ECE 667

Spring 2013

Synthesis and Verification of Digital Circuits

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)

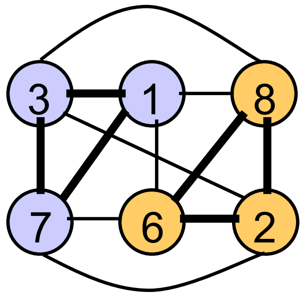
- Allocation = resource binding
  - Spatial mapping between operations and resources
  - Operators can be dedicated or generic (shared)
  - Operators and registers need to be allocated
- Sharing
  - Assignment of a resource to more than one operation
- Constrained resource binding
  - Resource-dominated 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
  - (v_i, v_j) 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, $\kappa(G_+(V,E))$

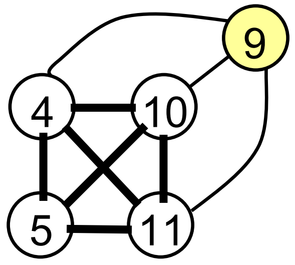
Compatibility Graph $G_+(V,E)$

- Minimum Clique covers in $G_+(V,E)$



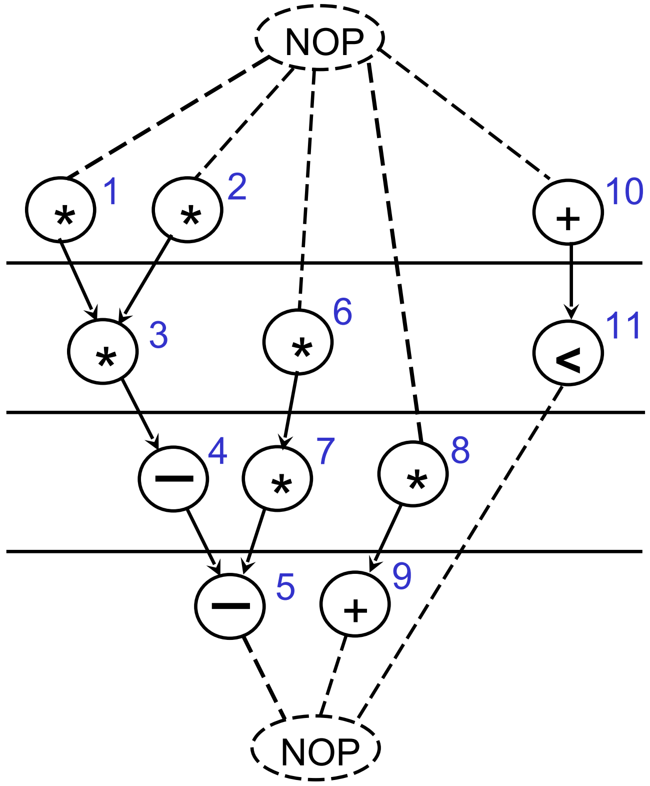
MULT

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



ALU

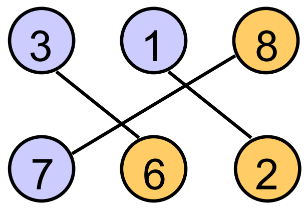
$\kappa(G_+(V,E)) = 2$



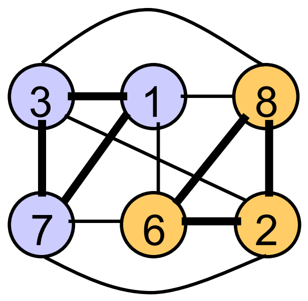
Conflict Graph $G_-(V,E)$

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

MULT



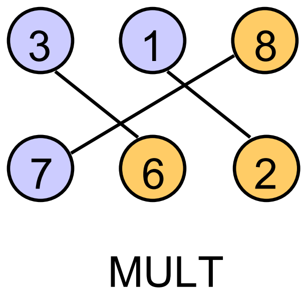
Conflict graph $G_-(V,E)$

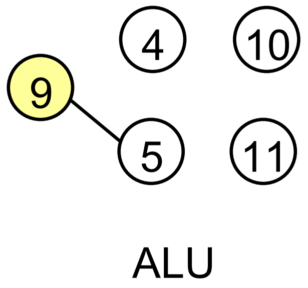


Compatibility graph $G_+(V,E)$

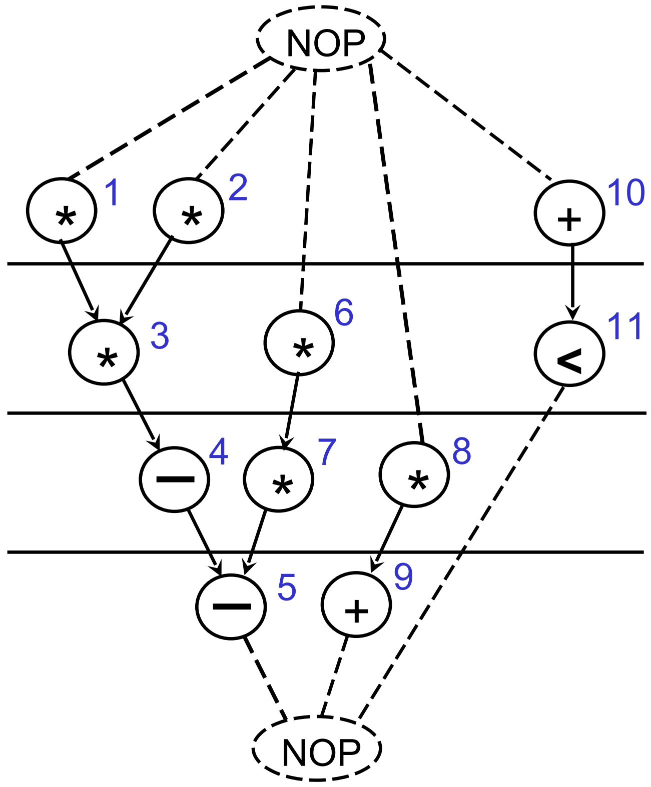
Conflict Graph $G_-(V,E)$ - Example

- Chromatic numbers in $G_-(V,E)$



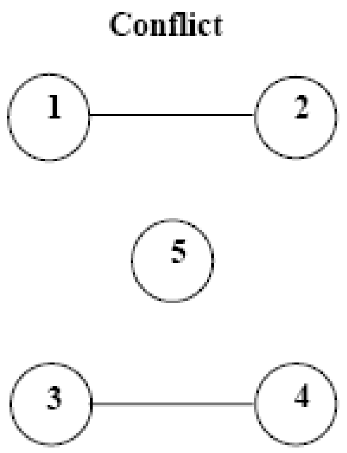


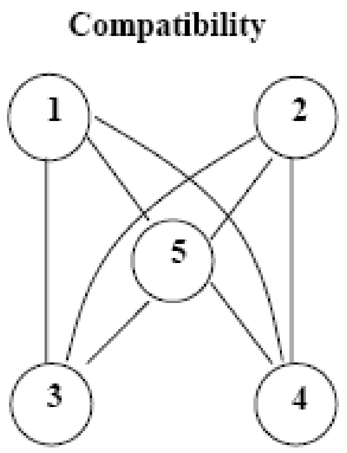
$\chi(G_-(V,E)) = 2$ $\chi(G_-(V,E)) = 2$

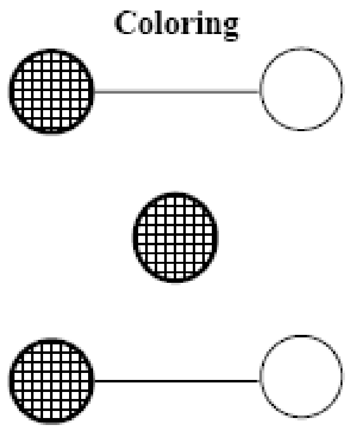


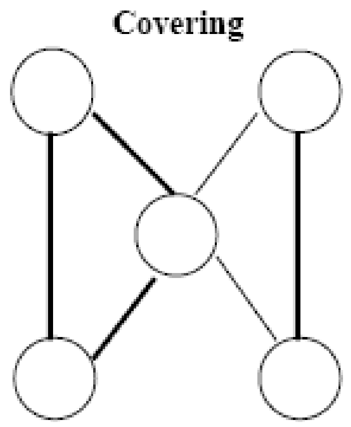
Clique vs Coloring - Example

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









ALU1: 1,3,5

ALU2: 2,4

Special Graphs

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

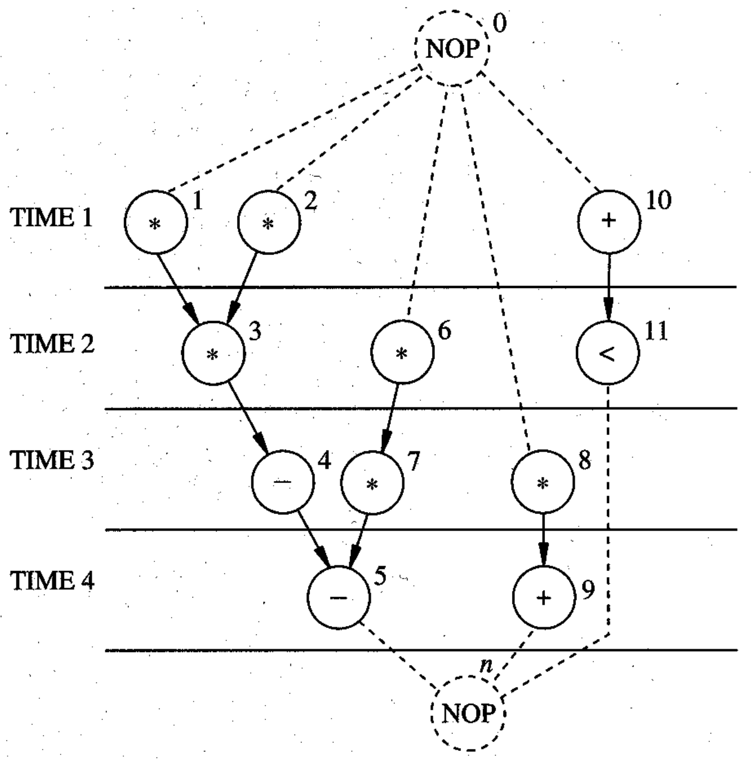
    $(v_i, v_j) \in F$ *and* $(v_j, v_k) \in F \Rightarrow (v_i, v_k) \in F$
- *Interval graph*
  - Vertices correspond to *intervals*
  - Edges correspond to interval *intersections*
  - Subset of *chordal* graphs
    - Every loop with more than 3 edges has a chord

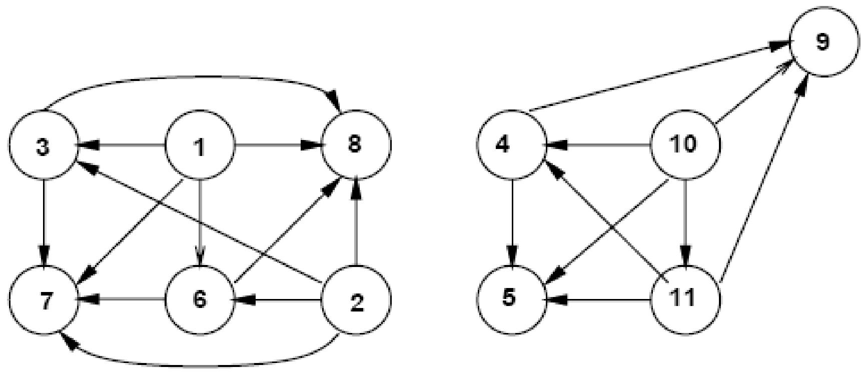
- The compatibility/conflict graphs have special properties
  - *Compatibility* $\Rightarrow$ comparability graph
  - *Conflict* $\Rightarrow$ interval graph

Comparability Graph

Representation of compatible relations

- Note: sequencing graph is assumed to be scheduled



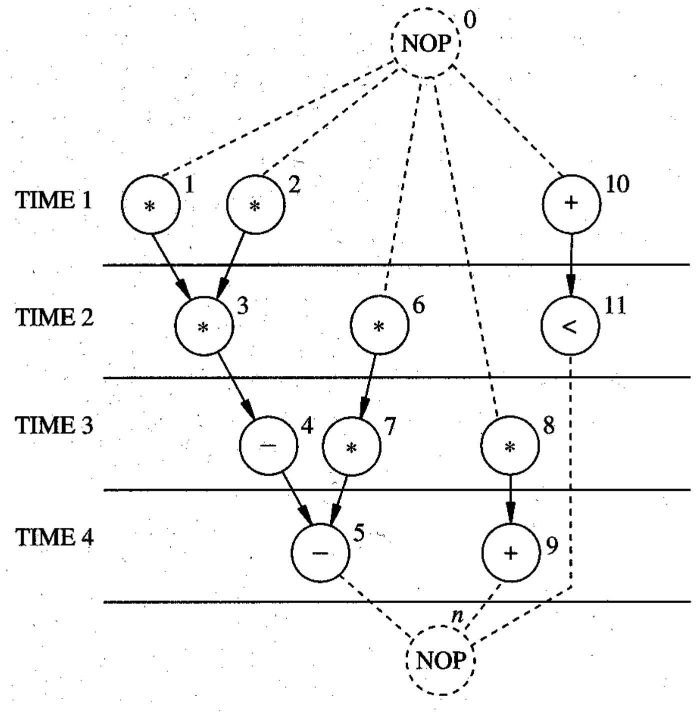


Note the orientation of edges, compare to compatibility graph.

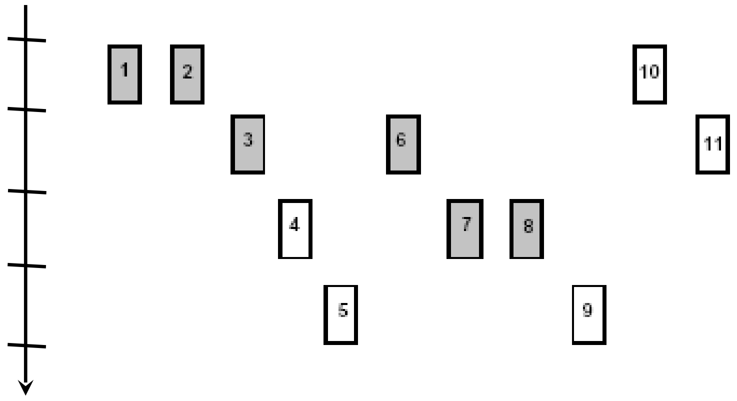
Interval Graph Representation

Interval representation of conflicting relation

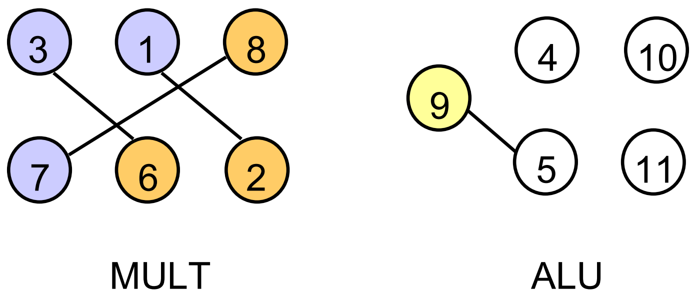
- Note: sequencing graph is assumed to be scheduled



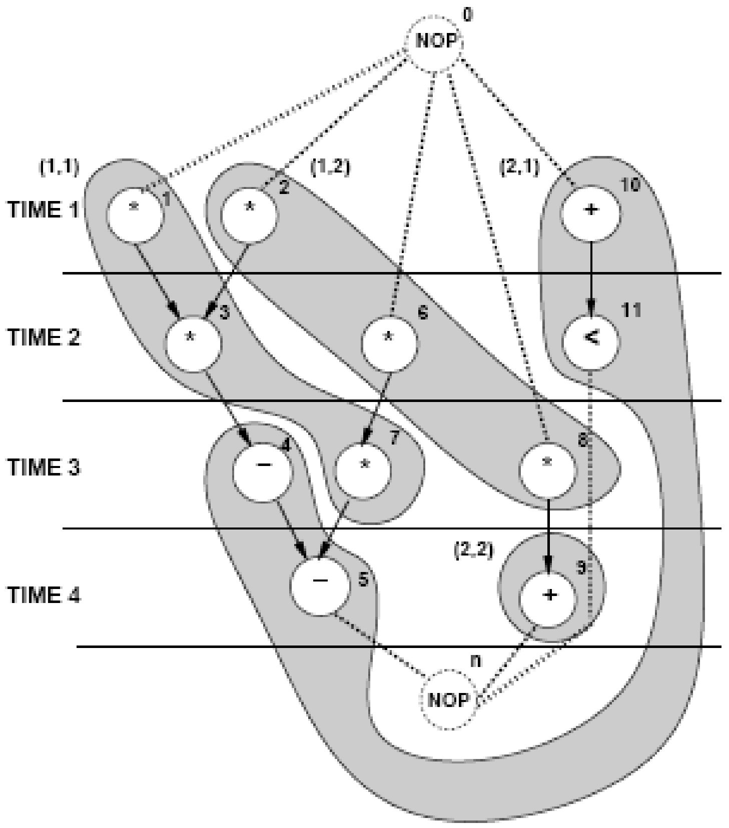
Intervals with "*Left*" and "*Right*" coordinates

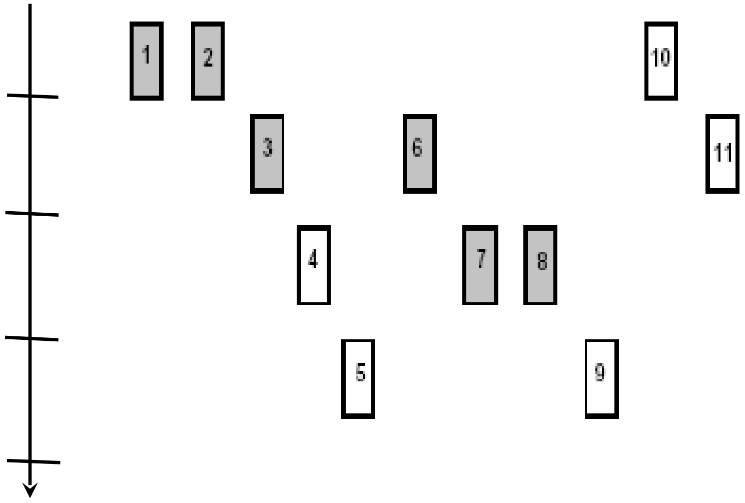


Compare with conflict graphs:



Operation Binding - Solution





Mult1: { 1, 3, 7 }, Mult2: { 2, 6, 8 }

ALU1: { 10,11, 4,5 }, ALU2: { 9 }

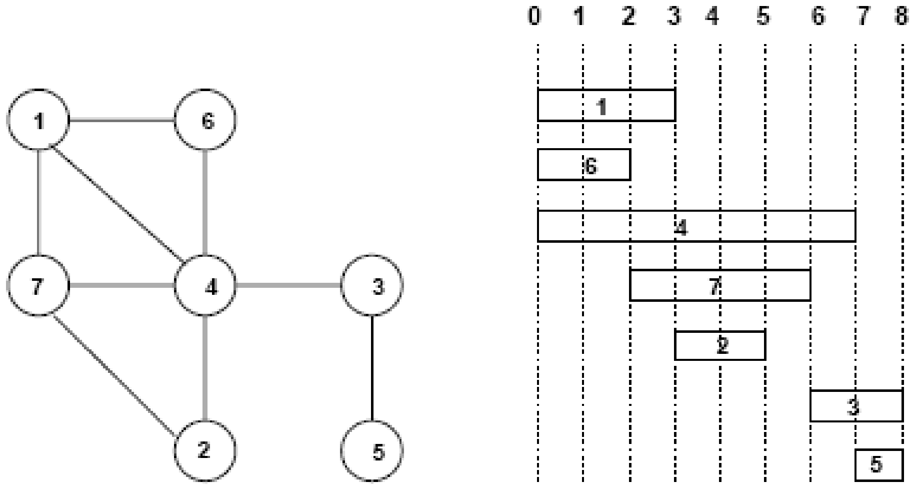
Left-Edge Algorithm

- Input
  - Set of intervals sorted with *left* and *right* edge coordinates
- Algorithm
  - Sort intervals by their *left* edge coordinates
  - Assign non-overlapping intervals to first track (color) using the sorted list
  - When possible intervals are exhausted, increase track (color) counter and repeat.
- Efficiency
  - Simple, polynomial time algorithm

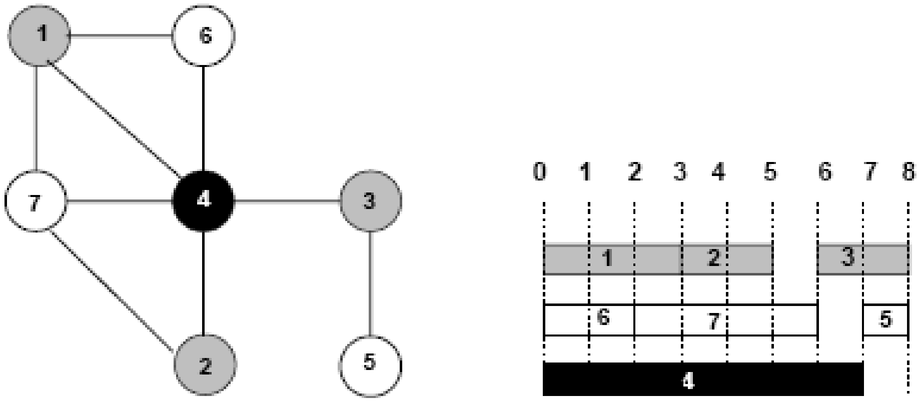
Left-Edge Algorithm

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

Left-Edge Algorithm - Example



Input



Solution

ILP Formulation of Operation Binding

- Boolean variables b_{ir}

$$b_{ir} = \begin{cases} 1 & \text{if operator } i \text{ is bound to resource } r \\ 0 & \textit{otherwise} \end{cases}$$

- Boolean variables x_{il}

x_{il} = 1 if operation *i* is scheduled to start at step *l*

- Each operation is bound to *one resource*

$$\sum_{r=1}^{a} b_{ir} = 1 \quad \forall i$$ (*a* = limit on resource *r*)

- At each step l, *at most one operation* can be executing for a given resource (horizontal constraint)

$$\sum_{i=1}^{n_{ops}} b_{ir} \sum_{m=l-d_i+1}^{l} x_{im} \leq 1 \quad \forall l \ \forall r$$

Operation Binding - Solution

- Equations for two multipliers:

  $b_{i1} + b_{i2} = 1, \; i=\{1,2,3,6,7,8\}$

MULT 1:

$$\sum_{i=\{1,2,3,6,7,8\}} b_{i1} x_{il} \le 1, \quad l = 1,2,\ldots,5$$

MULT 2:

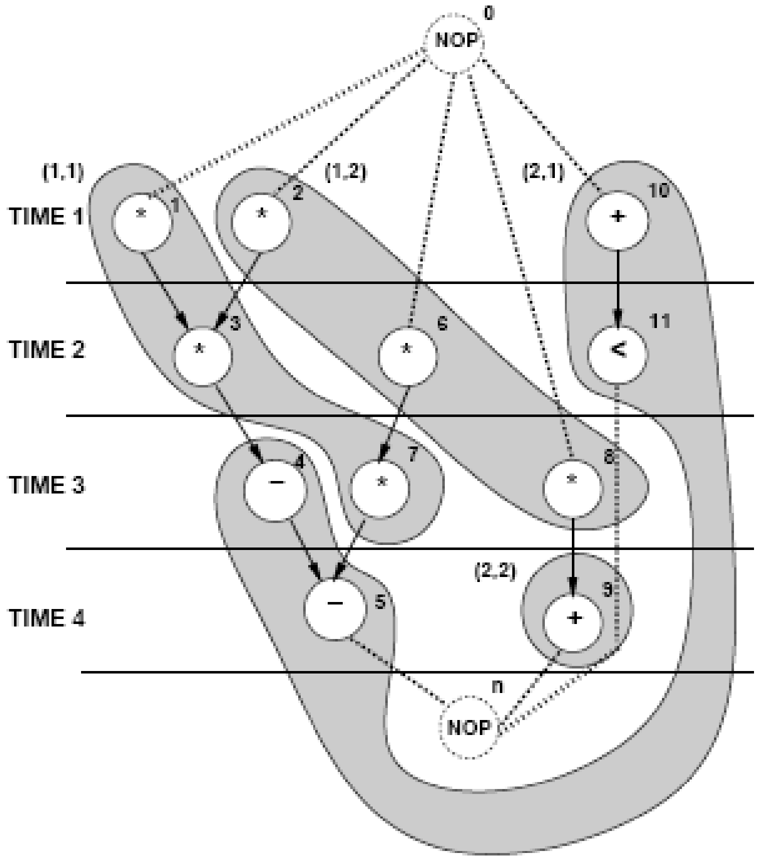
$$\sum_{i=\{1,2,3,6,7,8\}} b_{i2} x_{il} \le 1, \quad l = 1,2,\ldots,5$$

- Solution:

  $b_{11} = b_{31} = b_{71} = 1$

  $b_{22} = b_{62} = b_{82} = 1$

  all other $b_{ij} = 0$

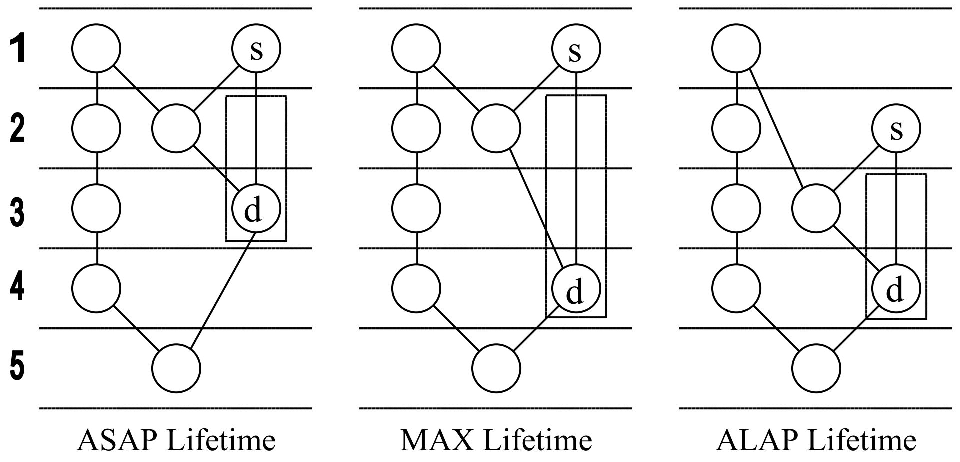


Register Binding Problem

- 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

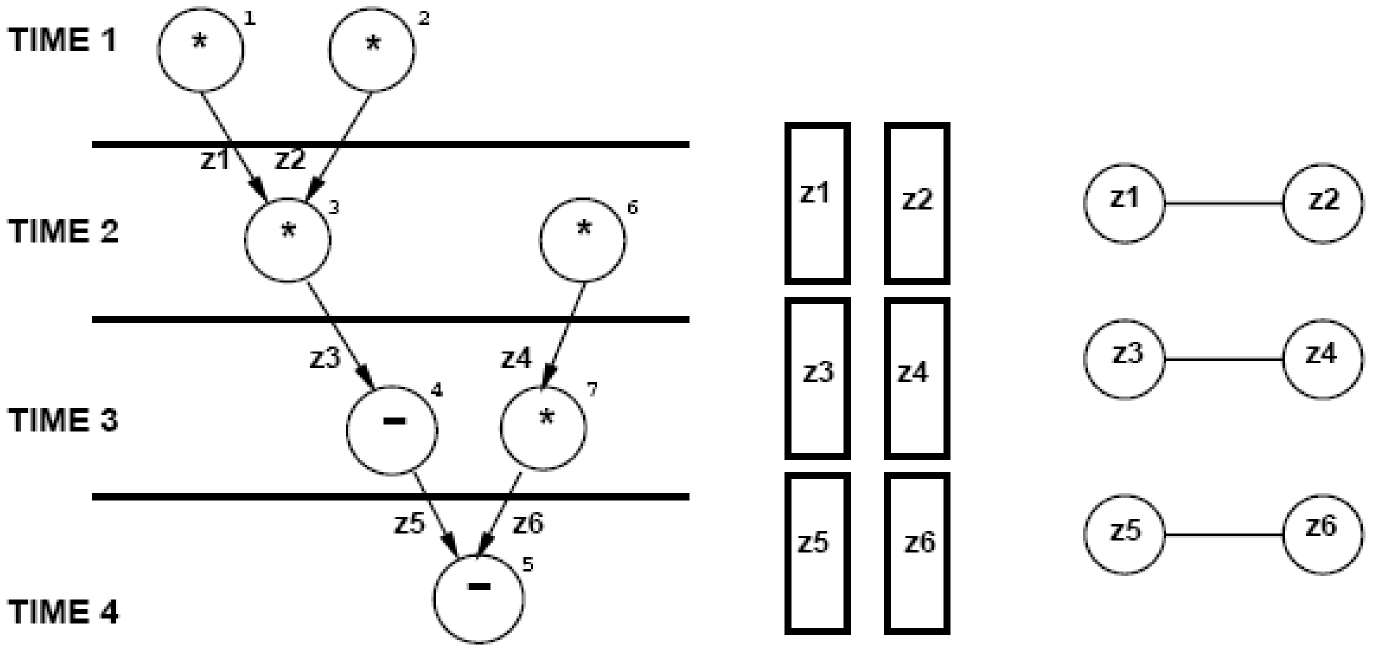


Register Binding Problem

- 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)

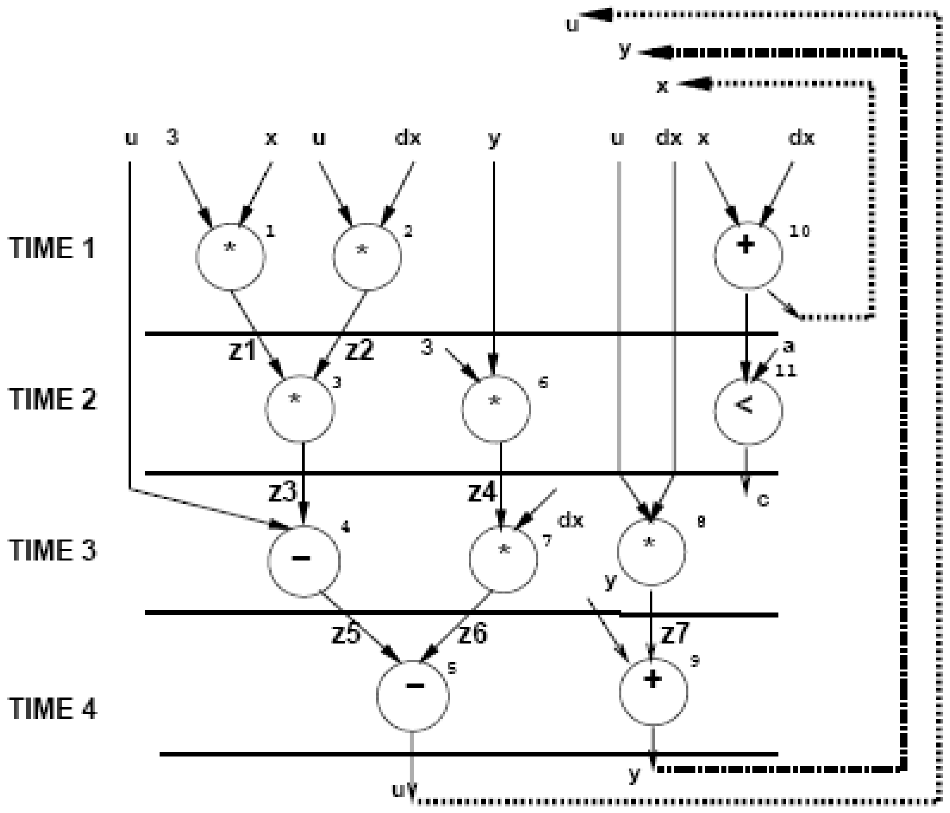
Register Binding Problem – Example 1

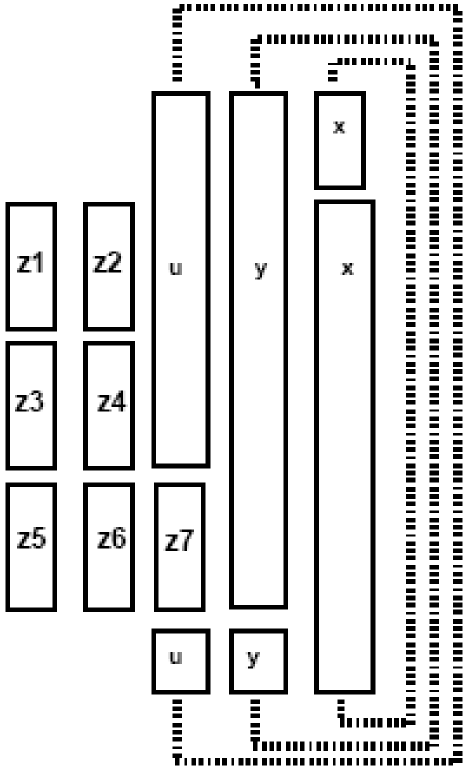
- Non-iterative designs
  - Create variable *compatibility* graph or *conflict* graph
  - Use left-edge algorithm to minimize the number of registers



Register Binding – Example 2

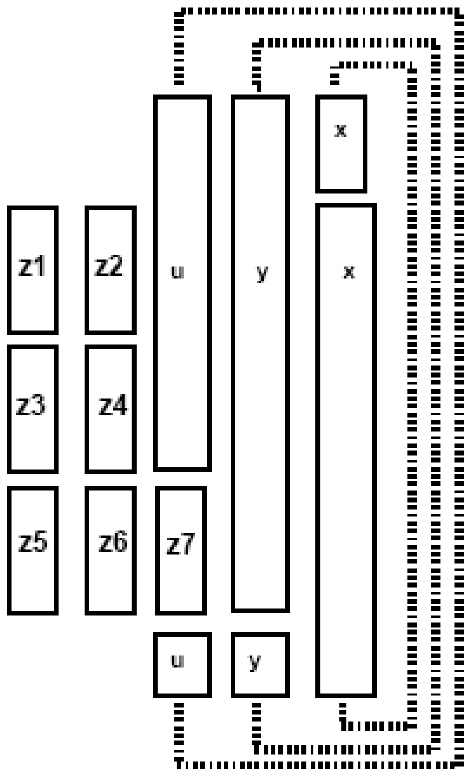
- Iterative designs
  - Sequencing graph and variable lifetimes



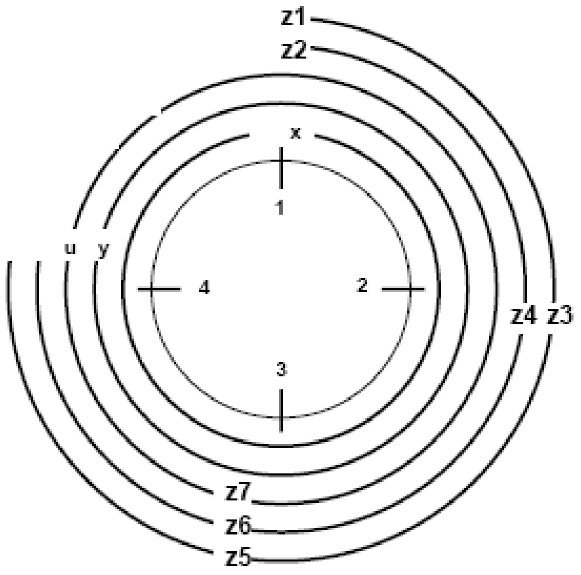


Circular Arc Conflict Graph

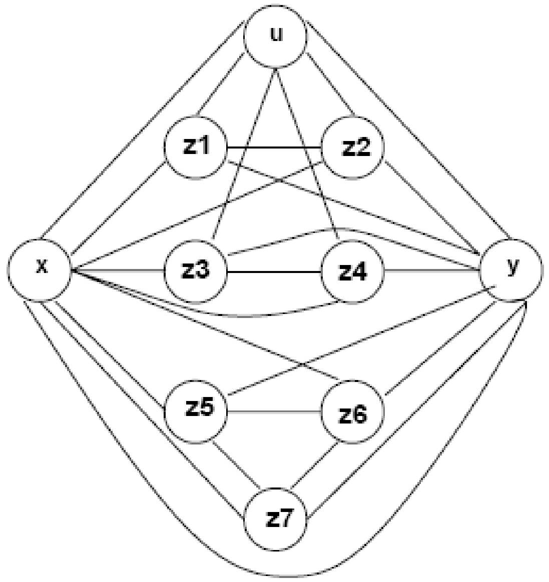
- Overlapping lifetimes of variables represent conflicts



Variable lifetimes



Variable lifetimes as arcs



Circular-arc conflict graph

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

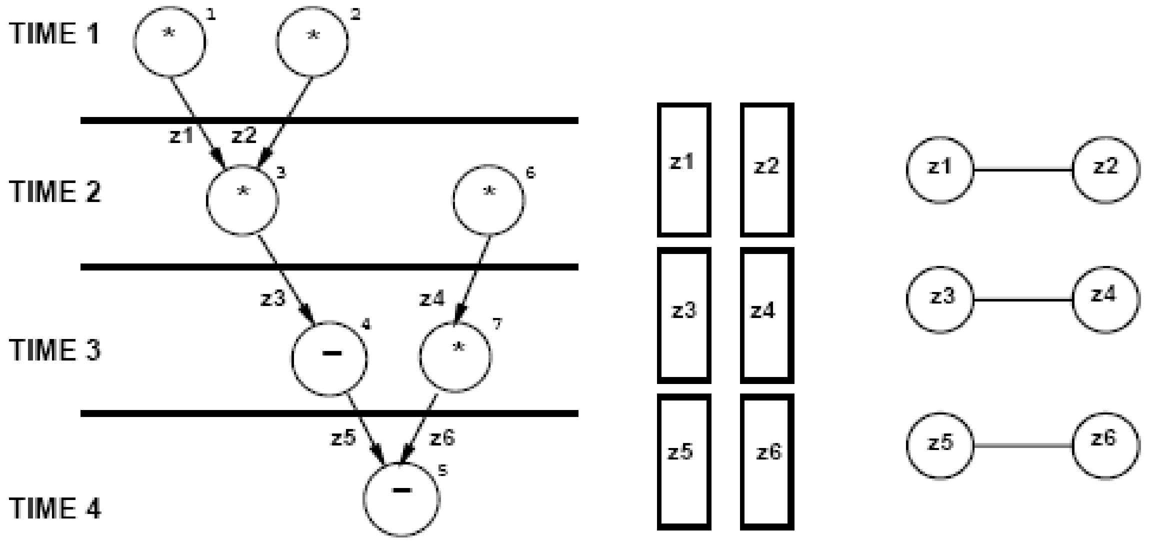
Bus Sharing and Binding

- Buses act as *transfer resources*
  - See architecture produced by GAUT
- Find the *minimum number of buses* to accommodate all data transfer
- Find the *maximum number of data transfers* for a fixed number of buses

- Similar to memory binding problem
- Possible solutions
  - ILP formulation
  - Heuristic algorithms

Bus Sharing and Binding - Example

- One bus
  - 3 variables
- Two buses
  - All variables can be transferred



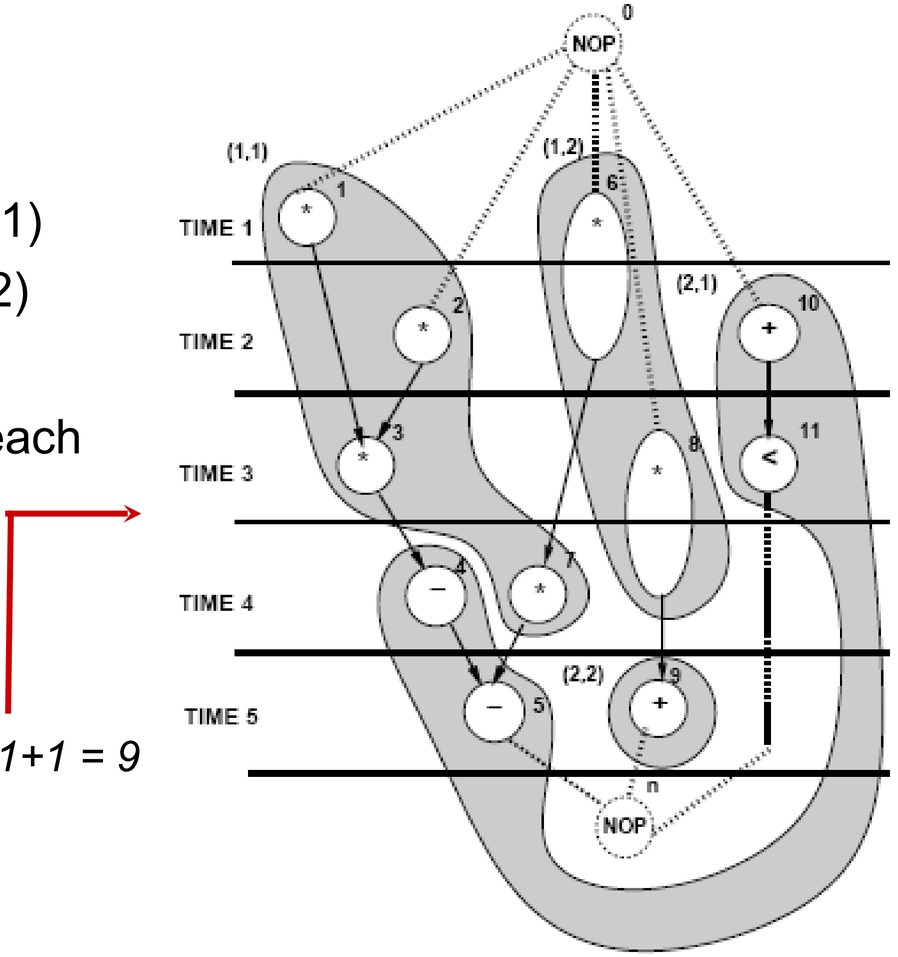
Module Selection Problem

- Resource-type (module) selection problem
  - Generalization of the binding problem
- Library of resources:
  - More than one resource per type
- Example:
  - Ripple-carry adder vs. carry look-ahead adder
- Resource modeling
  - Resource subtypes with *(area, delay)* parameters
- Solution
  - ILP formulation:
    - Decision variables: select resource subtype,
      determine *(area, delay)*
  - Heuristic algorithms:
    - Determine minimum latency with fastest resource subtypes
    - Recover area by using slower resources on non-critical paths

Module Selection - Example 1

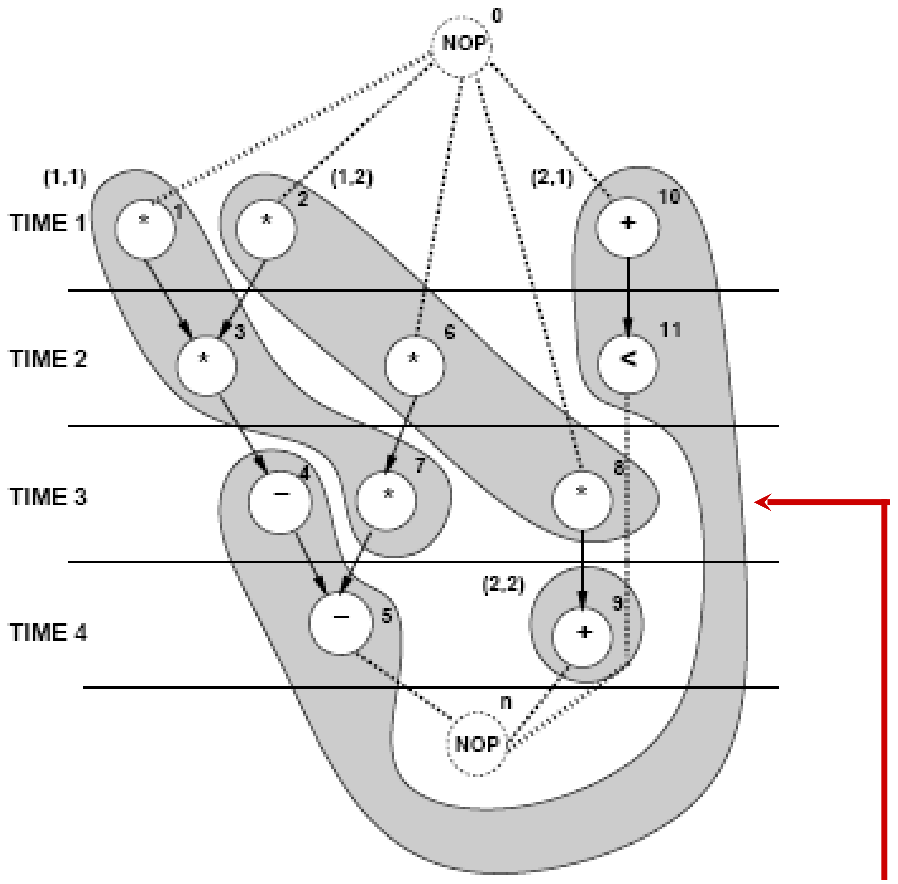
- Latency bound of 5
- Two multipliers available:
  - MULT1 with (*area, delay*) = (5,1)
  - MULT2 with (*area, delay*) = (2,2)
- Two ALUs available:
  - ALU with (*area, delay*) = (1,1) each

Area = 5+2+1+1 = 9



Module Selection Example 2

- Latency bound of 4
  - Fast multipliers for $\{v_1, v_2, v_3\}$
  - Slower multipliers can be used elsewhere
    - less sharing
- *Minimum latency* design
  - used *fast* multipliers only.
- Area recovery
  - On non-critical paths replace fast (large) multipliers by slow (small) ones



Area = 5+5+1+1=12

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