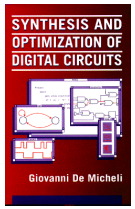
Resource sharing

Giovanni De Micheli

Integrated Systems Laboratory



Module 1

- **Objectives**
  - **Motivation and problem formulation**
  - **Flat and hierarchical graphs**
  - **Functional and memory resources**
  - **Extension to module selection**

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 is well defined**
- **Consider *operation types* independently**
  - **Problem decomposition**
  - **Perform analysis for each resource type**

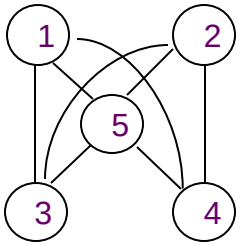
Compatibly and conflicts

- **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	

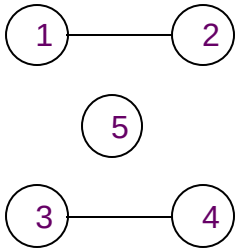
- ***Compatibility*** **graph:**
  - Vertices: operations
  - Edges: compatibility relation

Compatibility graph



- ***Conflict*** **graph:**
  - Complement of compatibility graph

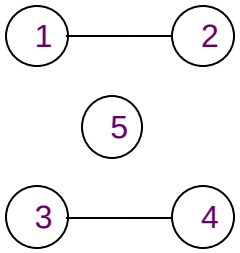
Conflict graph



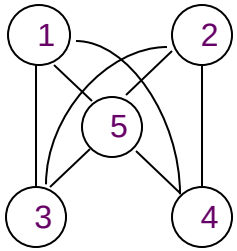
Example

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

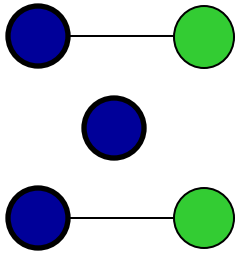
Conflict



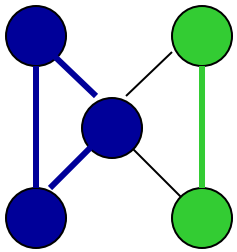
Compatibility



Coloring



Partitioning



ALU1: 1,3,5

ALU2: 2,4

Compatibility and conflicts

- **Compatibility graph:**
  - **Partition the graph into a minimum number of cliques**
  - **Find clique cover number k (G_+)**
- **Conflict graph:**
  - **Color the vertices by a minimum number of colors.**
  - **Find the chromatic number x (G_-)**
- **Intractable problems:**
  - **Heuristic algorithms**

Data-flow graphs
(flat sequencing graphs)

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

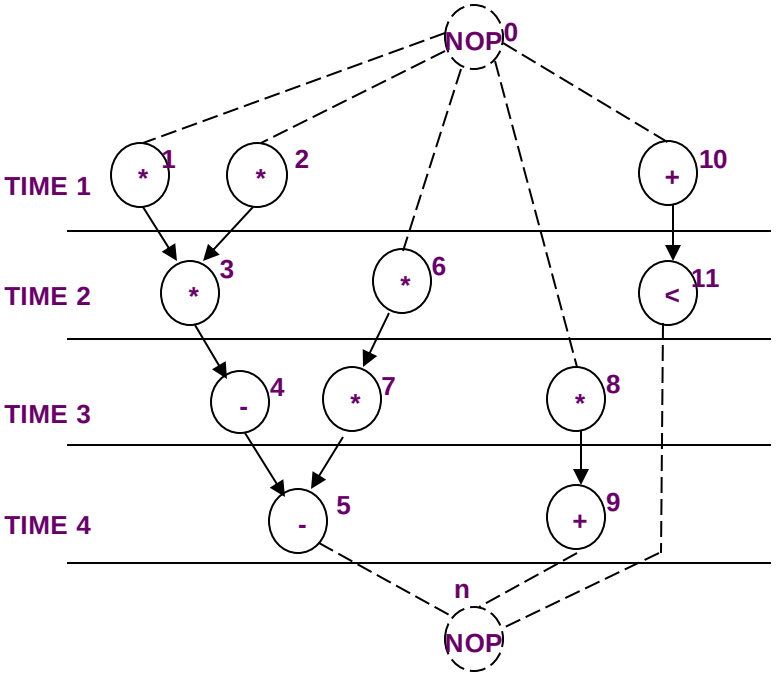
Perfect graphs

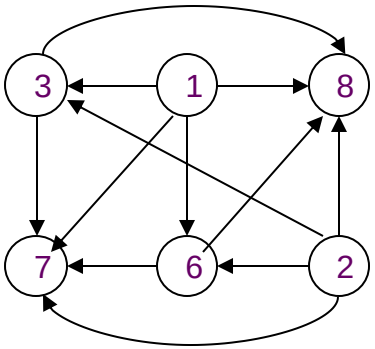
- ***Comparability graph*:**
  - **Graph $G(V, E)$ has an orientation $G(V, F)$ with the transitive property**

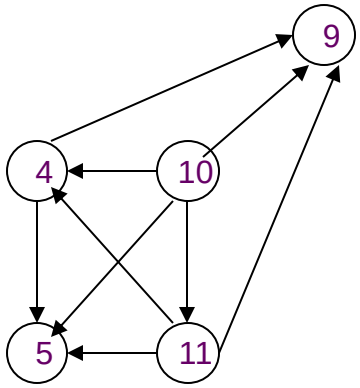
$$(v_i, v_j) \in F \quad \text{and} \quad (v_j, v_k) \in F \rightarrow (v_i, v_k) \in F$$

- ***Interval graph*:**
  - **Vertices correspond to *intervals***
  - **Edges correspond to interval intersection**
  - **Subset of *chordal* graphs**
    - **Every loop with more than three edges has a chord**

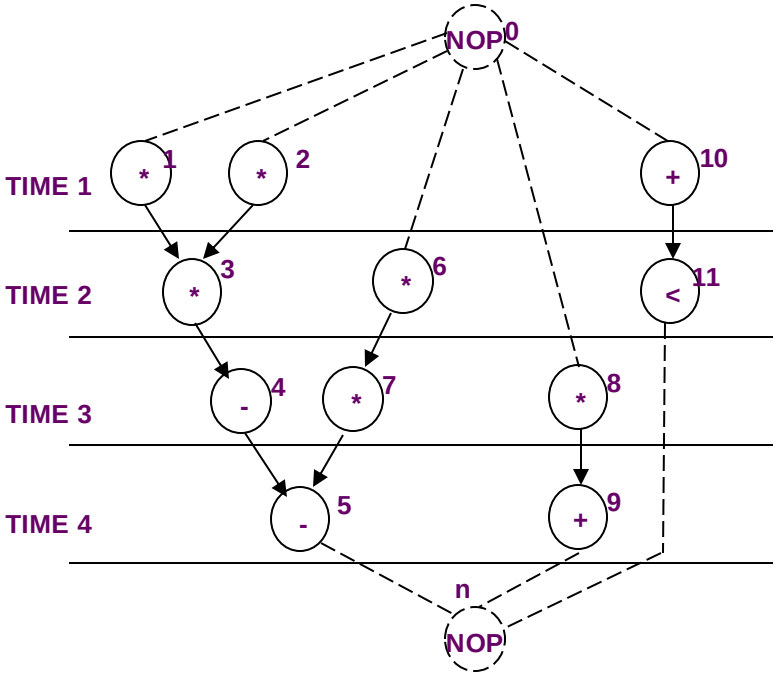
Example

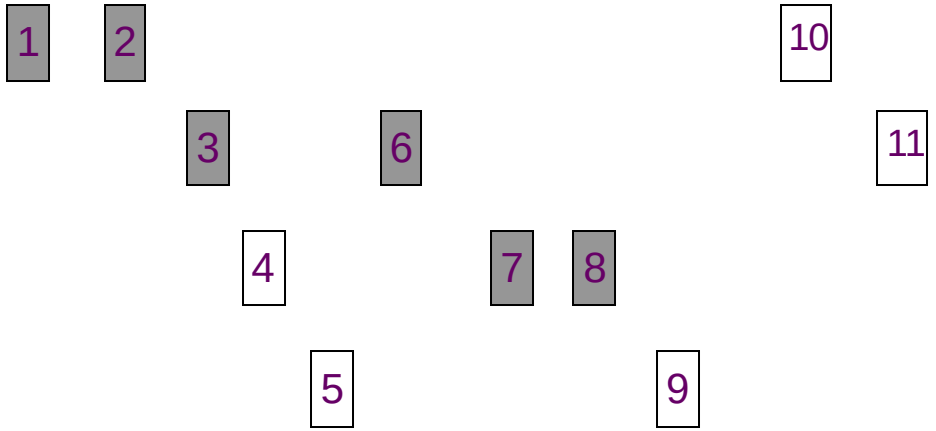

NOP
0
TIME 1
*
1
*
2
+
10
TIME 2
*
3
*
6
<
11
TIME 3
-
4
*
7
*
8
TIME 4
-
5
+
9
n
NOP


3
1
8
7
6
2


9
4
10
5
11

Example

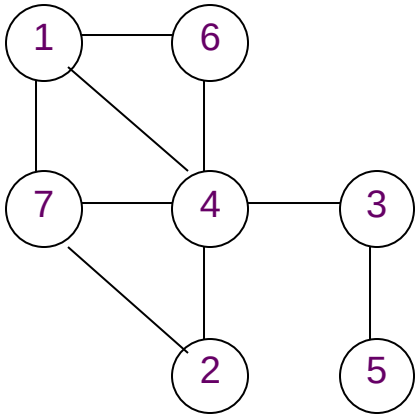

NOP
0
*
1
*
2
+
10
TIME 1
*
3
*
6
<
11
TIME 2
-
4
*
7
*
8
TIME 3
-
5
+
9
TIME 4
n
NOP


1
2
10
3
6
11
4
7
8
5
9

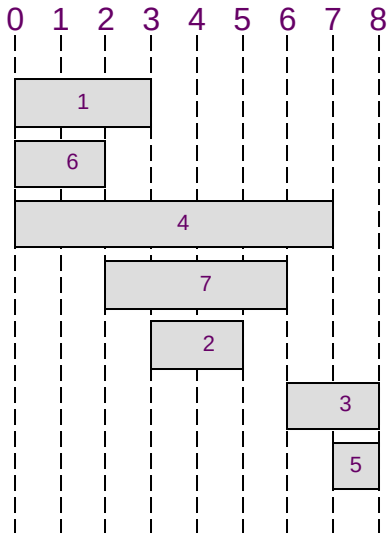
Left-edge algorithm

- **Input:**
  - **Set of intervals with *left* and *right edge***
    - ***Start and Stop times***
  - **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**

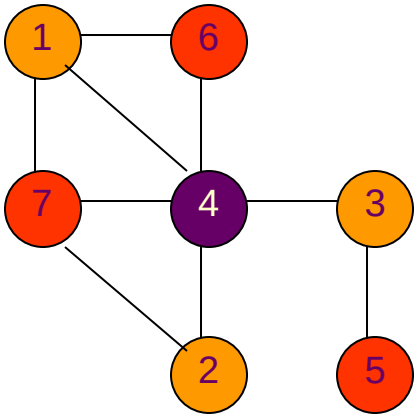
Example



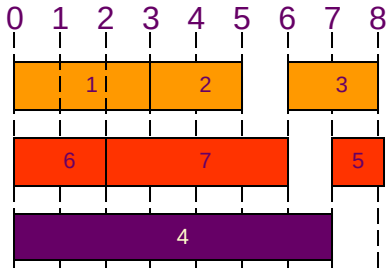
Conflict graph



Intervals



Colored conflict graph



Coloring

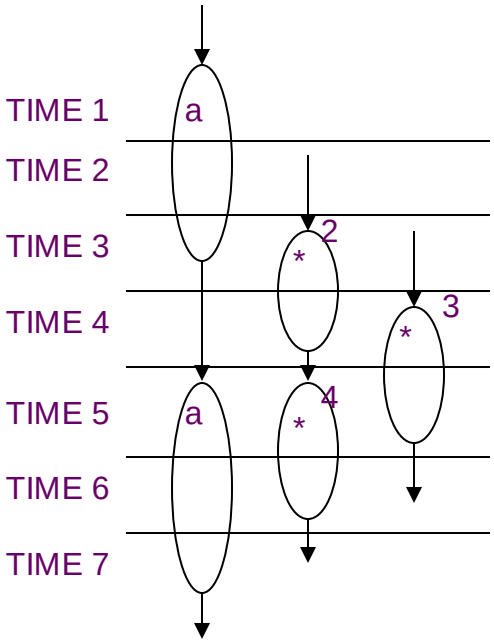
Left-edge algorithm

```
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 ϵ 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;
    }
}
```

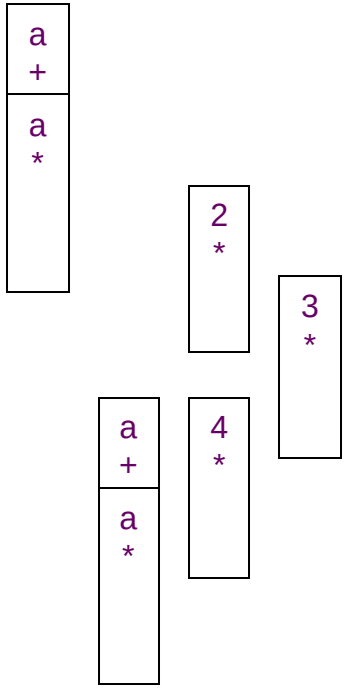
Hierarchical sequencing graphs

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

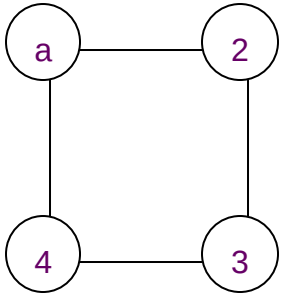
Example


TIME 1
TIME 2
TIME 3
TIME 4
TIME 5
TIME 6
TIME 7
a
2
*
3
*
a
4
*

(a)

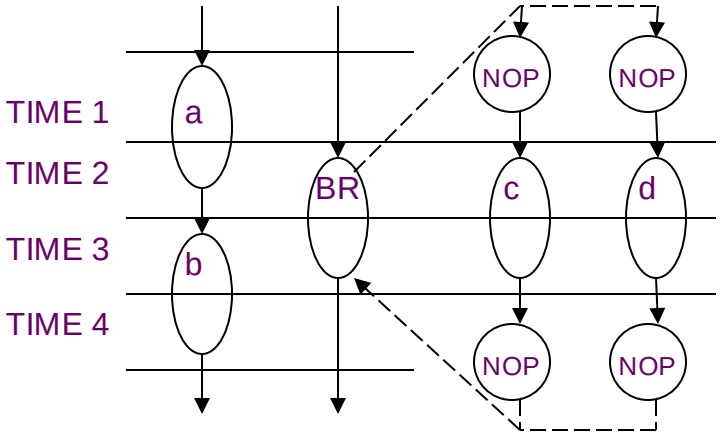

a
+
a
*
2
*
3
*
a
+
a
*
4
*

(b)

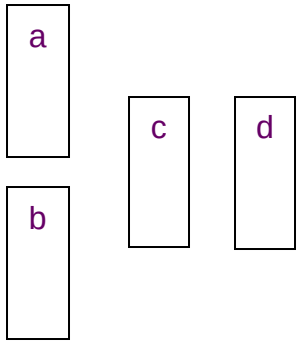

a
2
4
3

(c)

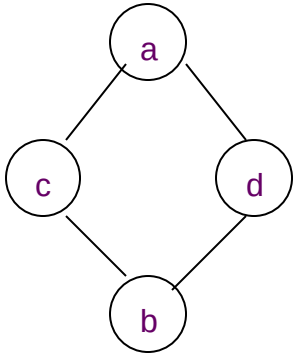
Example



(a)



(b)



(c)

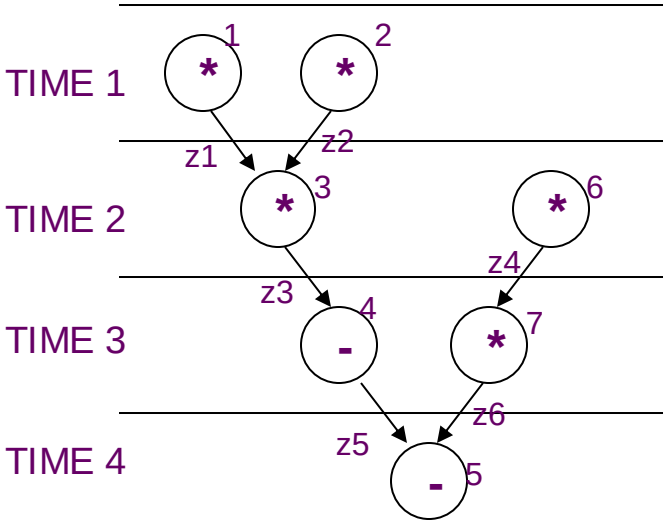
Register binding problem

- **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**

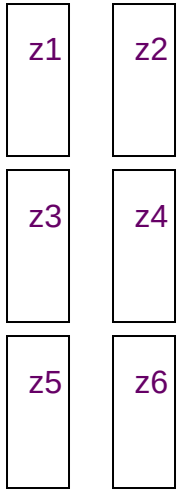
Register sharing in data-flow graphs

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

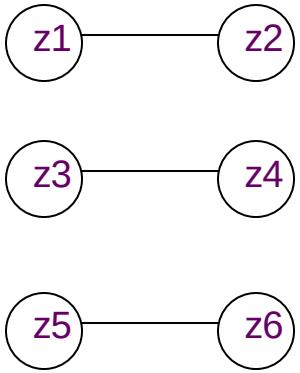
Example


TIME 1
TIME 2
TIME 3
TIME 4
1
2
3
4
5
6
7
*
*
*
*
-
-
z1
z2
z3
z4
z5
z6

(a)


z1
z2
z3
z4
z5
z6

(b)

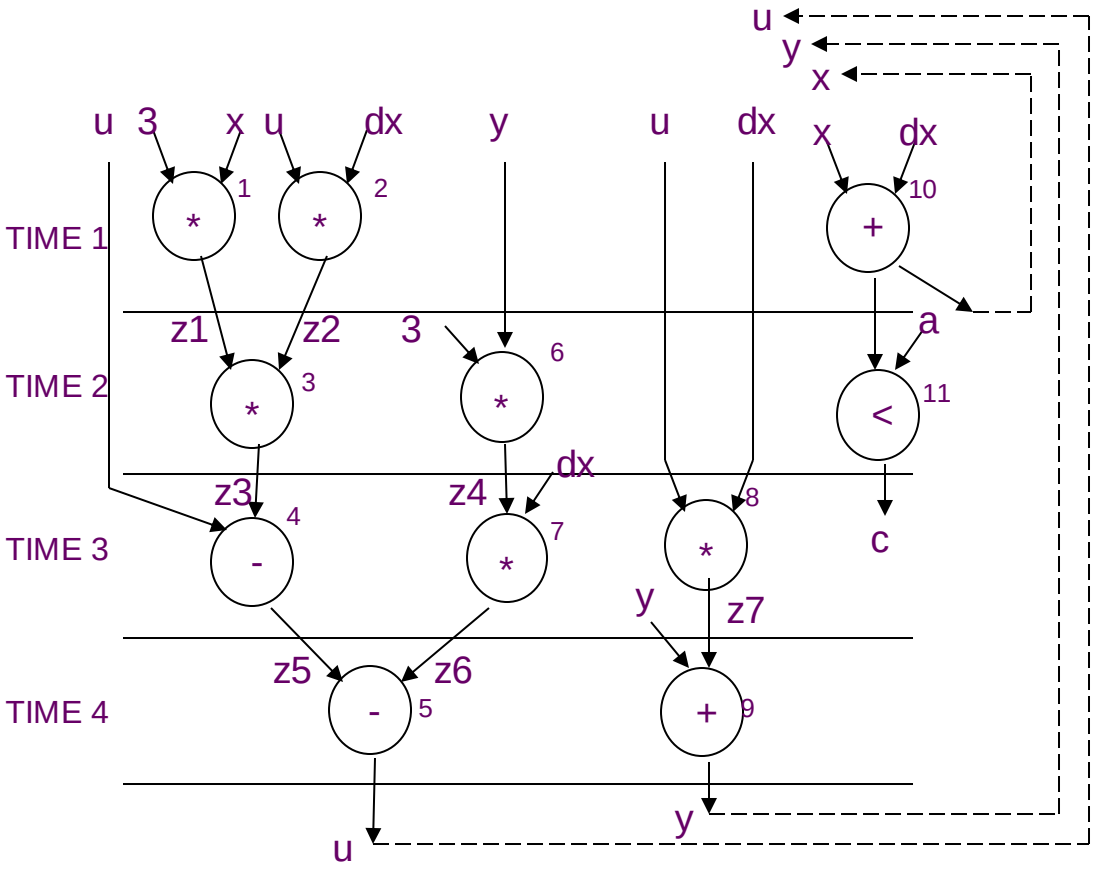

z1
z2
z3
z4
z5
z6

(c)

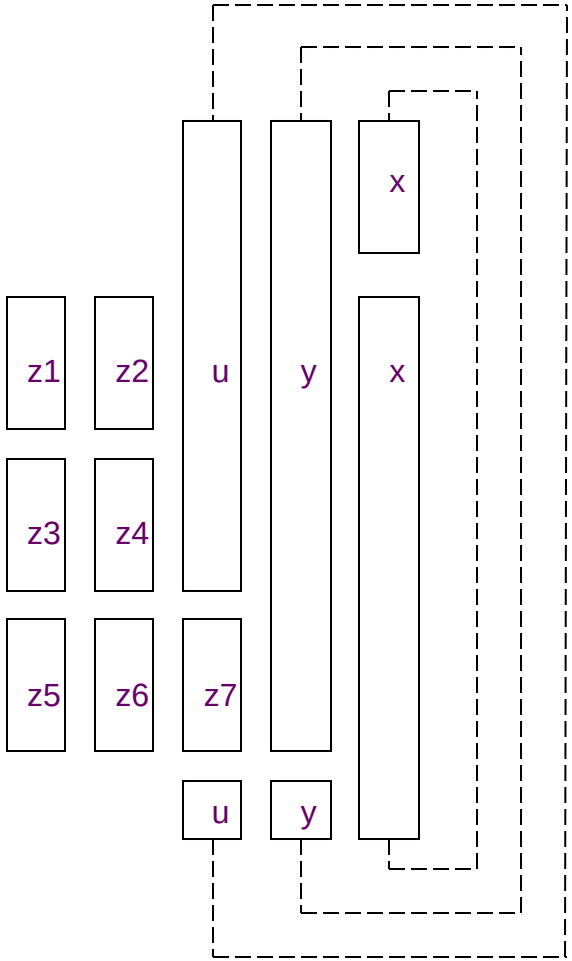
Register sharing general case

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

Example

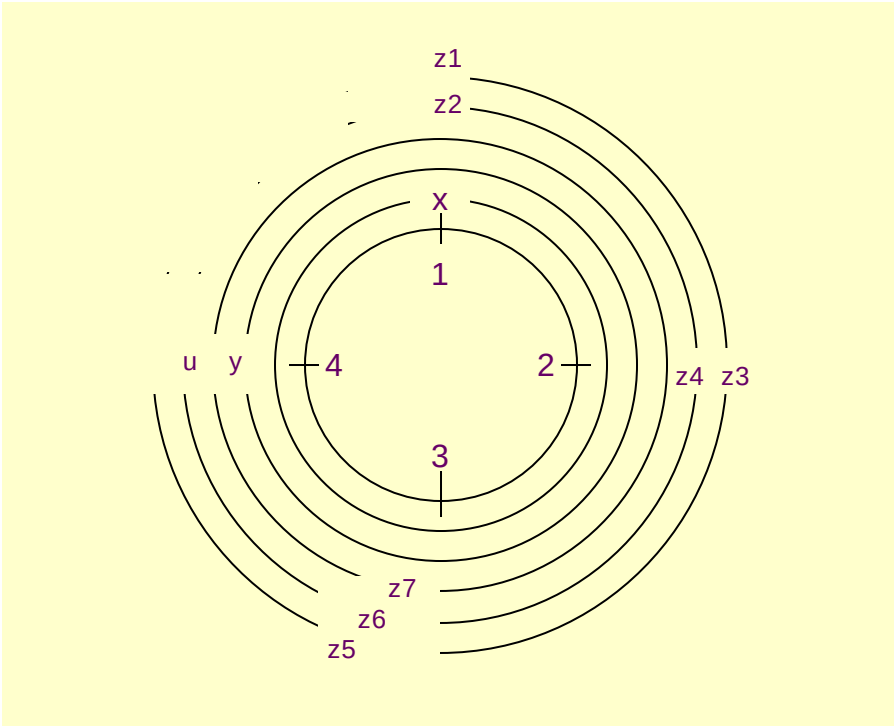

u
y
x
u 3 x u dx y u dx x dx
TIME 1
TIME 2
TIME 3
TIME 4
1
2
10
z1 z2 3 a
3
6
11
dx
z3 z4
4
7
8
c
y z7
z5 z6
5
9
u
y

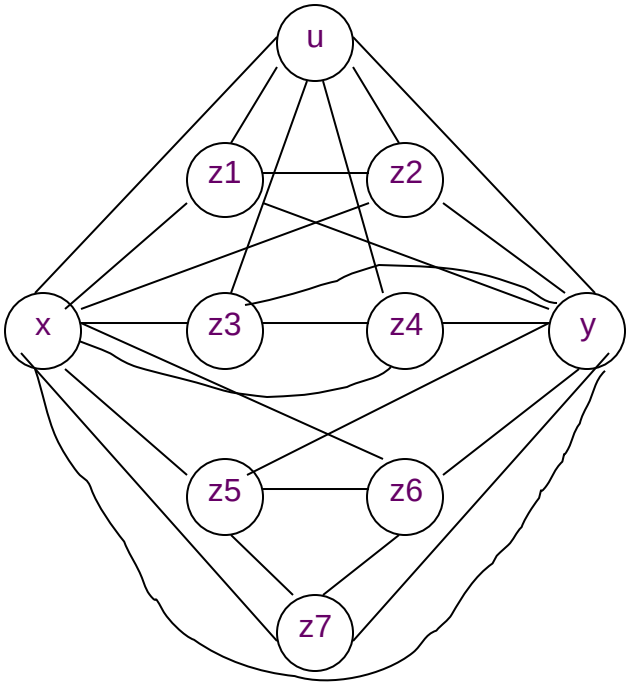
(a)


z1
z2
u
y
x
z3
z4
z5
z6
z7
u
y

(b)

Example
Variable-lifetimes and circular-arc conflict graph

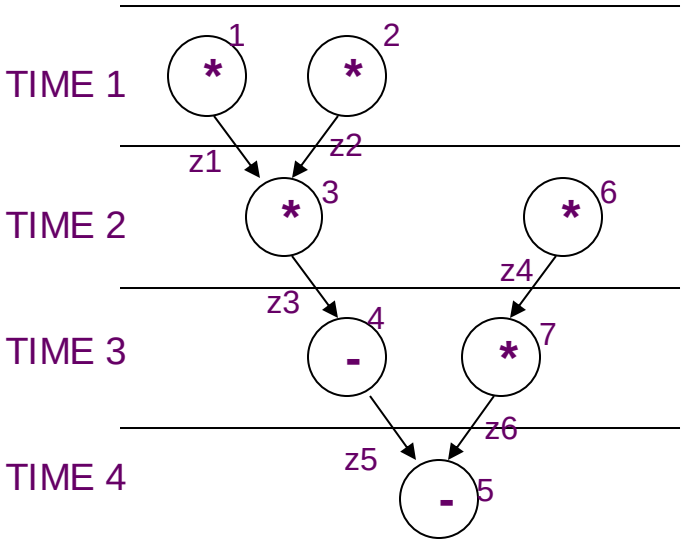




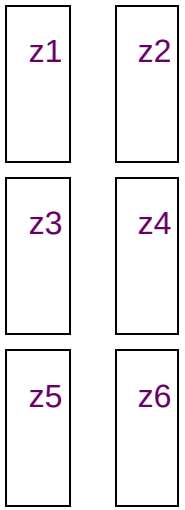
Bus sharing and binding

- 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 memory binding problem

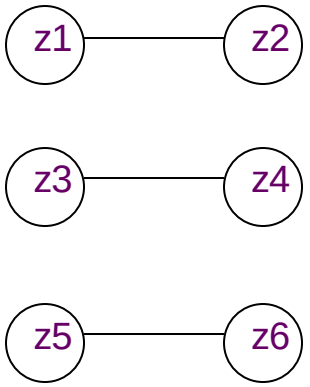
Example



(a)



(b)



(c)

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

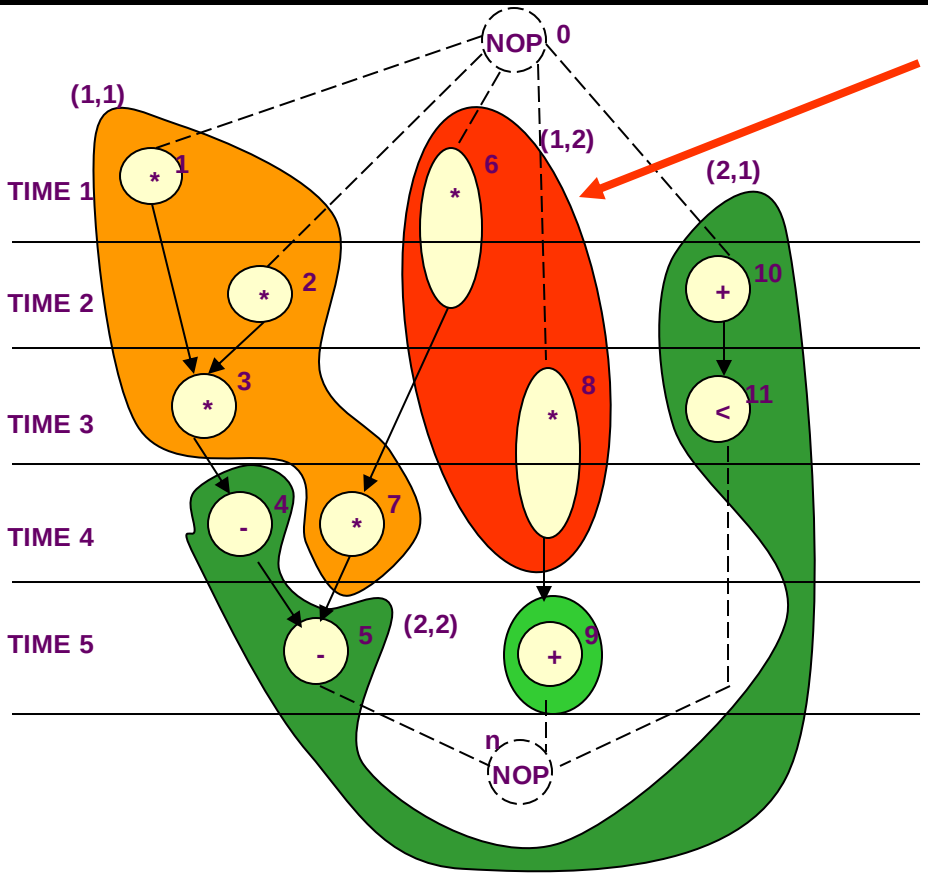
Module selection problem

- **Extension of resource sharing**
  - **Library of resources:**
  - **More than one resource per type**
- **Example:**
  - **Ripple-carry adder (small and slow)**
  - **Carry look-ahead adder (big and fast)**
- **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**

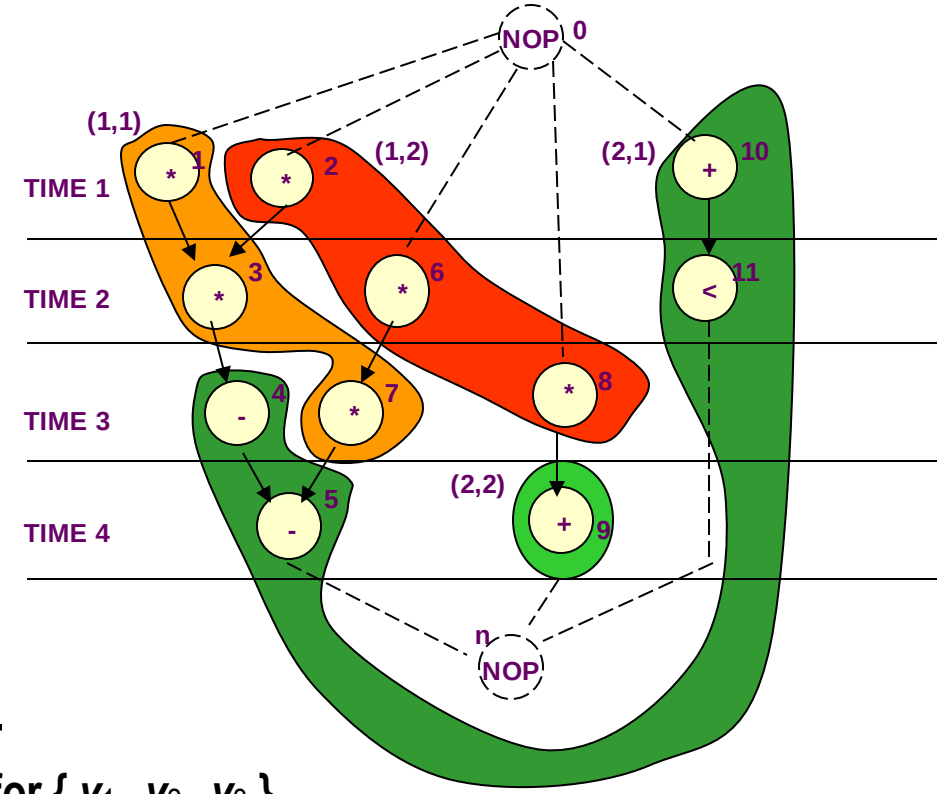
Example



Slow multipliers save area

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

Example 2



- **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**

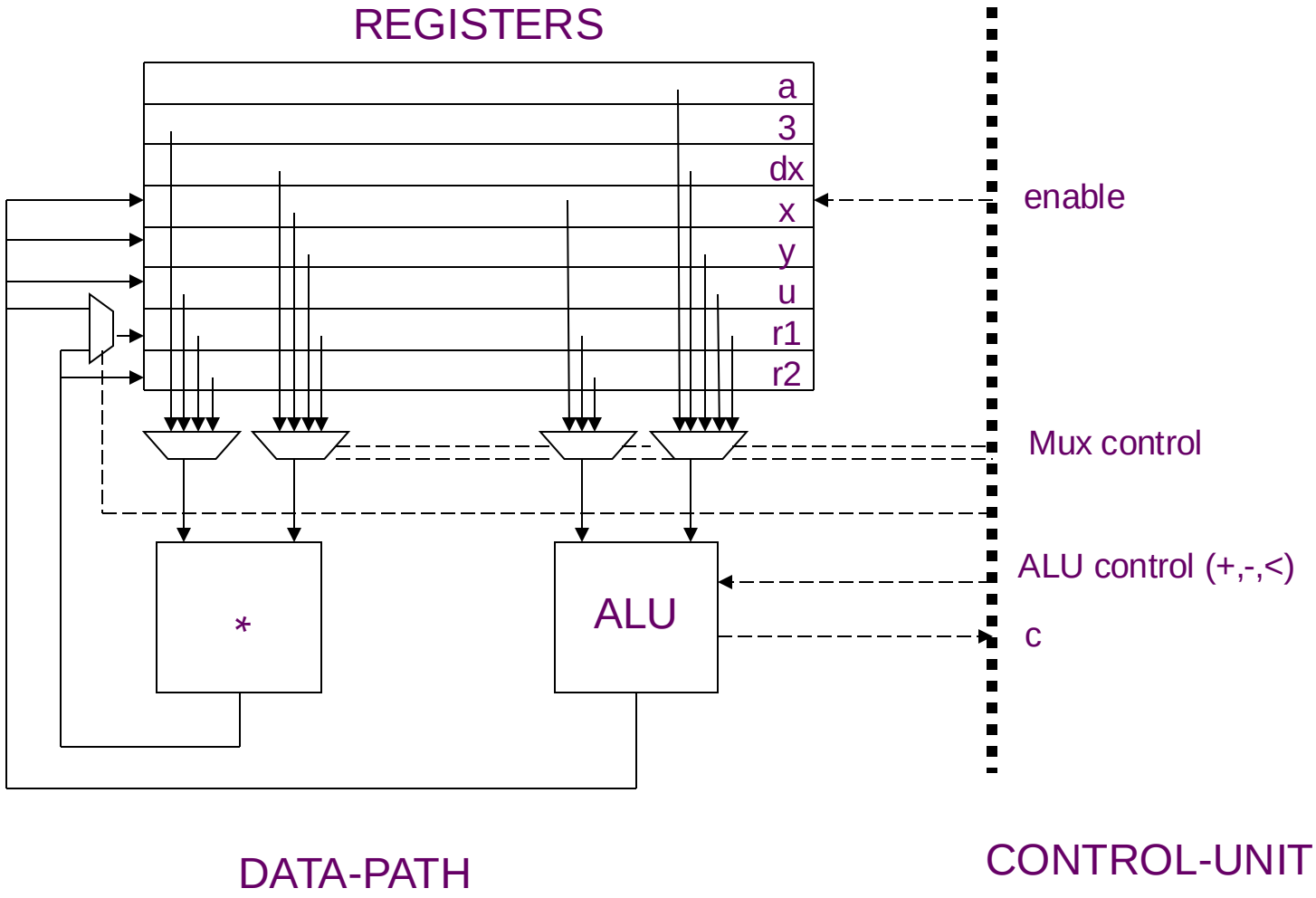
Module 2

- **Objectives**
  - **Data path generation**
  - **Control synthesis**

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

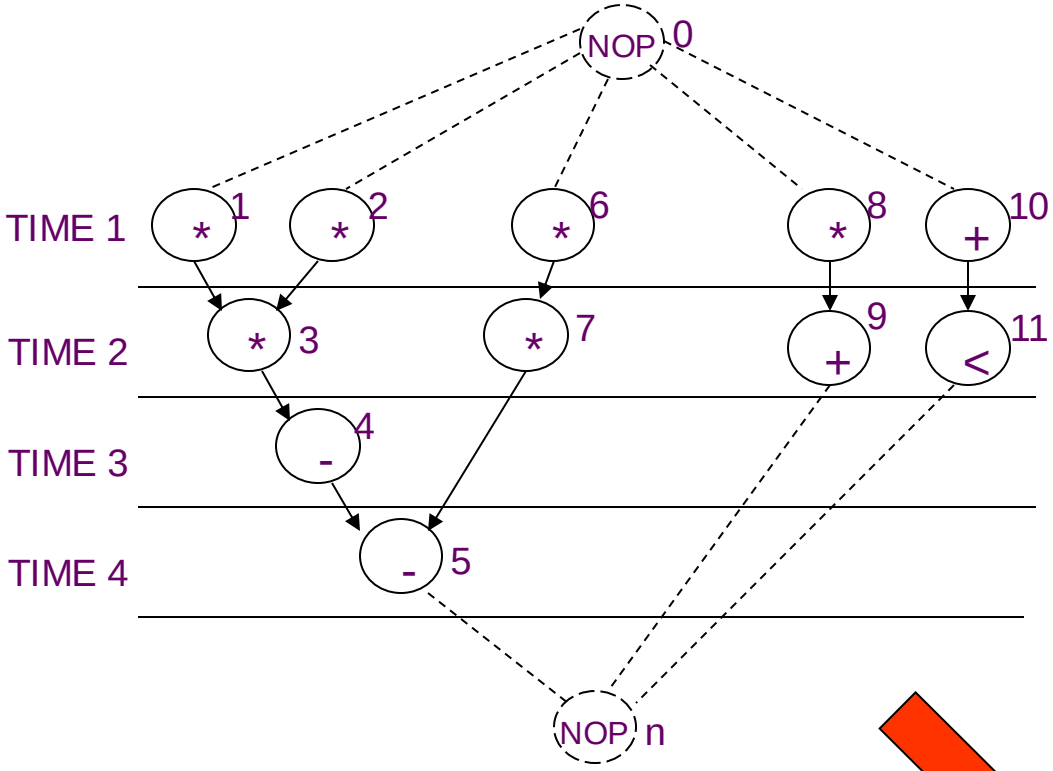
Example

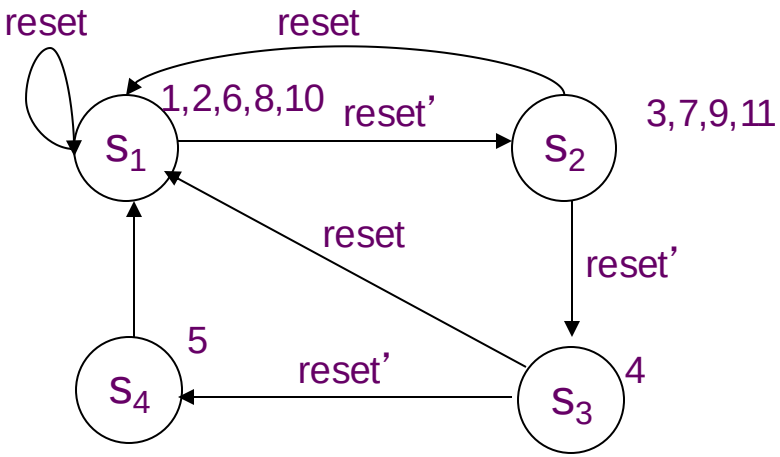

REGISTERS
a
3
dx
x
y
u
r1
r2
enable
Mux control
ALU control (+,-,<)
c
*
ALU
DATA-PATH
CONTROL-UNIT

Control synthesis

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

Example


NOP
0
TIME 1
*
1
*
2
*
6
*
8
+
10
TIME 2
*
3
*
7
+
9
<
11
TIME 3
-
4
TIME 4
-
5
NOP
n


reset
reset
1,2,6,8,10
reset'
3,7,9,11
S1
S2
reset
reset'
5
S4
reset'
4
S3

Summary

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