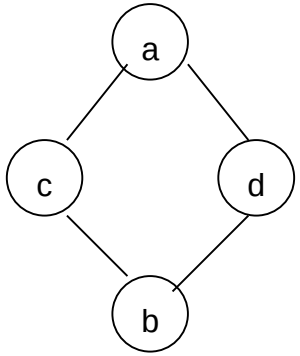


(a)



Execution intervals

(b)



c and d are compatible
→ Non-chordal conflict graph

(c)

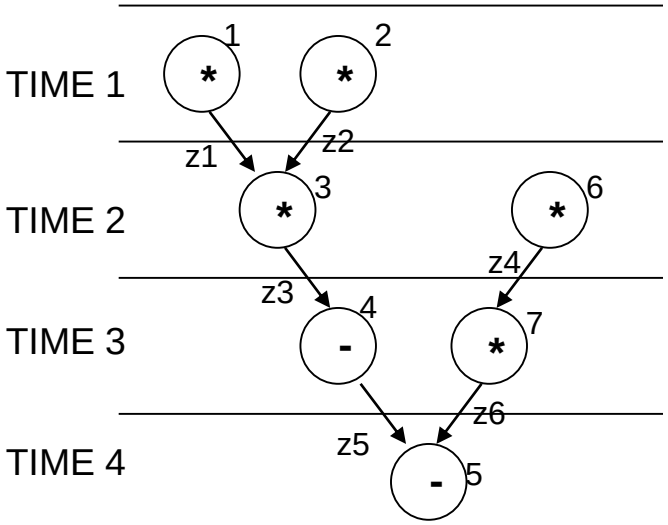
Register Binding Problem

- Given a schedule:
  - *Lifetime intervals* for variables
  - *Lifetime overlaps*
- Conflict graph (interval graph):
  - Vertices ↔ variables
  - Edges ↔ overlaps
  - Interval graph
- Compatibility graph (comparability graph):
  - Complement of conflict graph

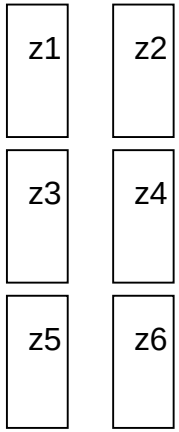
Register Sharing in Data-flow Graphs

- Given:
  - Variable lifetime conflict graph
- Find:
  - Minimum number of registers storing all the variables
- Key point:
  - Interval graph
    - Left-edge algorithm (polynomial-time complexity)

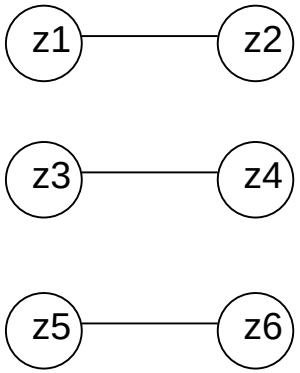
Example of DFG



(a)



(b)

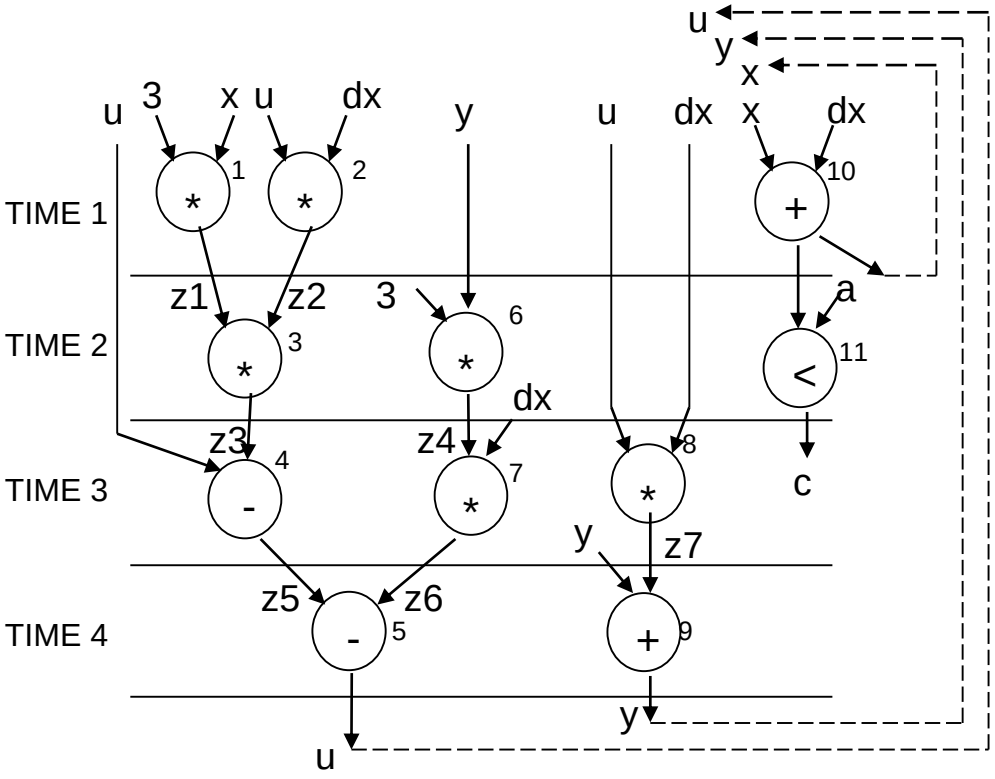


(c)

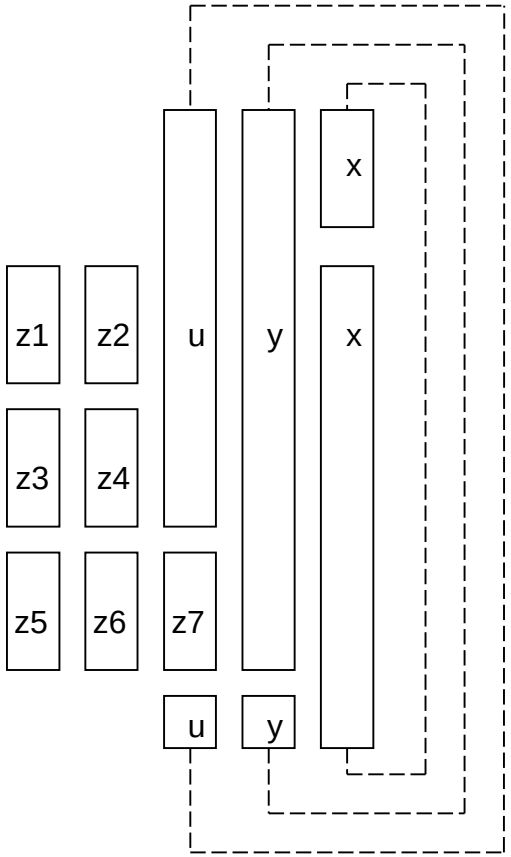
Register Sharing General Case

- Iterative conflicts:
  - Preserve values across iterations
  - Circular-arc conflict graph
    - Coloring is intractable
- Hierarchical graphs:
  - General conflict graphs
    - Coloring is intractable
- Heuristic algorithms

Example of General Case

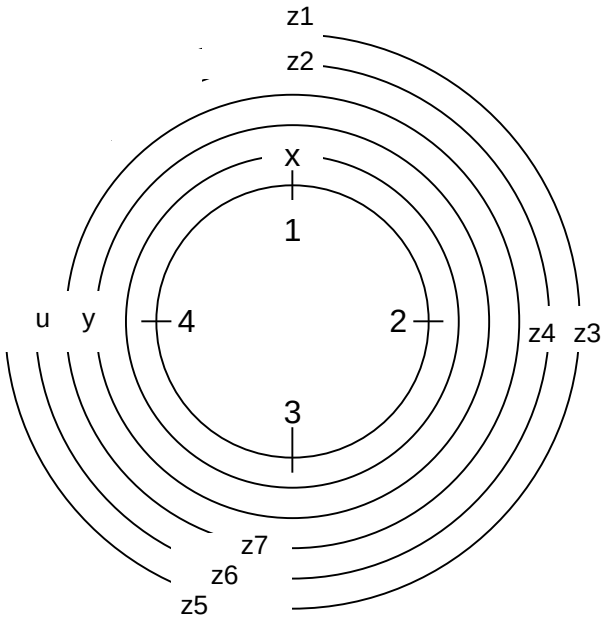


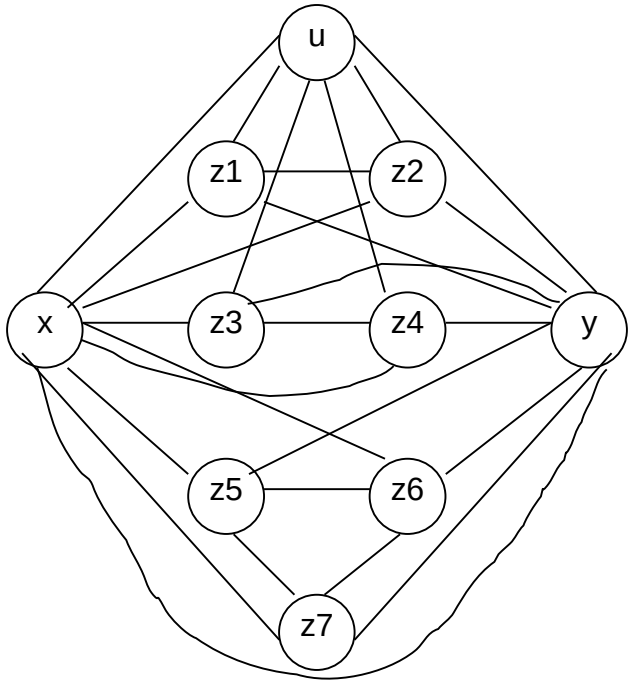
(a)



(b)

Example Variable-lifetimes and Circular-arc Conflict Graph





Multi-port Memory Binding

- Assume memory is large enough to hold all data
- Find *minimum number* of ports to access the required number of variables
- Variables use the same port (fixed port):
  - Port compatibility/conflict
  - Similar to resource binding
- Variables can use any port (dynamic port):
  - Decision variable x_{il} is TRUE when variable *i* is accessed at step *l*
  - Optimum: max $\sum_{i=1..n_{var}} x_{il}$ ***s.t.*** $\mathbf{1 \le l \le \lambda + 1}$

Multi-port Memory Binding

- Find max number of variables to be stored through a fixed number of ports *a*
  - Boolean variables { b_i, $i = 1, 2, \ldots, n_{var}$ }:
    - Variable with b_i =1 will be stored in memory

max $\sum_{i=1..n_{var}} b_i$ such that

$\sum_{i=1..n_{var}} b_i \, x_{il} \leq a \qquad l = 1,2,\ldots,\lambda + 1$

Example – Dynamic Port

Time – step 1 : $r_3 = r_1 + r_2$; $r_{12} = r_1$

Time – step 2 : $r_5 = r_3 + r_4$; $r_7 = r_3 * r_6$; $r_{13} = r_3$

Time – step 3 : $r_8 = r_3 + r_5$; $r_9 = r_1 + r_7$; $r_{11} = r_{10} / r_5$

Time – step 4 : $r_{14} = r_{11} \,\&\, r_8$; $r_{15} = r_{12} \mid r_9$

Time – step 5 : $r_1 = r_{14}$; $r_2 = r_{15}$

$$\max \sum_{i=1} b_i \text{ such that}$$

$$b_1 + b_2 + b_3 + b_{12} \le a$$
$$b_3 + b_4 + b_5 + b_6 + b_7 + b_{13} \le a$$
$$b_1 + b_3 + b_5 + b_7 + b_8 + b_9 + b_{10} + b_{11} \le a$$
$$b_8 + b_9 + b_{11} + b_{12} + b_{14} + b_{15} \le a$$
$$b_1 + b_2 + b_{14} + b_{15} \le a$$

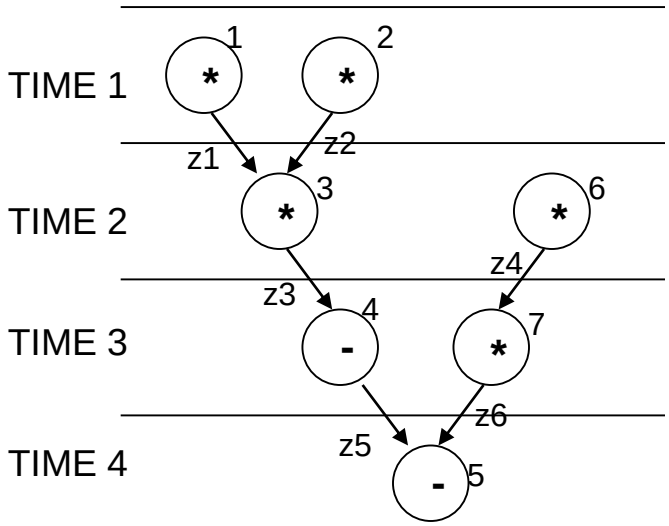
Example – Dynamic Port

- One port $a = 1$:
  - $\{ b_2 , b_4 , b_8 \}$ non-zero
  - 3 variables stored: v_2 , v_4 , v_8
- Two ports $a = 2$:
  - 6 variables stored: $v_2 , v_4 , v_5 , v_{10} , v_{12}, v_{14}$
- Three ports $a = 3$:
  - 9 variables stored: $v_1 , v_2 , v_4 , v_6 , v_8 , v_{10} , v_{12} , v_{13} , v_{14}$

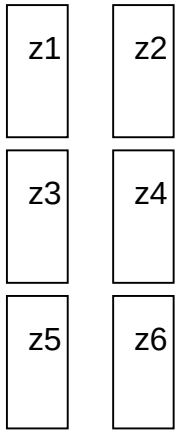
Bus Sharing and Binding

- Find the *minimum number of busses* to accommodate all data transfer
- Find the *maximum number of data transfers* for a fixed number of busses
- Similar to multi-port memory binding problem
- ILP formulation or heuristic algorithms

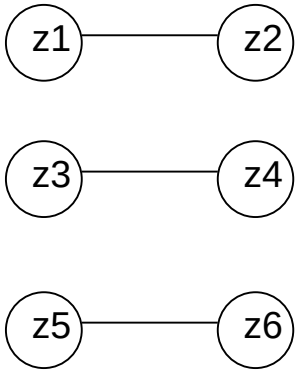
Example



(a)



(b)



(c)

- One bus:
  - 3 variables can be transferred
- Two busses:
  - All variables can be transferred

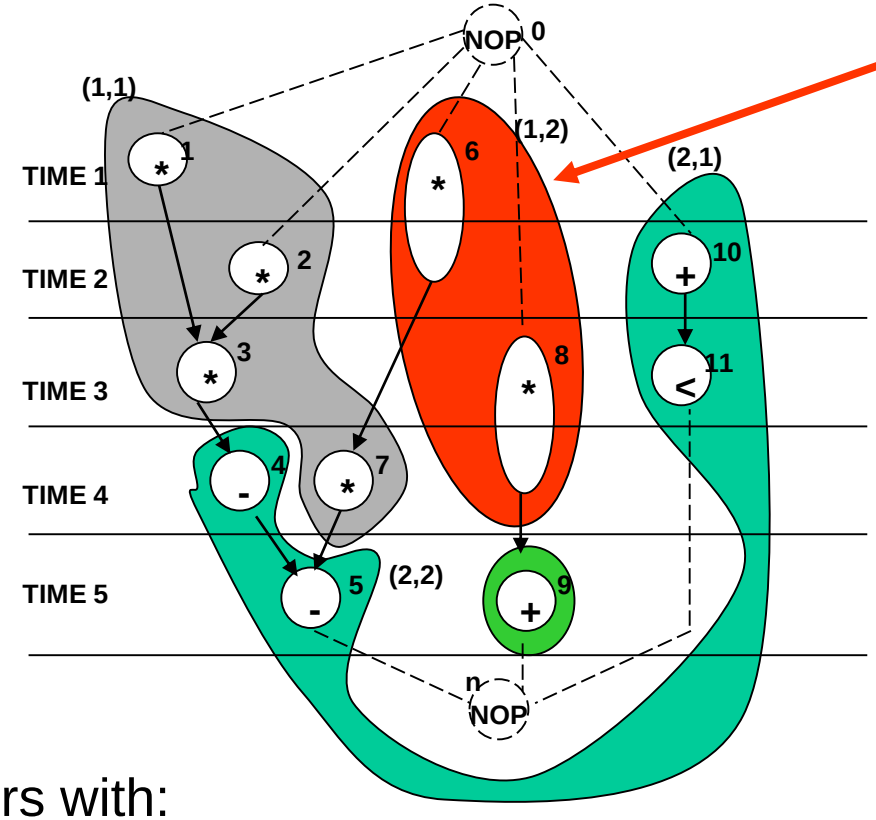
Module Selection Problem

- Extension of resource sharing
  - Library of resources
  - More than one resource per type
- Example:
  - Ripple-carry adder
  - Carry-look-ahead adder
- Resource modeling:
  - Resource *subtypes* with
    - (area, delay) parameters

Module Selection Solution

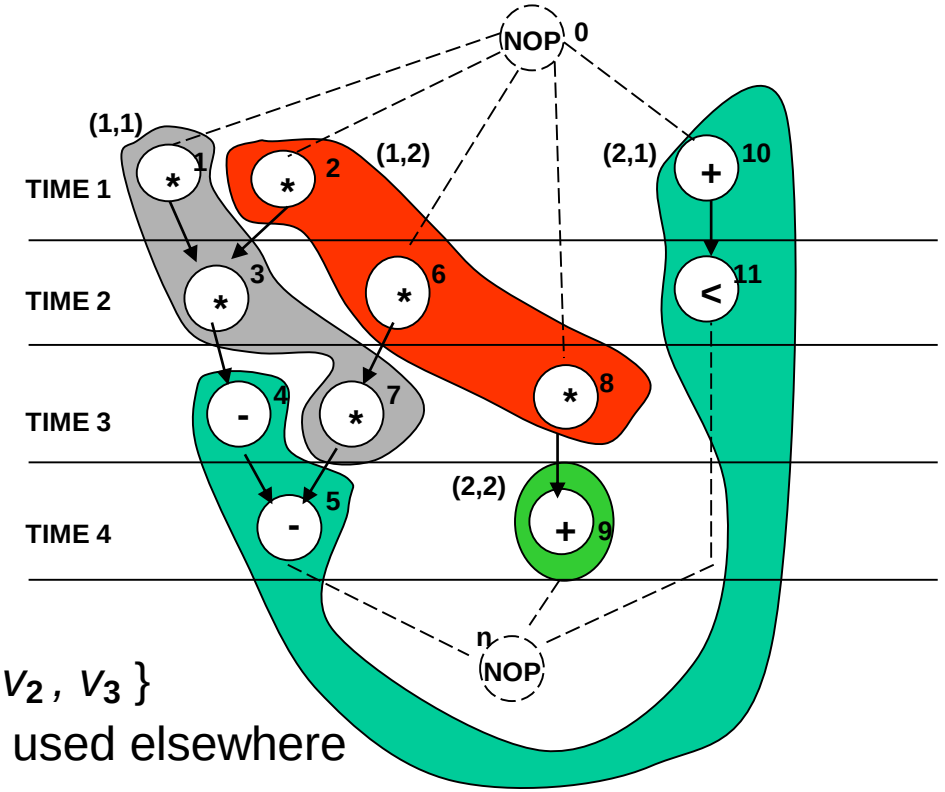
- ILP formulation:
  - Decision variables
    - Select resource sub-type
    - Determine (*area, delay*)
- Heuristic algorithm
  - Determine *minimum latency* with fastest resource subtypes
  - Recover area by using slower resources on non-critical paths

Example – Module Selection



- Multipliers with:
  - (Area, delay) = (5,1) and (2,2)
- Latency bound of 5

Example – Module Selection

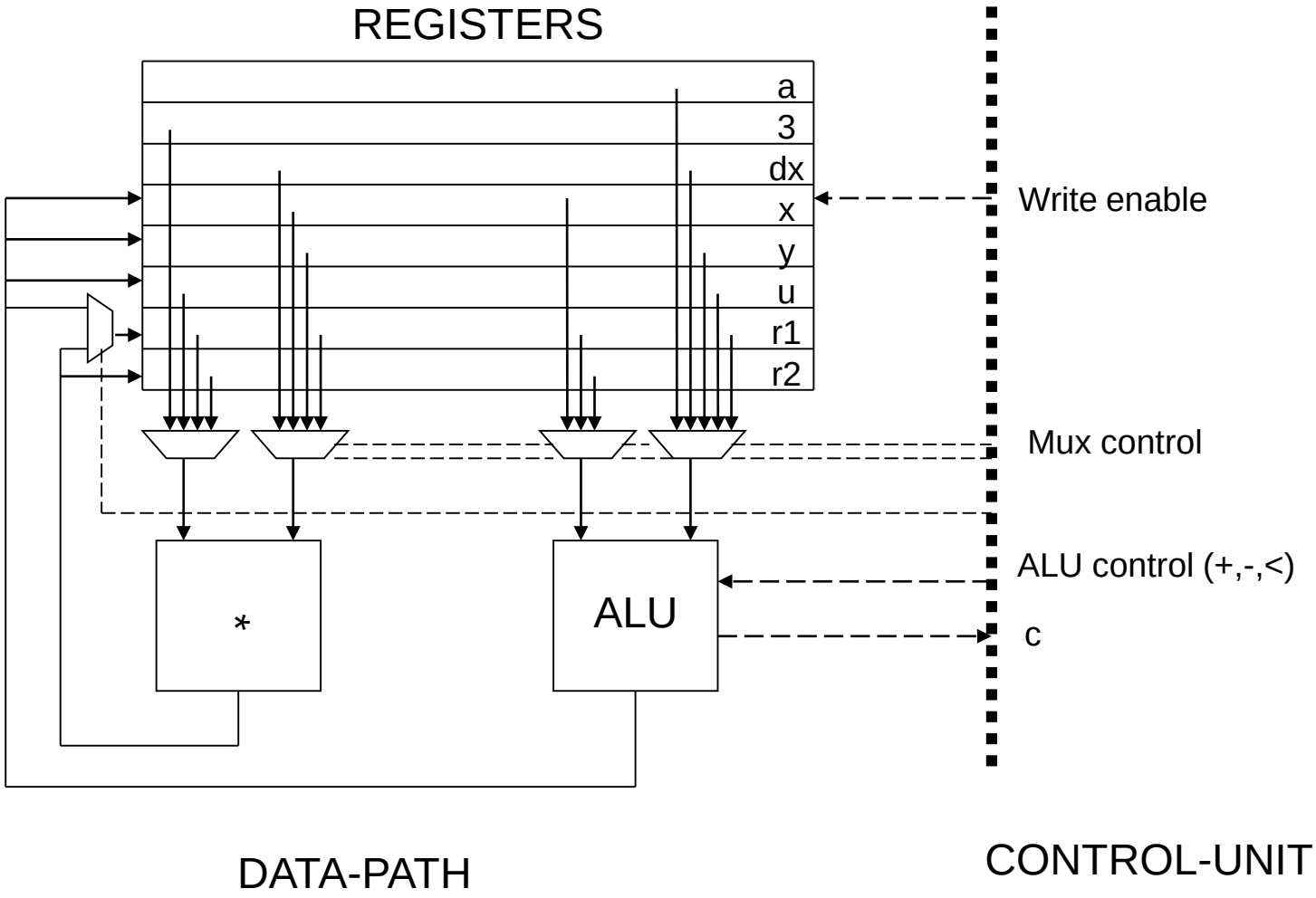


- Latency bound of 4
  - Fast multipliers for { v_1 , v_2 , v_3 }
  - Slower multiplier can be used elsewhere
    - Less sharing
- Minimum-latency design uses fast multipliers only
  - Impossible to use slow multipliers

Data Path Synthesis

- Applied after resource binding
- Connectivity synthesis:
  - Connection of resources to *multiplexers busses* and *registers*
  - Control unit interface
  - I/O ports
- Physical data path synthesis
  - Specific techniques for regular datapath design
    - Regularity extraction

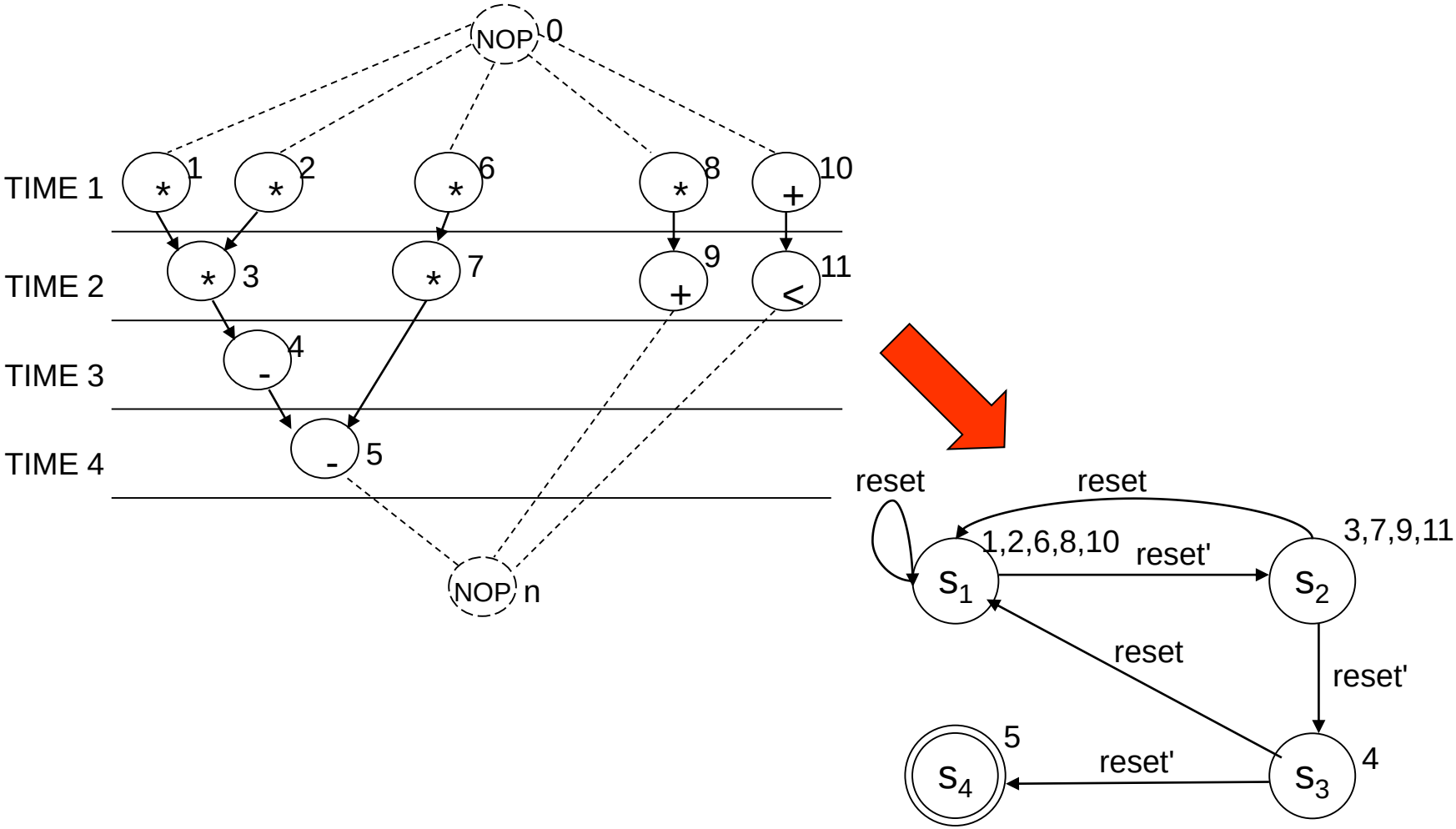
Example – Data Path



Control Synthesis

- Synthesis of the control unit
- Logic model:
  - Synchronous FSM
- Physical implementation:
  - Hard-wired or distributed FSM
  - Microcode

Example – Control



Summary

- Resource sharing is reducible to vertex coloring or to clique covering:
  - Simple for flattened graphs
  - Intractable for imperfect graphs, but still easy in practice
  - Resource sharing has several extensions:
    - Module selection
- Data path design and control synthesis are conceptually simple but still important steps
  - Generated data paths are interconnections of blocks
  - Control is one or more finite-state machines


WWW.MJJQ.COM TRAVEL PHOTOGRAPHY WALLPAPERS ©2006.02