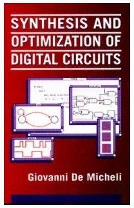
Resource sharing

Giovanni De Micheli
Integrated Systems Centre
EPF Lausanne



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

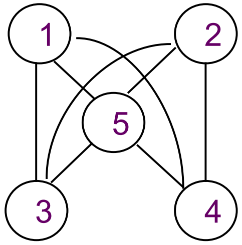
- **Scheduled sequencing graphs**
  - **Operation concurrency well defined**
- **Consider *operation types* independently**
  - **Problem decomposition**
  - **Perform analysis for each resource type**

Compatibly and conflicts

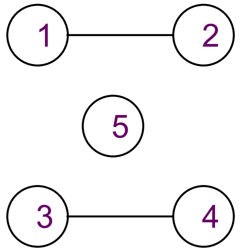
- **Operation compatibility:**
  - Same type
  - Non concurrent
- ***Compatibility* graph:**
  - Vertices: operations
  - Edges: compatibility relation
- ***Conflict* graph:**
  - Complement of compatibility graph

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



Conflict graph



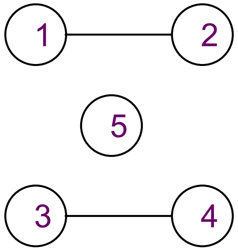
Compatibility and conflicts

- **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

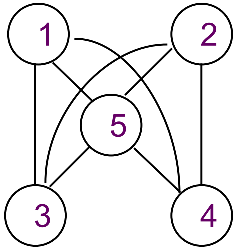
Example

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

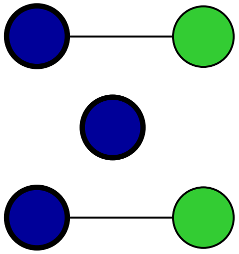
Conflict



Compatibility

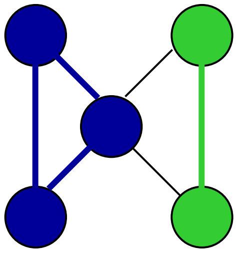


Coloring



ALU1: 1,3,5
ALU2: 2,4

Partitioning



Perfect graphs

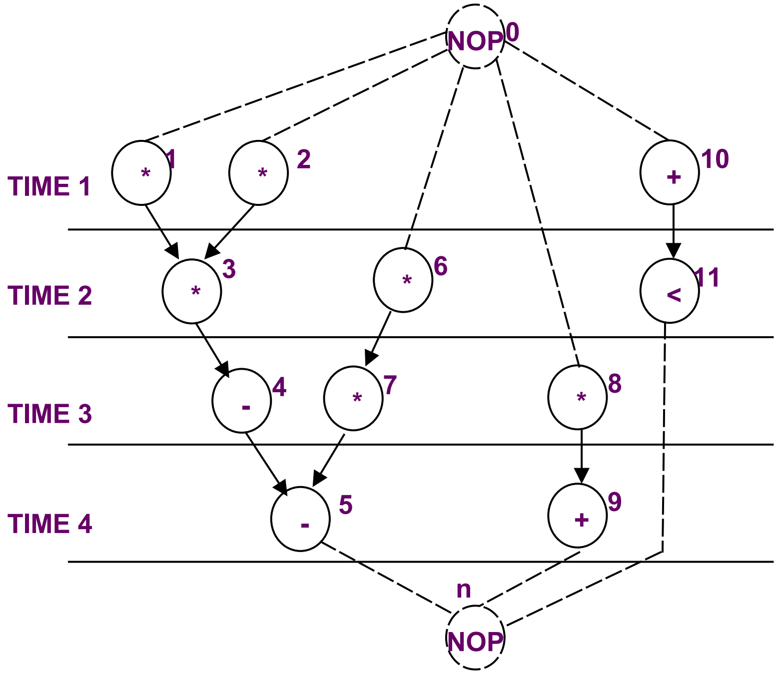
- ***Comparability graph***:
  - Graph G (V, E) has an orientation G (V, F) with the transitive property

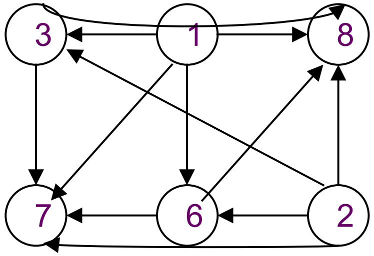
    $(v_i, v_j) \in F$ and $(v_j, v_k) \in F \rightarrow (v_i, v_k) \in F$
- ***Interval graph***:
  - Vertices correspond to *intervals*
  - Edges correspond to interval intersection
  - Subset of *chordal* graphs
    - Every loop with more than three edged has a chord

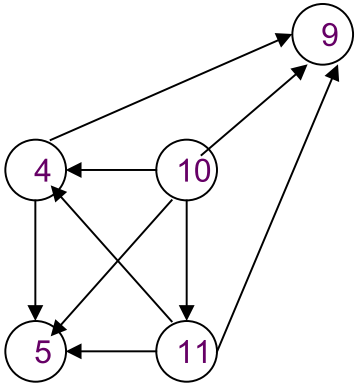
Data-flow graphs
(flat sequencing graphs)

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

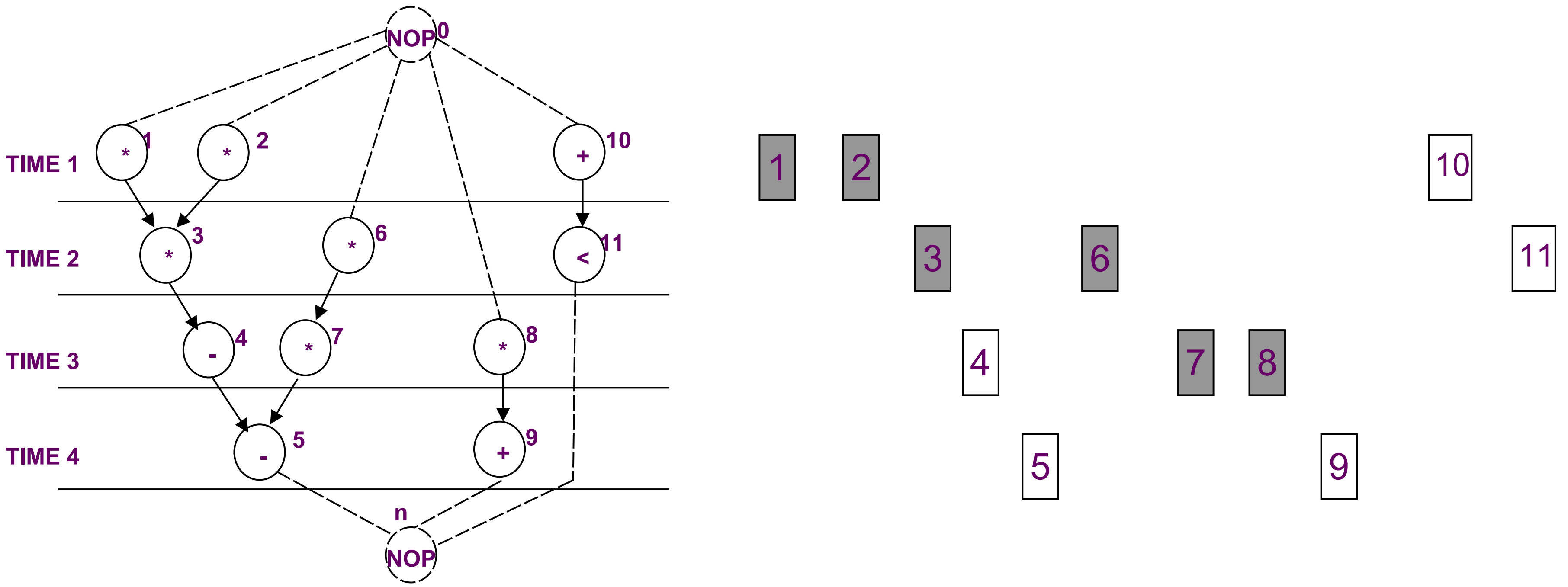
Example


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


3
1
8
7
6
2


9
4
10
5
11

Example



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

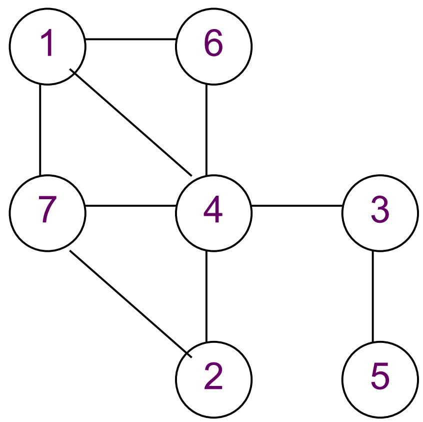
Left-edge algorithm

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

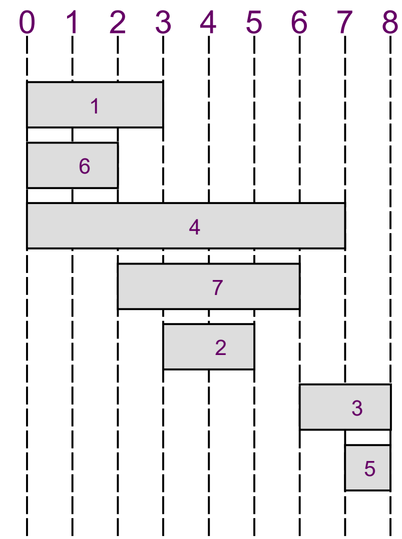
Left-edge algorithm

```
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;
    }
}
```

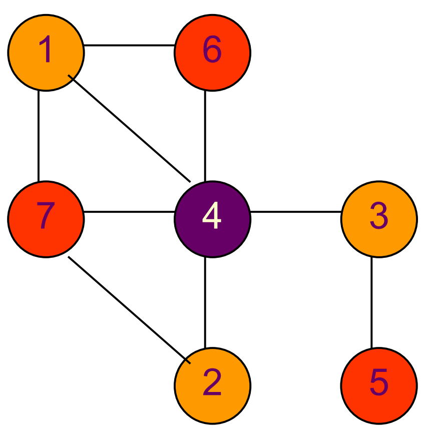
Example



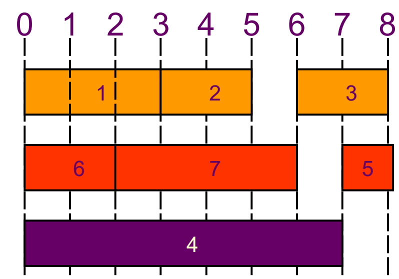
Conflict graph



Intervals



Colored conflict graph

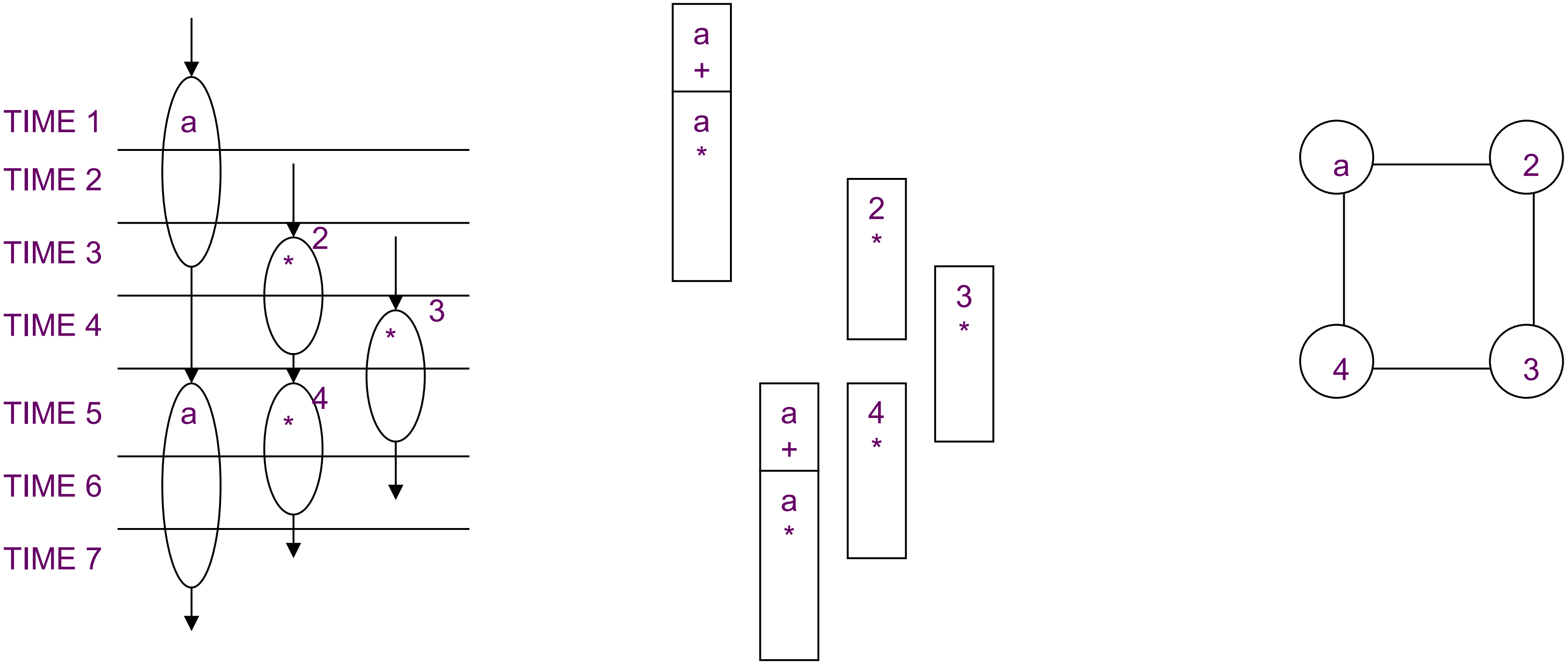


Coloring

Hierarchical sequencing graphs

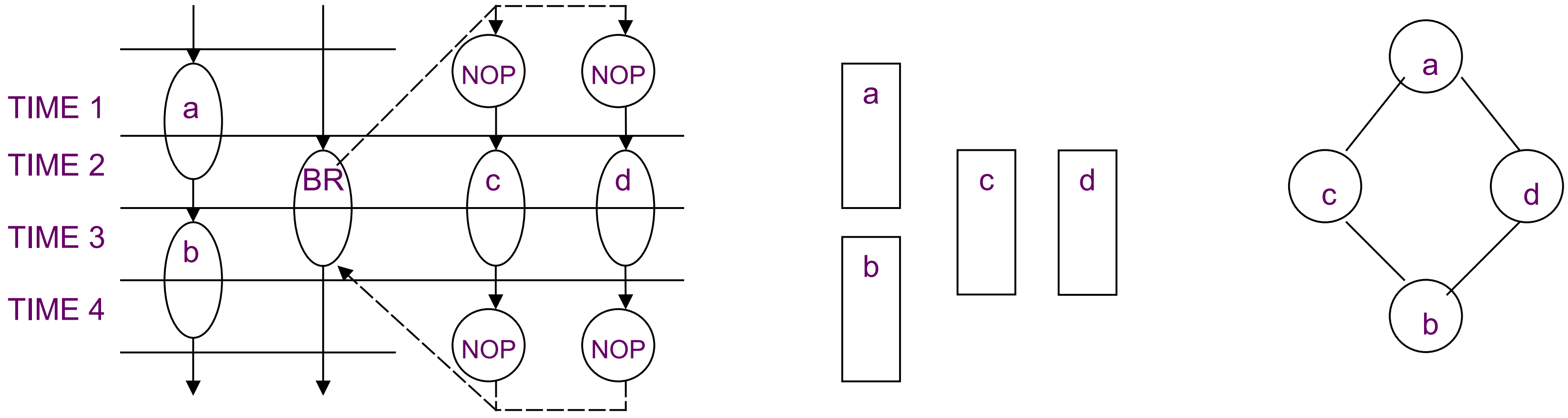
- Hierarchical conflict/compatibility graphs:
  - Easy to compute
  - Prevent sharing across hierarchy
- Flatten hierarchy:
  - Bigger graphs
  - Destroy nice properties

Example



(a) (b) (c)

Example



(a) (b) (c)

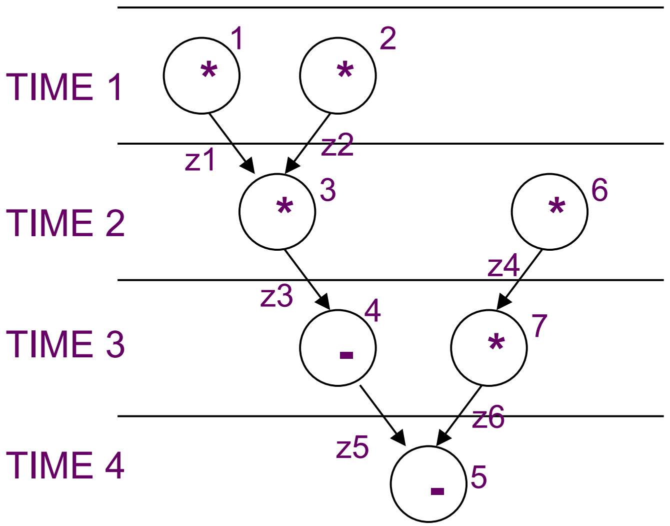
Register binding problem

- **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**

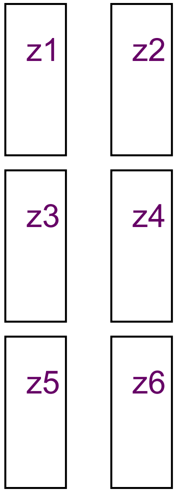
Register sharing in data-flow graphs

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

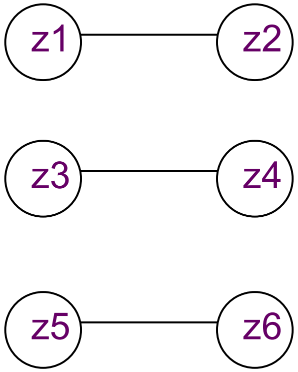
Example


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)

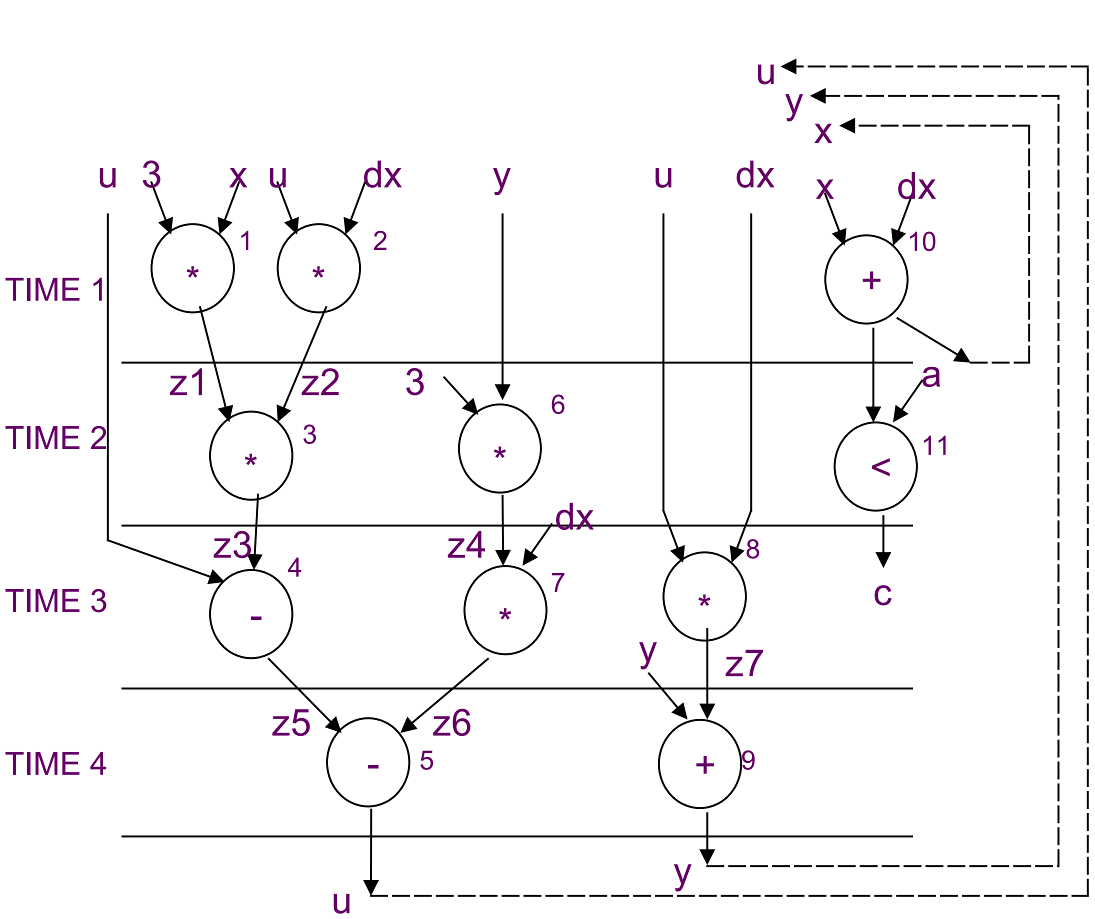

z1
z2
z3
z4
z5
z6

(c)

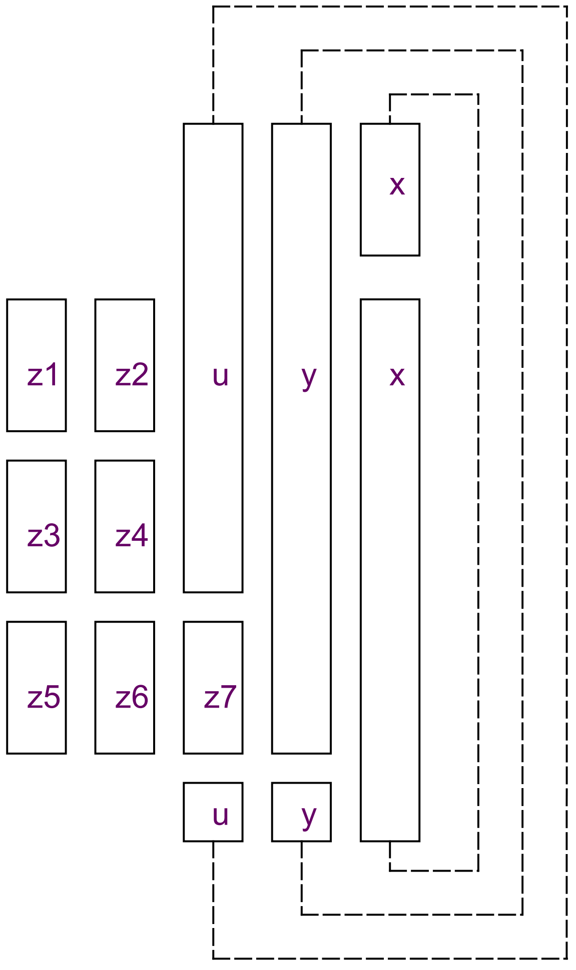
Register sharing general case

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

Example

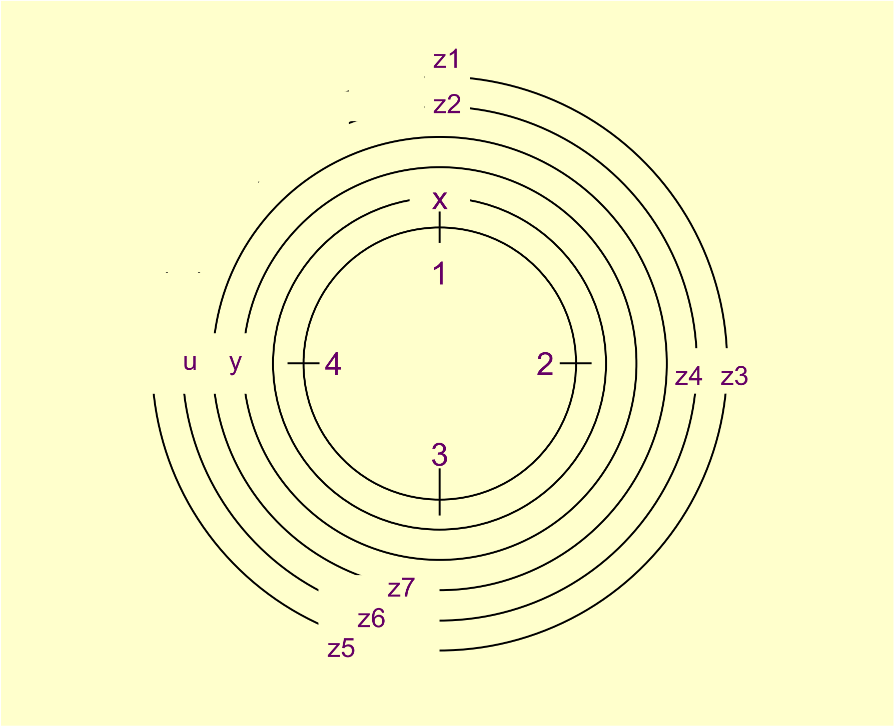

u 3 x u dx y u dx x dx
TIME 1
TIME 2
TIME 3
TIME 4
z1 z2 3 a
z3 z4 dx
y z7
z5 z6
u y x
c
1 2 3 4 5 6 7 8 9 10 11

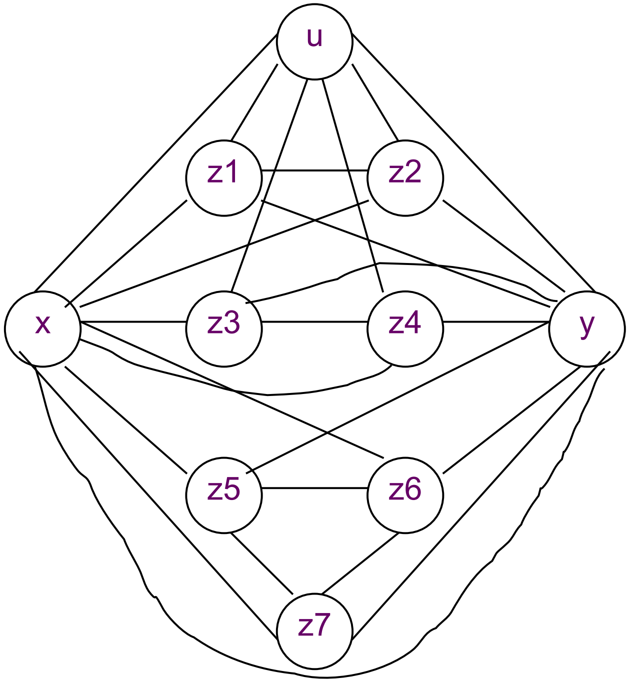
(a)


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

(b)

Example
Variable-lifetimes and circular-arc conflict graph





Multiport-memory binding

- Find *minimum number* of ports to access the required number of variables
- Variables use the same port:
  - Port compatibility/conflict
  - Similar to resource binding
- Variables can use any port:
  - Decision variable x_{il} id TRUE when variable *i* is accessed is step *l*
  - Optimum: $\max \sum_{j=1..nvar} x_{il} \quad s.t.\ 1 \leq l \leq \lambda + 1$

Multiport-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 $i=1$ will be stored in array

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

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

Example

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

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

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

$Time - step\ 4 : r_{14} = r_{11}\ \&\ r_8\ ;\ r_{15} = r_{12}\ |\ r_9$

$Time - step\ 5 : r_1 = r_{11}\ ;\ r_2 = r_{15}$

$$\max \sum_{i=1}^{15} b_i \text{ such that}$$

$$b_1 + b_2 + b_3 + b_{12} \leq a$$

$$b_3 + b_4 + b_5 + b_6 + b_7 + b_{13} \leq a$$

$$b_1 + b_3 + b_5 + b_7 + b_8 + b_9 + b_{10} + b_{11} \leq a$$

$$b_8 + b_9 + b_{11} + b_{12} + b_{14} + b_{15} \leq a$$

$$b_1 + b_2 + b_{14} + b_{15} \leq a$$

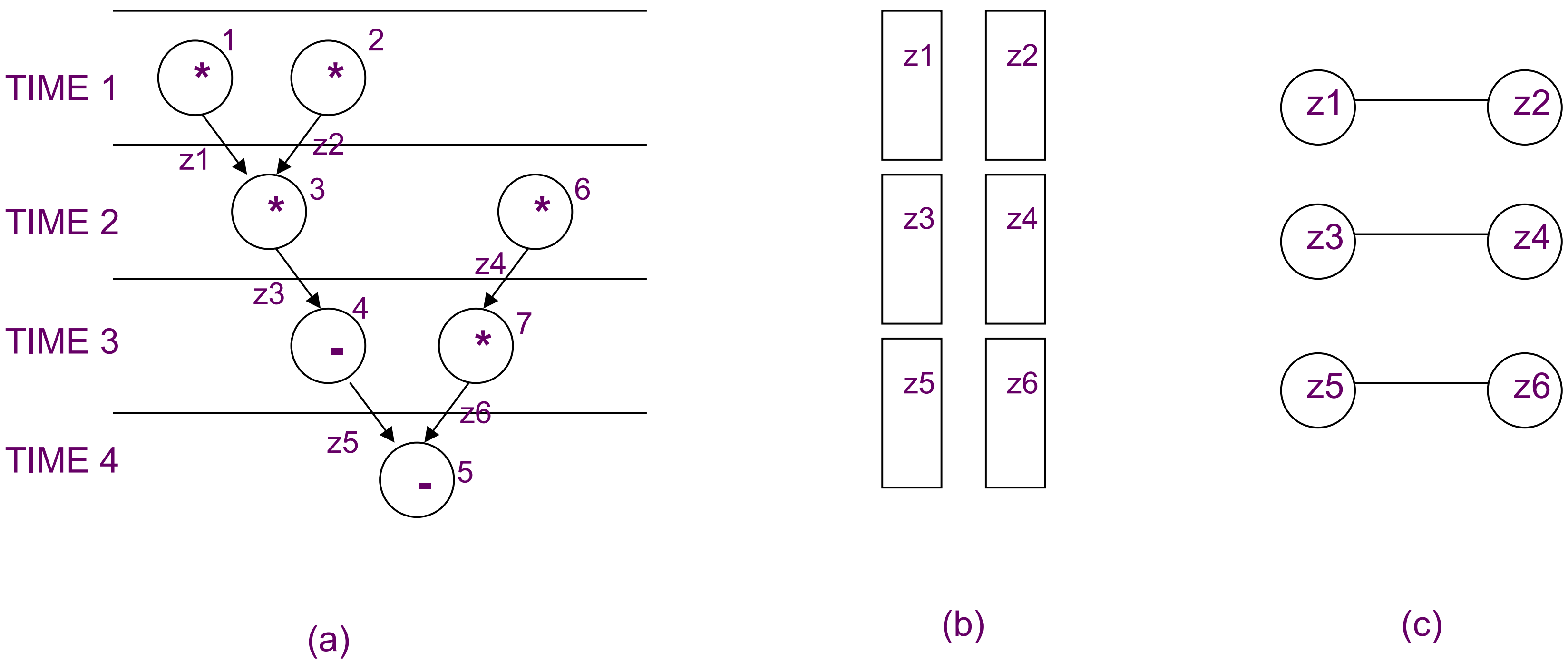
Example

- 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: $v_1, v_2, v_4, v_6, v_8, v_{10}, v_{12}, v_{13}$

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
- ILP formulation or heuristic algorithms

Example



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

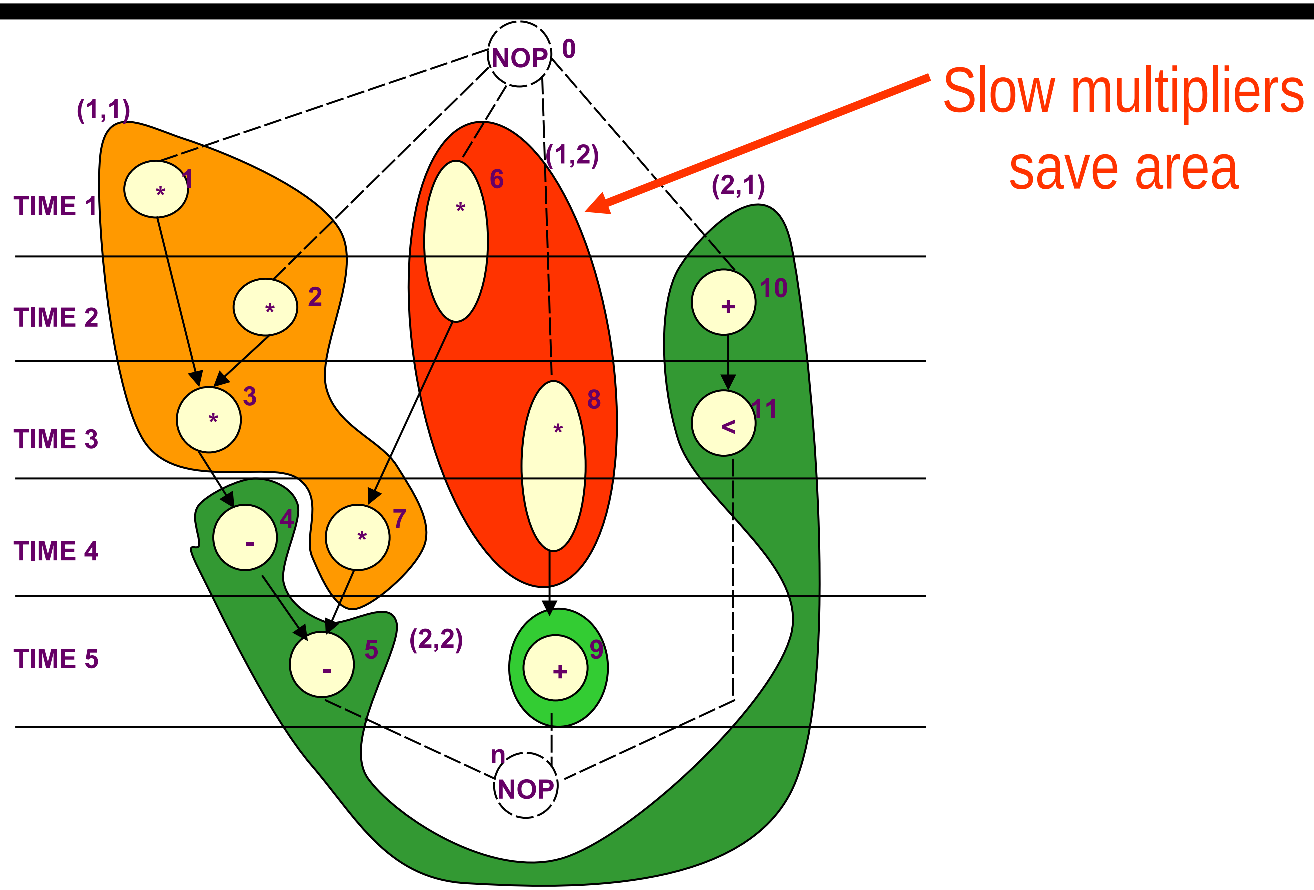
Module selection problem

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

Module selection solution

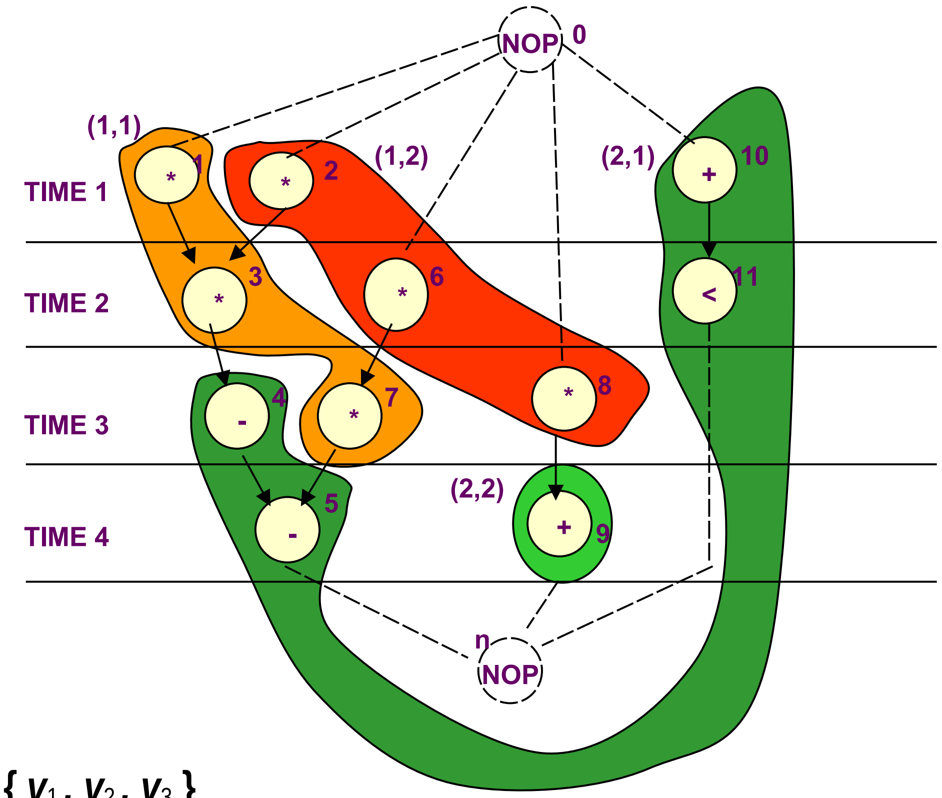
- **ILP formulation:**
  - **Decision variables**
    - **Select resource sub-type**
    - **Determine (*area, delay*)**
- **Heuristic algorithm**
  - **Determine *minimum latency* with fastest resource subtypes**
  - **Recover area by using slower resources on non-critical paths**

Example



- **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

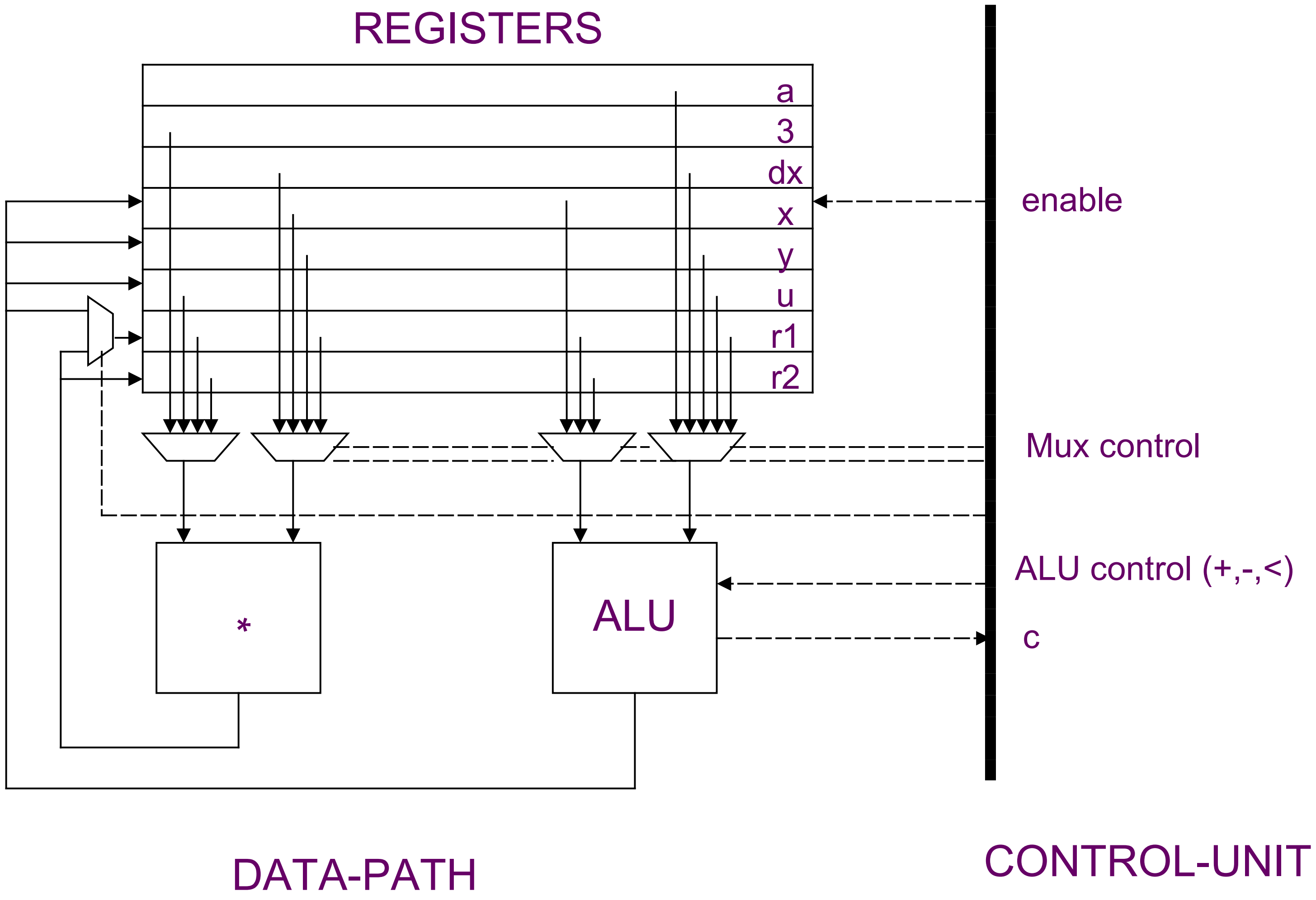
Module 2

- 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

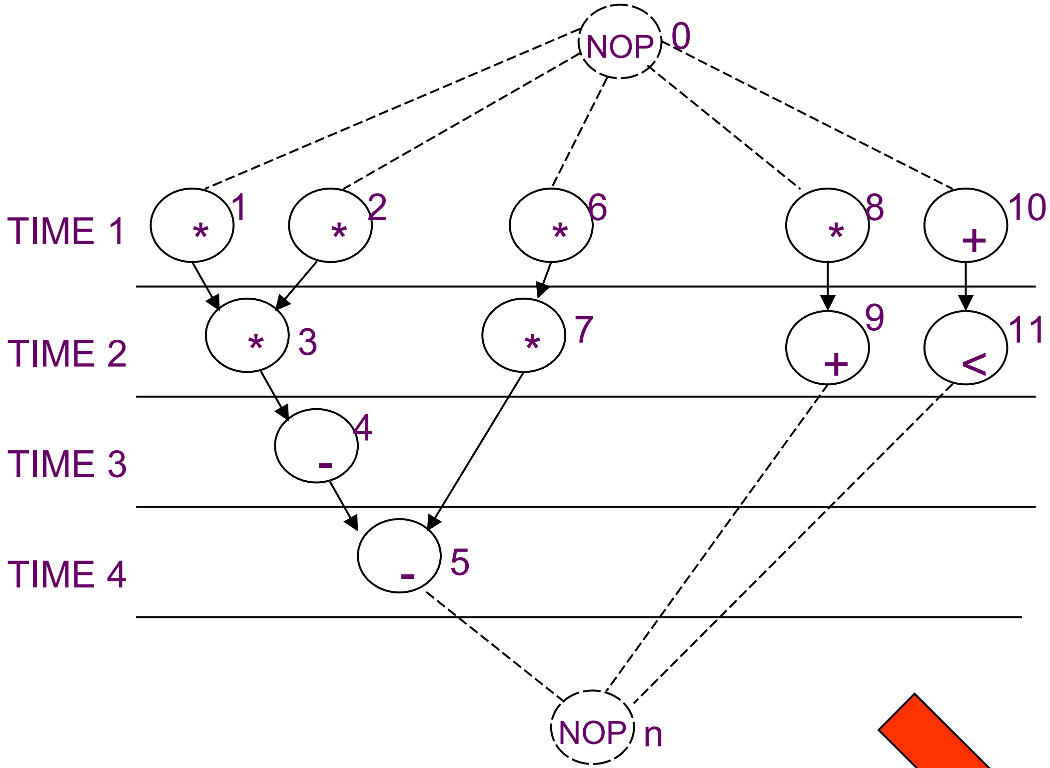
Example


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

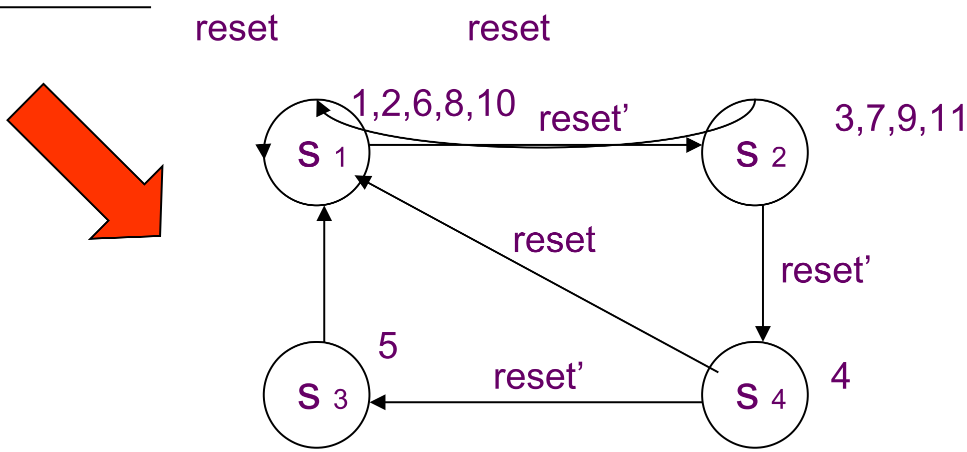
Control synthesis

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

Example



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



reset
reset
1,2,6,8,10
reset’
3,7,9,11
S 1
S 2
reset
reset’
5
4
reset’
S 3
S 4

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**
  - **Generated data path is an interconnection of blocks**
  - **Control is one or more finite-state machines**