COE 561
Digital System Design & Synthesis
Architectural Synthesis

Dr. Aiman H. El-Maleh

Computer Engineering Department

King Fahd University of Petroleum & Minerals

[Adapted from slides of Prof. G. De Micheli: Synthesis & Optimization of Digital Circuits]

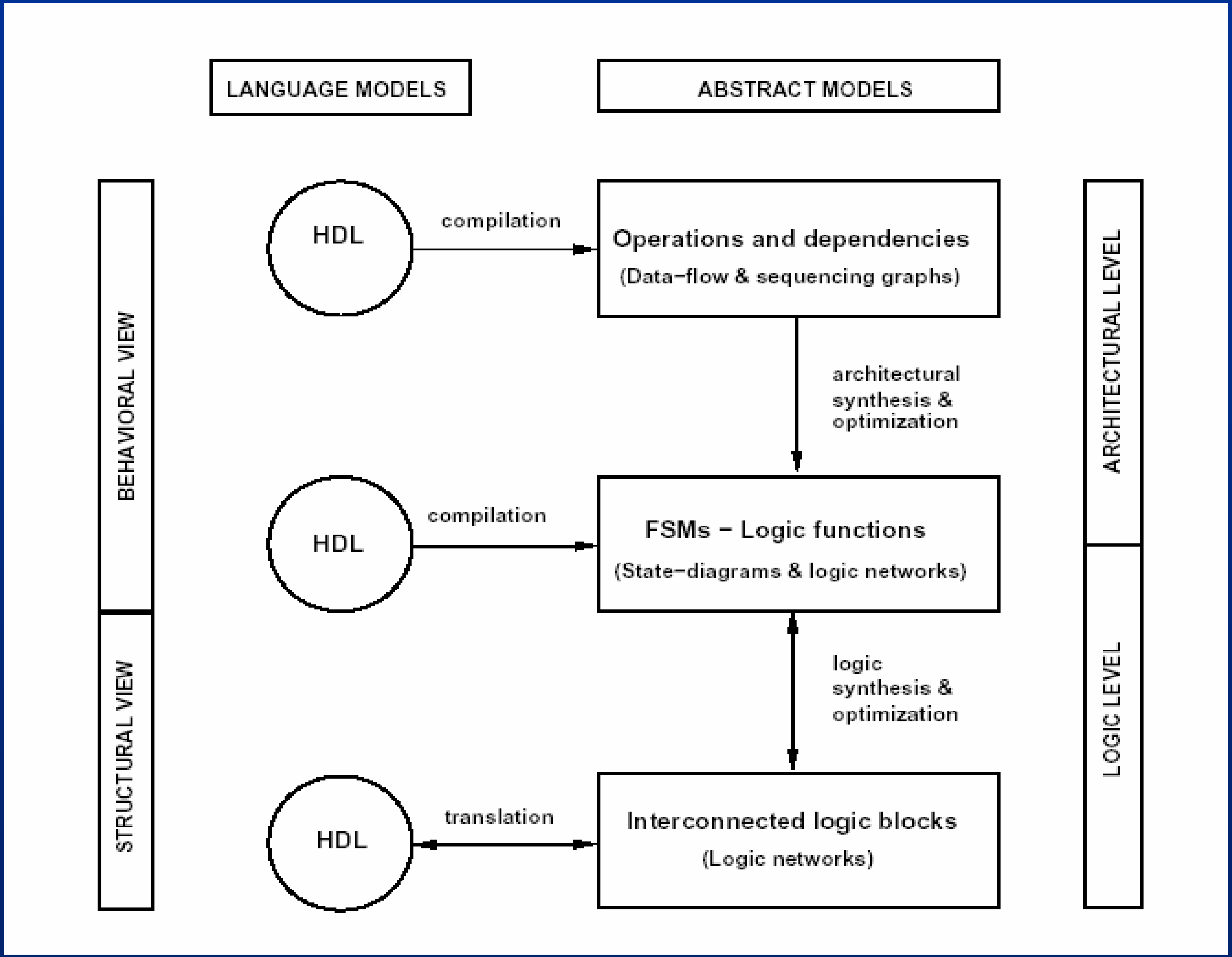
Outline

- **Motivation**
- **Dataflow graphs & Sequencing graphs**
- **Resources**
- **Synthesis in temporal domain: Scheduling**
- **Synthesis in spatial domain: Binding**
- **Scheduling Models**
  - Unconstrained scheduling
  - Scheduling with timing constraints
  - Scheduling with resource constraints
- **Algorithmic Solution to the Optimum Binding Problem**
- **Register Binding Problem**

Synthesis

- **Transform behavioral into structural view.**
- **Architectural-level synthesis**
  - Architectural abstraction level.
  - Determine macroscopic structure.
  - Example: major building blocks like adder, register, mux.
- **Logic-level synthesis**
  - Logic abstraction level.
  - Determine microscopic structure.
  - Example: logic gate interconnection.

Synthesis and Optimization


LANGUAGE MODELS
ABSTRACT MODELS
BEHAVIORAL VIEW
STRUCTURAL VIEW
ARCHITECTURAL LEVEL
LOGIC LEVEL
HDL
compilation
Operations and dependencies
(Data-flow & sequencing graphs)
architectural synthesis & optimization
HDL
compilation
FSMs – Logic functions
(State-diagrams & logic networks)
logic synthesis & optimization
HDL
translation
Interconnected logic blocks
(Logic networks)

Architectural-Level Synthesis Motivation

- **Raise input abstraction level.**
  - Reduce specification of details.
  - Extend designer base.
  - Self-documenting design specifications.
  - Ease modifications and extensions.
- **Reduce design time.**
- **Explore and optimize macroscopic structure**
  - Series/parallel execution of operations.

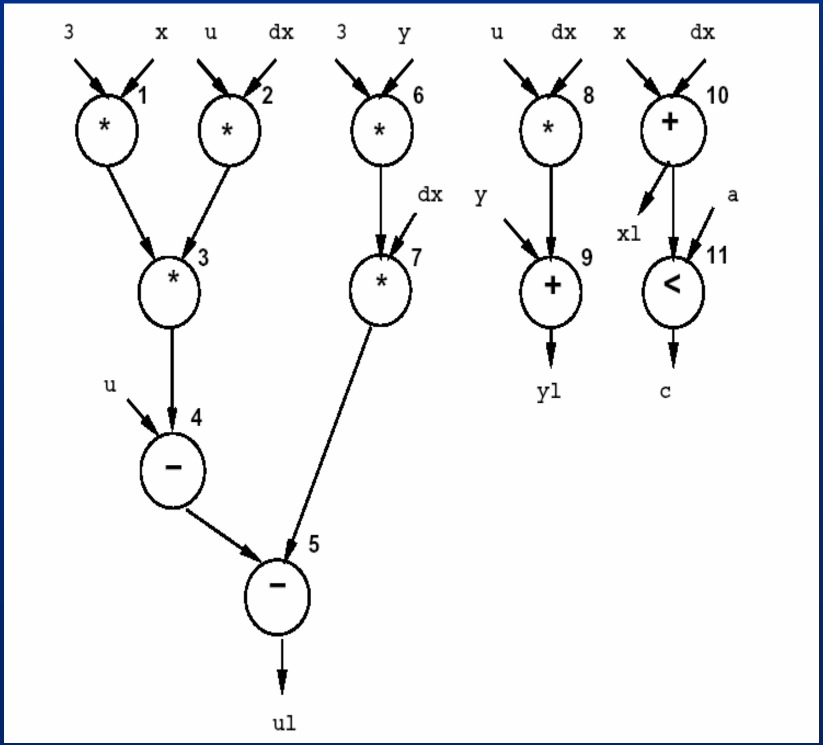
Architectural-Level Synthesis

- **Translate HDL models into sequencing graphs.**
- **Behavioral-level optimization**
  - Optimize abstract models independently from the implementation parameters.
- **Architectural synthesis and optimization**
  - Create macroscopic structure
    - data-path and control-unit.
  - Consider area and delay information of the implementation.

Dataflow Graphs …

- Behavioral views of architectural models.
- Useful to represent data-paths.
- Graph
  - Vertices = operations.
  - Edges = dependencies.
- Dependencies arise due
  - Input to an operation is result of another operation.
  - Serialization constraints in specification.
  - Two tasks share the same resource.

$$\begin{aligned} xl &= x + dx \\ ul &= u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx) \\ yl &= y + u \cdot dx \\ c &= xl < a \end{aligned}$$

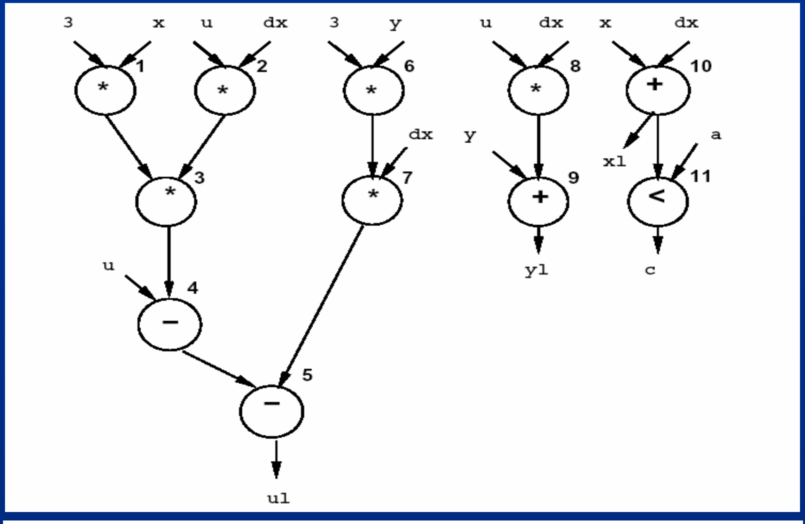


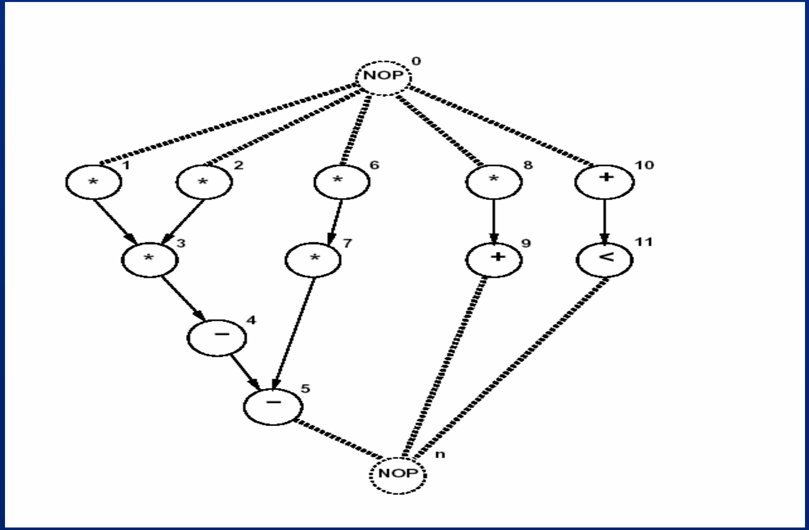
… Dataflow Graphs

- Assumes the existence of variables who store information required and generated by operations.
- Each variable has a *lifetime* which is the interval from *birth* to *death.*
- *Variable birth* is the time at which the value is generated.
- *Variable death* is the latest time at which the value is referenced as input to an operation.
- Values must be preserved during life-time.

Sequencing Graphs

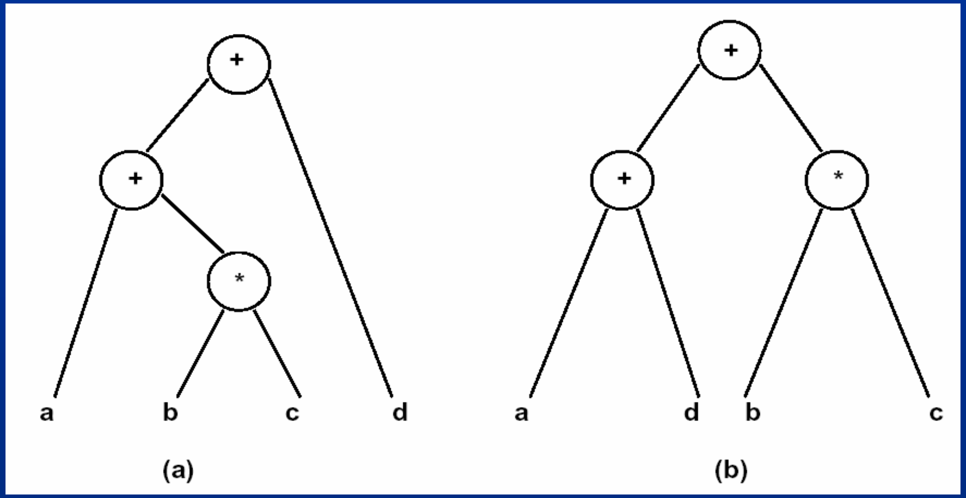
- **Useful to represent data-path and control.**
- **Extended dataflow graphs**
  - Control Data Flow Graphs (CDFGs).
  - Polar: source and sink.
  - Operation serialization.
  - Hierarchy.
  - Control-flow commands
    - branching and iteration.
- **Paths in the graph represent concurrent streams of operations.**



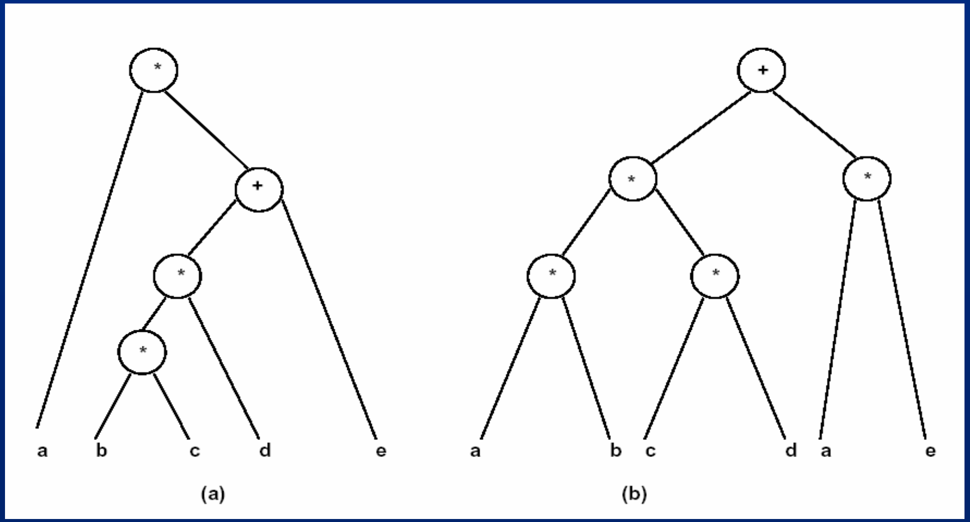


Behavioral-level optimization

- Tree-height reduction using commutativity and associativity
- x = a + b * c + d =>
  x = (a + d) + b * c



- Tree-height reduction using distributivity
- x = a * (b * c * d + e) =>
  x = a * b * c * d + a * e



Architectural Synthesis and Optimization

- **Synthesize macroscopic structure in terms of building-blocks.**
- **Explore area/performance trade-offs**
  - maximum performance implementations subject to area constraints.
  - minimum area implementations subject to performance constraints.
- **Determine an optimal implementation.**
- **Create logic model for data-path and control.**

Circuit Specification for Architectural Synthesis

- **Circuit behavior**
  - Sequencing graphs.
- **Building blocks**
  - Resources.
    - Functional resources: process data (e.g. ALU).
    - Memory resources: store data (e.g. Register).
    - Interface resources: support data transfer (e.g. MUX and Buses).
- **Constraints**
  - Interface constraints
    - Format and timing of I/O data transfers.
  - Implementation constraints
    - Timing and resource usage.
      - Area
      - Cycle-time and latency

Resources

- **Functional resources: perform operations on data.**
  - Example: arithmetic and logic blocks.
  - Standard resources
    - Existing macro-cells.
    - Well characterized (area/delay).
    - Example: adders, multipliers, ALUs, Shifters, ...
  - Application-specific resources
    - Circuits for specific tasks.
    - Yet to be synthesized.
    - Example: instruction decoder.
- **Memory resources: store data.**
  - Example: memory and registers.
- **Interface resources**
  - Example: busses and ports.

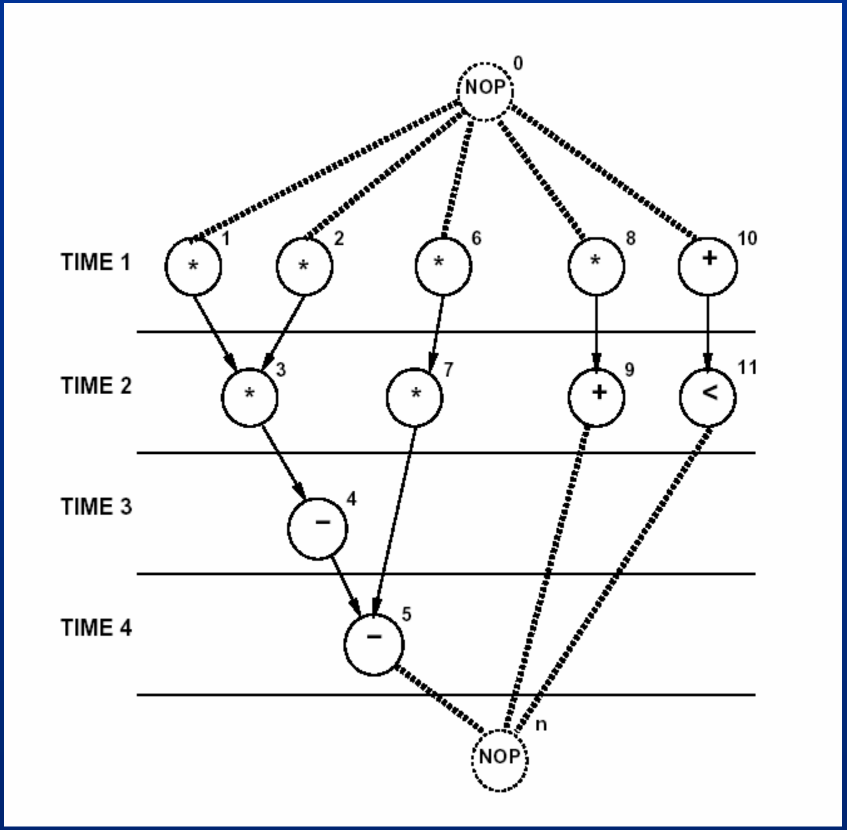
Resources and Circuit Families

- **Resource-dominated circuits.**
  - Area and performance depend on few, well-characterized blocks.
  - Example: DSP circuits.
- **Non resource-dominated circuits.**
  - Area and performance are strongly influenced by sparse logic, control and wiring.
  - Example: some ASIC circuits.

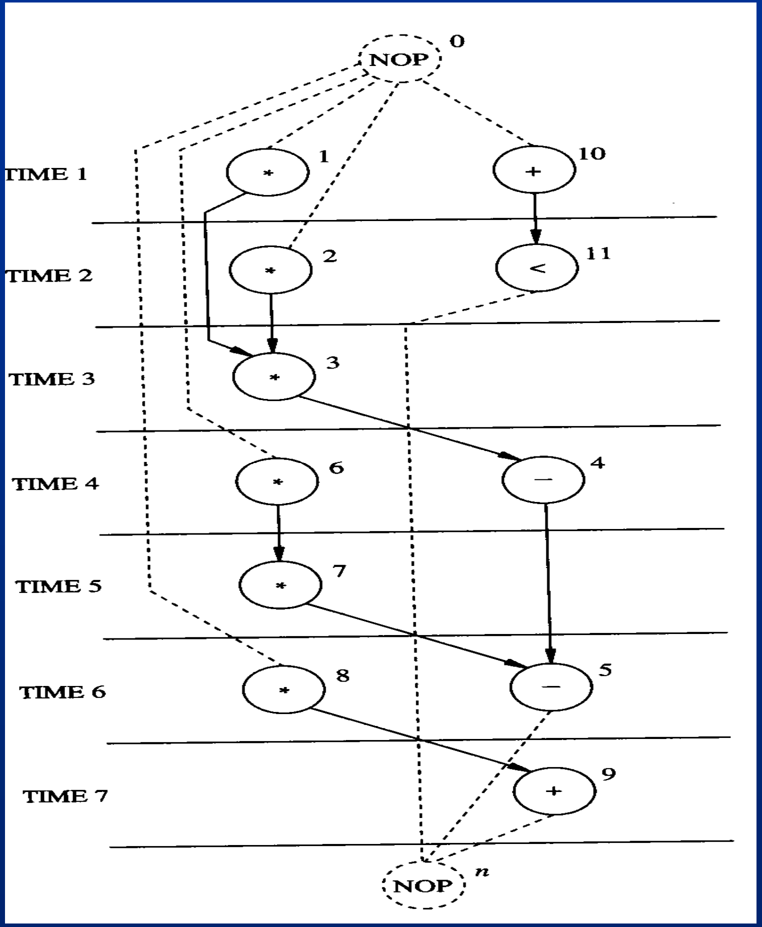
Synthesis in the Temporal Domain: Scheduling

- **Scheduling**
  - Associate a start-time with each operation.
  - Satisfying all the sequencing (timing and resource) constraint.
- **Goal**
  - Determine area/latency trade-off.
  - Determine latency and parallelism of the implementation.
- **Scheduled sequencing graph**
  - Sequencing graph with start-time annotation.
- **Unconstrained scheduling.**
- **Scheduling with timing constraints**
- **Scheduling with resource constraints.**

Scheduling …

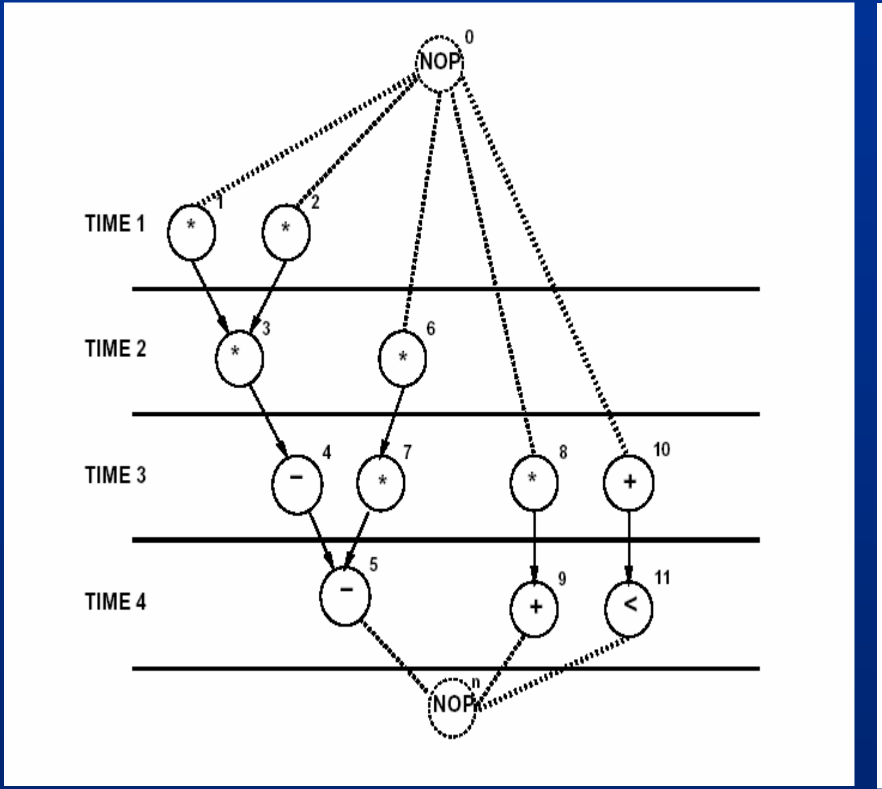


4 Multipliers, 2 ALUs

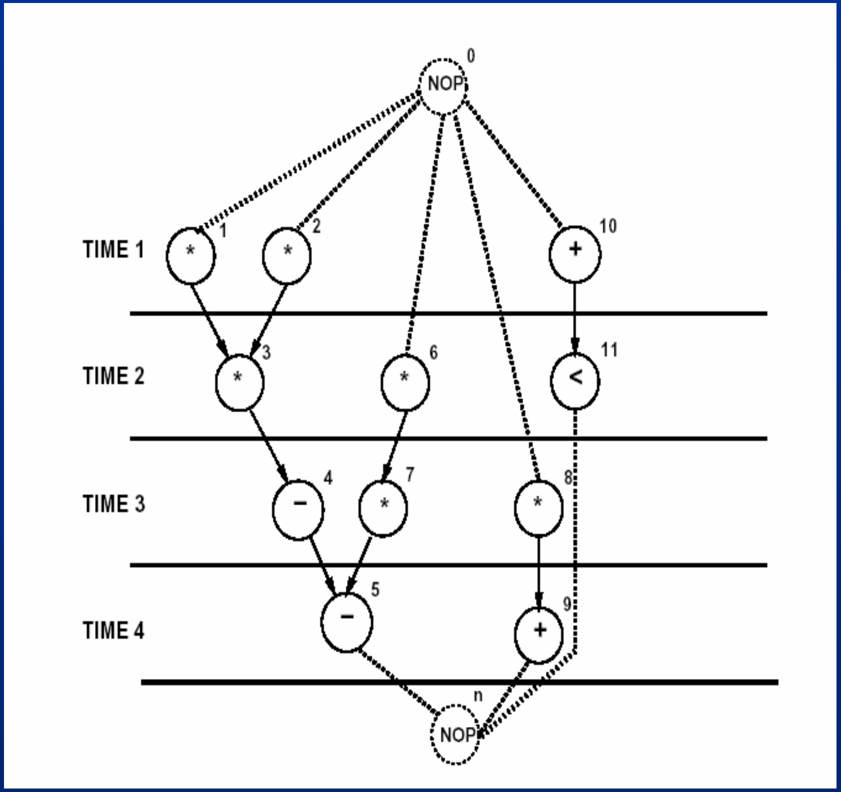


1 Multiplier , 1 ALU

… Scheduling



2 Multipliers, 3 ALUs

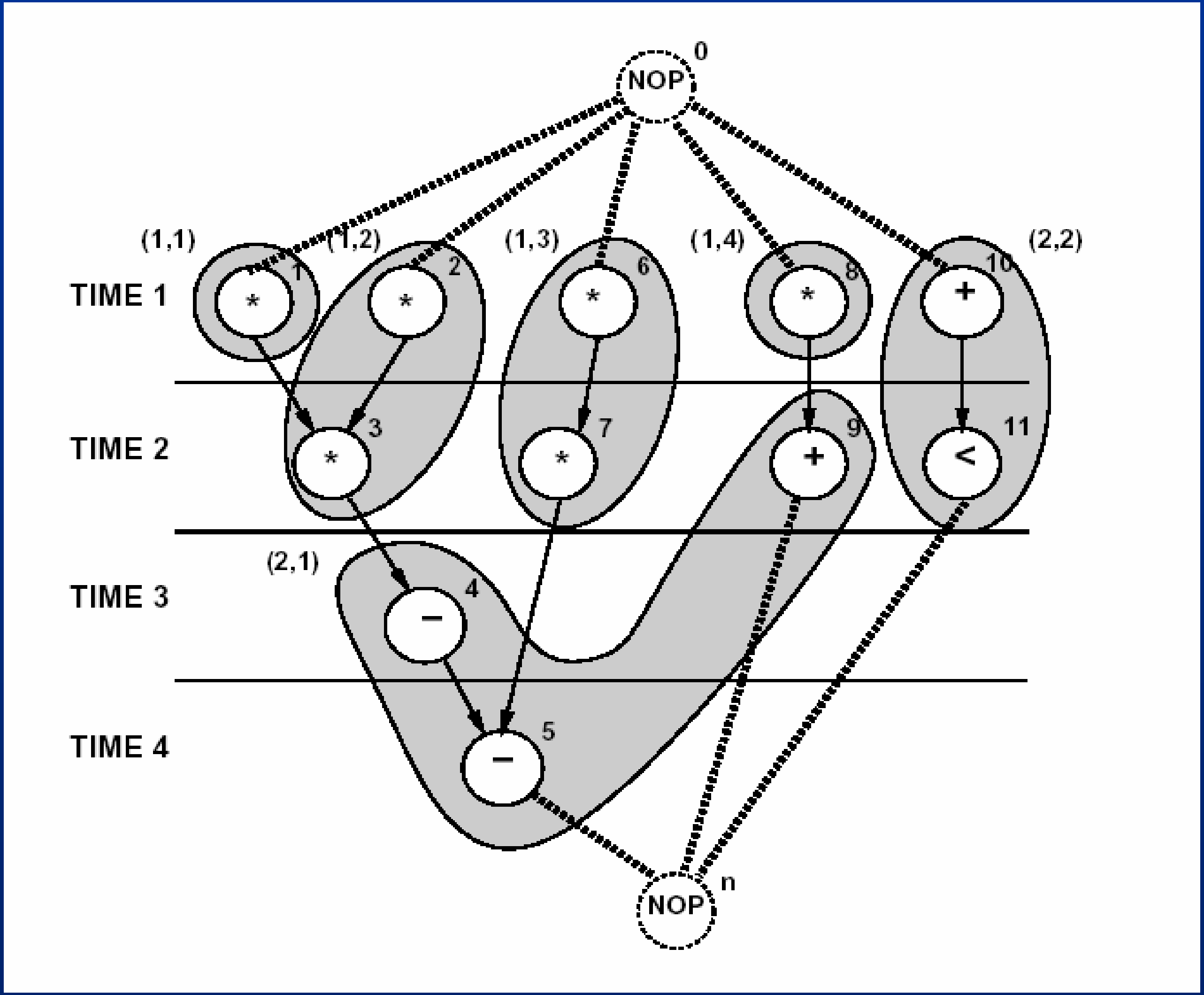


2 Multipliers, 2 ALUs

Synthesis in the Spatial Domain: Binding

- **Binding**
  - Associate a resource with each operation with the same type.
  - Determine area of the implementation.
- **Sharing**
  - Bind a resource to more than one operation.
  - Operations must not execute concurrently.
- **Bound sequencing graph**
  - Sequencing graph with resource annotation.

Example: Bound Sequencing Graph



Performance and Area Estimation

- **Resource-dominated circuits**
  - Area = sum of the area of the resources bound to the operations.
    - Determined by binding.
  - Latency = start time of the sink operation (minus start time of the source operation).
    - Determined by scheduling
- **Non resource-dominated circuits**
  - Area also affected by
    - registers, steering logic, wiring and control.
  - Cycle-time also affected by
    - steering logic, wiring and (possibly) control.

Scheduling

- **Circuit model**
  - Sequencing graph.
  - Cycle-time is given.
  - Operation delays expressed in cycles.
- **Scheduling**
  - Determine the start times for the operations.
  - Satisfying all the sequencing (timing and resource) constraint.
- **Goal**
  - Determine area/latency trade-off.
- **Scheduling affects**
  - Area: maximum number of concurrent operations of same type is a lower bound on required hardware resources.
  - Performance: concurrency of resulting implementation.

Scheduling Models

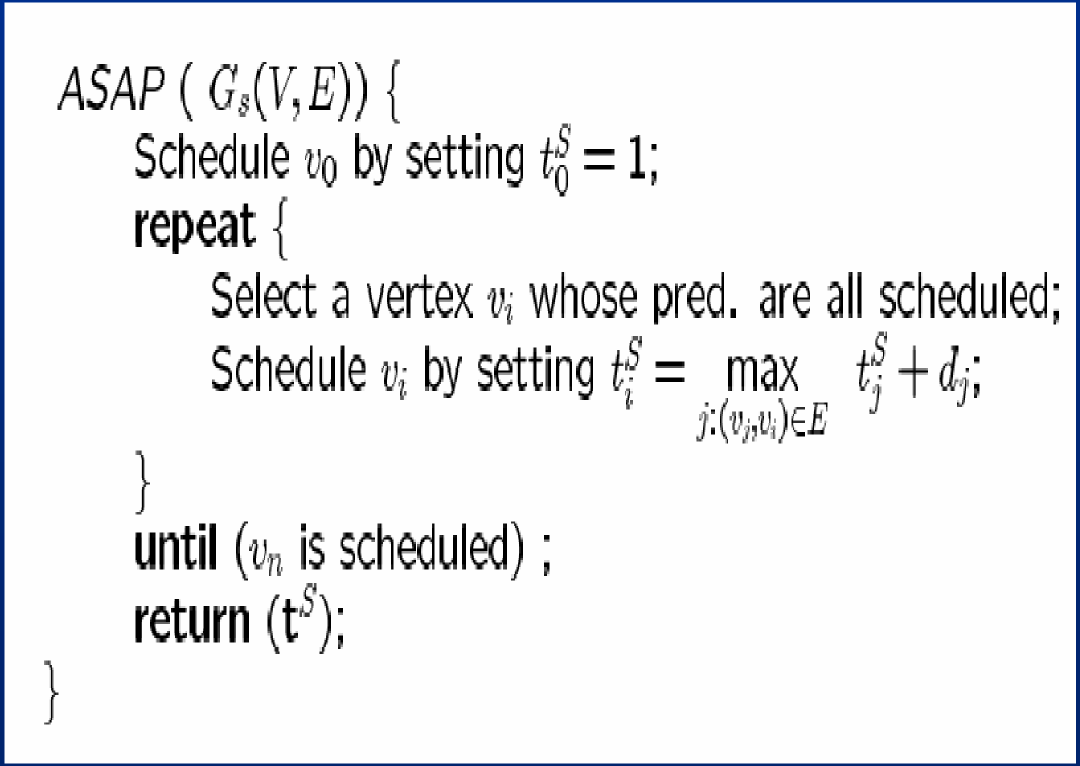
- **Unconstrained scheduling.**
- **Scheduling with timing constraints**
  - Latency.
  - Detailed timing constraints.
- **Scheduling with resource constraints.**
- **Simplest scheduling model**
  - All operations have bounded delays.
  - All delays are in cycles.
    - Cycle-time is given.
  - No constraints - no bounds on area.
  - Goal
    - Minimize latency.

Minimum-Latency Unconstrained Scheduling Problem

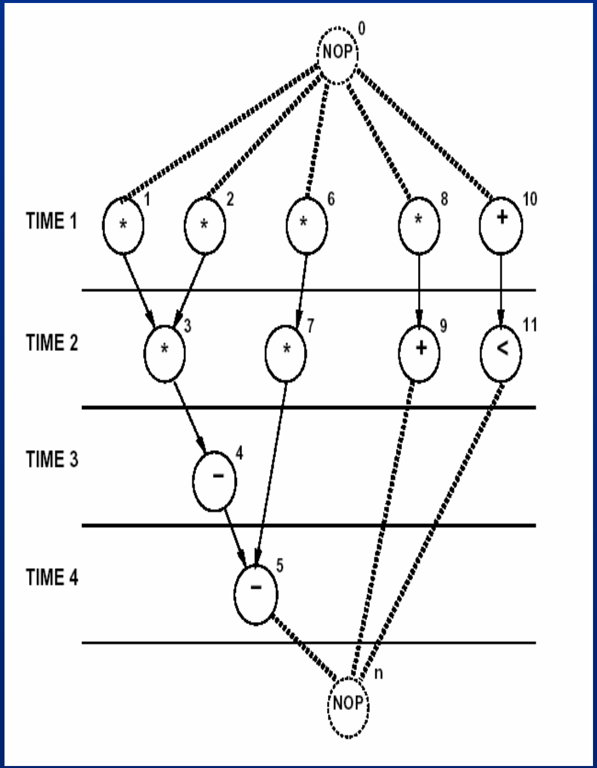
- **Given a set of operations V with integer delays D and a partial order on the operations E**
- **Find an integer labeling of the operations $\varphi : V \rightarrow Z^{+}$, such that**
  - $t_i = \varphi(v_i)$,
  - $t_i \geq t_j + d_j \quad \forall$ i, j s.t. $(v_j, v_i) \in E$
  - and t_n is *minimum*.
- **Unconstrained scheduling used when**
  - Dedicated resources are used.
  - Operations differ in type.
  - Operations cost is marginal when compared to that of steering logic, registers, wiring, and control logic.
  - Binding is done before scheduling: resource conflicts solved by serializing operations sharing same resource.
  - Deriving bounds on latency for constrained problems.

ASAP Scheduling Algorithm

- Denote by *t^s* the start times computed by the *as soon as possible* (ASAP) algorithm.
- Yields *minimum* values of start times.

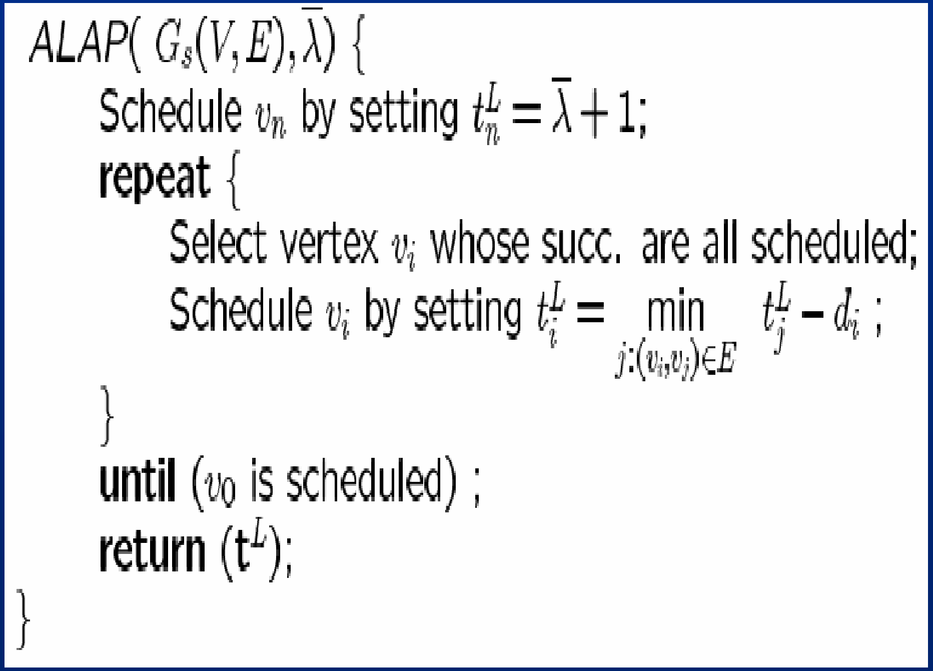


```
ASAP ( $G_s(V,E)$ ) {
    Schedule $v_0$ by setting $t_0^S = 1$;
    repeat {
        Select a vertex $v_i$ whose pred. are all scheduled;
        Schedule $v_i$ by setting $t_i^S = \max_{j:(v_j,v_i)\in E} t_j^S + d_j$;
    }
    until ($v_n$ is scheduled) ;
    return ($t^S$);
}
```

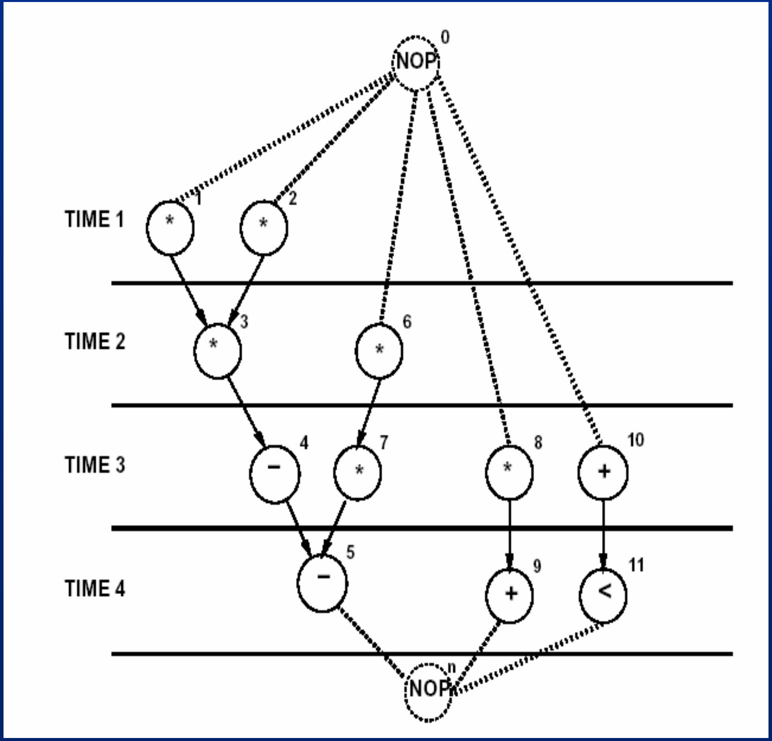


ALAP Scheduling Algorithm

- Denote by t^L the start times computed by the *as late as possible* (ALAP) algorithm.
- Yields *maximum* values of start times.
- Latency upper bound $\overline{\lambda}$



```
ALAP( $G_s(V,E),\overline{\lambda}$) {
    Schedule $v_n$ by setting $t_n^L = \overline{\lambda}+1$;
    repeat {
        Select vertex $v_i$ whose succ. are all scheduled;
        Schedule $v_i$ by setting $t_i^L = \min_{j:(v_i,v_j)\in E} t_j^L - d_i$ ;
    }
    until ($v_0$ is scheduled) ;
    return (t$^L$);
}
```

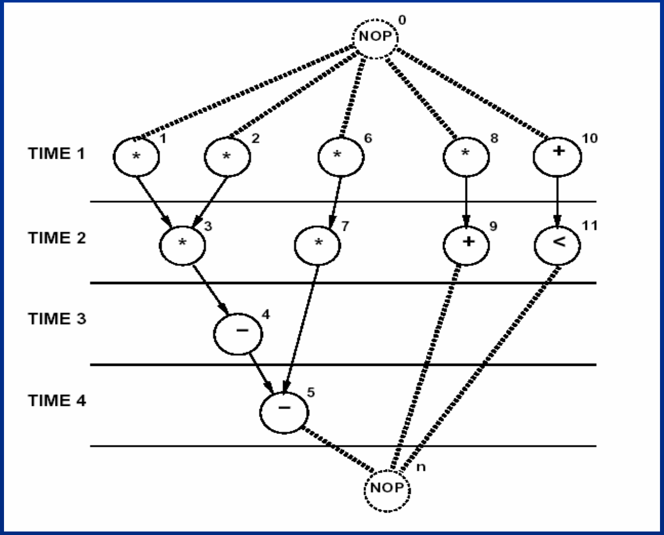


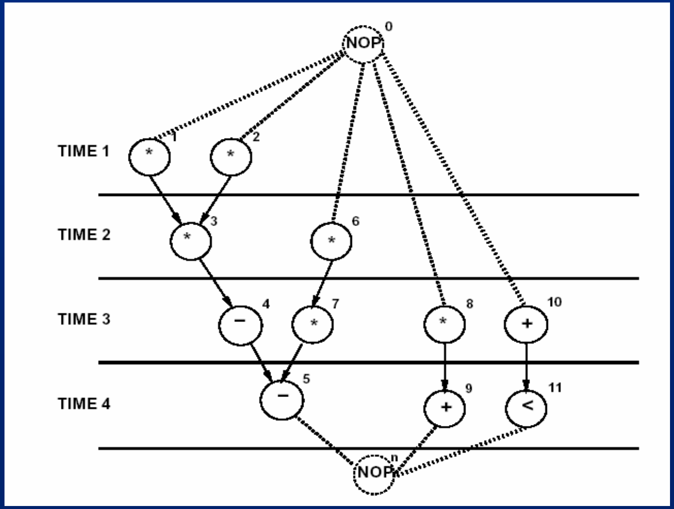
Latency-Constrained Scheduling

- **ALAP solves a latency-constrained problem.**
- **Latency bound can be set to latency computed by ASAP algorithm.**
- **Mobility**
  - Defined for each operation.
  - Difference between ALAP and ASAP schedule.
  - Zero mobility implies that an operation can be started only at one given time step.
  - Mobility greater than 0 measures span of time interval in which an operation may start.
- **Slack on the start time.**

Example

- Operations with zero mobility
  - **{v1, v2, v3, v4, v5}.**
  - **Critical path.**
- Operations with mobility one
  - **{v6, v7}.**
- Operations with mobility two
  - **{v8, v9, v10, v11}**





Minimum Latency Resource-Constrained Scheduling Problem

- Given a set of ops V with integer delays D, a partial order on the operations E, and upper bounds $\{a_k; k = 1, 2, \ldots, n_{res}\}$
- Find an integer labeling of the operations $\varphi : V \to Z^+$, **such that**
  - $t_i = \varphi(v_i)$,
  - $t_i \geq t_j + d_j \quad \forall$ i, j s.t. $(v_j, v_i) \in E$

$$|\{v_i | \mathcal{T}(v_i) = k \text{ and } t_i \leq l < t_i + d_i\}| \leq a_k$$
$$\forall \text{types } k = 1, 2, \ldots, n_{res} \text{ and } \forall \text{ steps } l$$

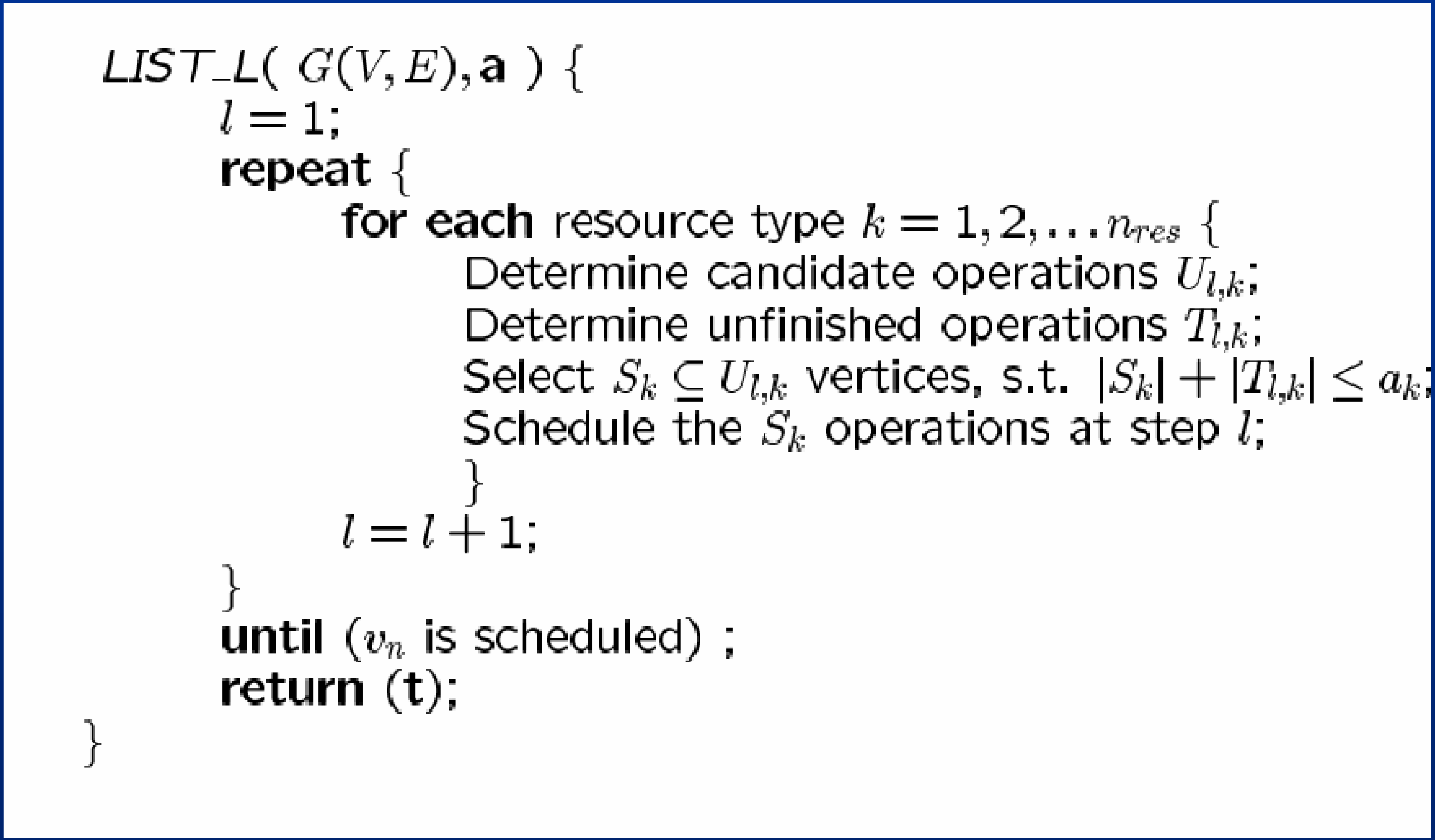
$$\mathcal{T} : V \to \{1,2, \ldots n_{res}\}$$

  - and t_n is *minimum*.
- **Number of operations of any given type in any schedule step does not exceed bound.**

List Scheduling Algorithms

- **Heuristic method for**
  - Minimum latency subject to resource bound.
  - Minimum resource subject to latency bound.
- **Greedy strategy.**
- **Priority list heuristics.**
  - Assign a weight to each vertex indicating its scheduling priority
    - Longest path to sink.
    - Longest path to timing constraint.

List Scheduling Algorithm for Minimum Latency …



```
$LIST\_L(\ G(V,E), \mathbf{a}\ )$ {
    $l = 1$;
    repeat {
        for each resource type $k = 1, 2, \ldots n_{res}$ {
            Determine candidate operations $U_{l,k}$;
            Determine unfinished operations $T_{l,k}$;
            Select $S_k \subseteq U_{l,k}$ vertices, s.t. $|S_k| + |T_{l,k}| \leq a_k$
            Schedule the $S_k$ operations at step $l$;
            }
        $l = l + 1$;
    }
    until ($v_n$ is scheduled) ;
    return ($\mathbf{t}$);
}
```

… List Scheduling Algorithm for Minimum Latency

- **Candidate Operations $U_{l,k}$**
  - Operations of type *k* whose predecessors are scheduled and completed at time step before *l*

$$U_{l,k} = \{v_i \in V : Type(v_i) = k \textbf{ and } t_j + d_j \leq l \ \forall j : (v_j, v_i) \in E\}$$

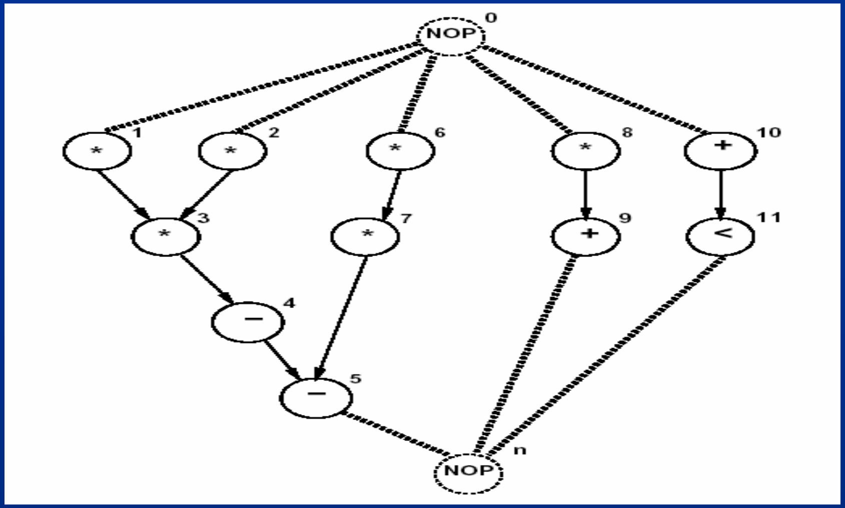
- **Unfinished operations $T_{l,k}$ are operations of type *k* that started at earlier cycles and whose execution is not finished at time *l***

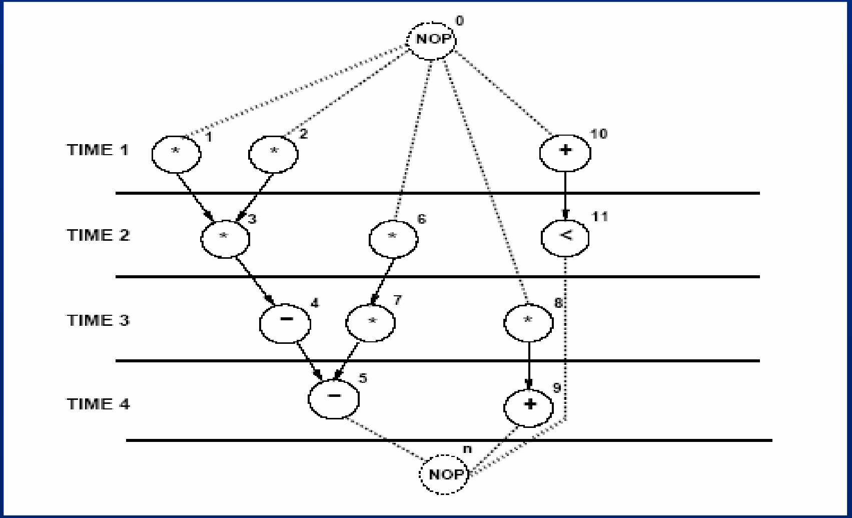
$$T_{l,k} = \{v_i \in V : Type(v_i) = k \textbf{ and } t_j + d_j > l \ \forall j : (v_j, v_i) \in E\}$$

  - Note that when execution delays are 1, $T_{l,k}$ is empty.

Example

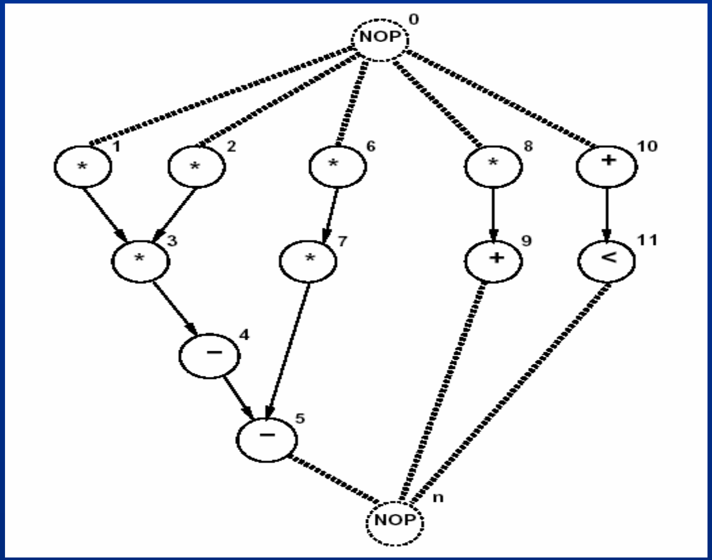
- Assumptions
  - **a_1 = 2 multipliers with delay 1.**
  - **a_2 = 2 ALUs with delay 1.**
- **First Step**
  - $U_{1,1} = \{v_1, v_2, v_6, v_8\}$
  - Select $\{v_1, v_2\}$
  - $U_{1,2} = \{v_{10}\}$; selected
- **Second step**
  - $U_{2,1} = \{v_3, v_6, v_8\}$
  - select $\{v_3, v_6\}$
  - $U_{2,2} = \{v_{11}\}$; selected
- **Third step**
  - $U_{3,1} = \{v_7, v_8\}$
  - Select $\{v_7, v_8\}$
  - $U_{3,2} = \{v_4\}$; selected
- **Fourth step**
  - $U_{4,2} = \{v_5, v_9\}$; selected

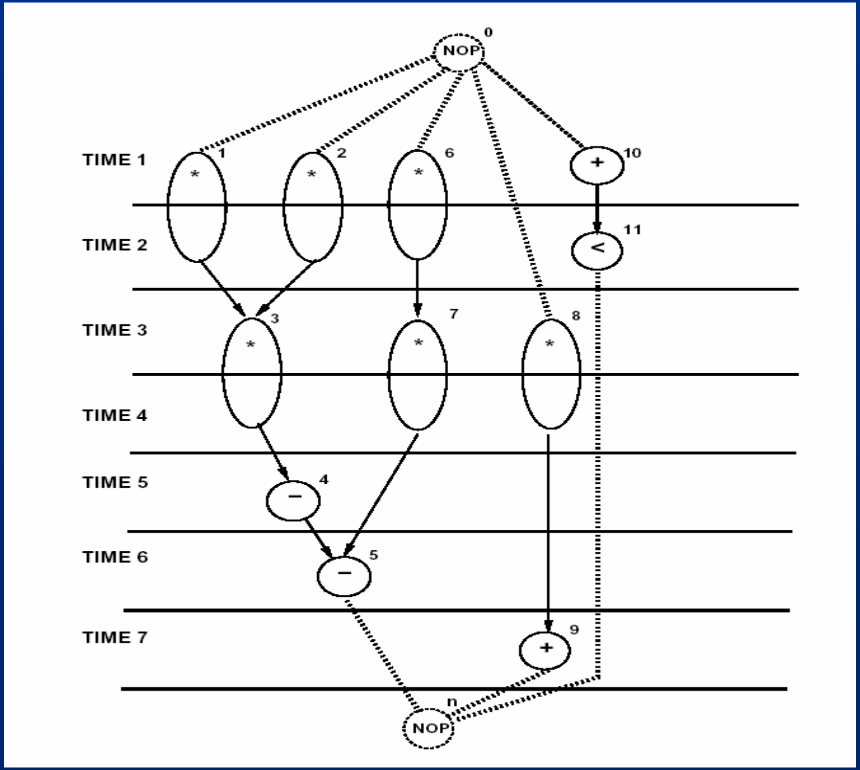


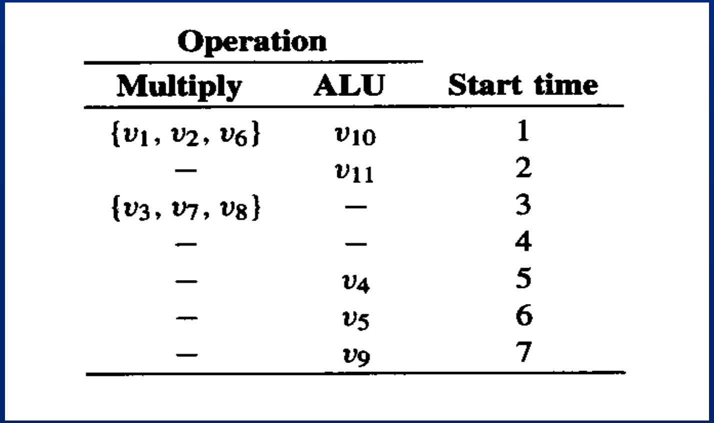


Example

- Assumptions
  - a_1 = 3 multipliers with delay 2.
  - a_2 = 1 ALU with delay 1.

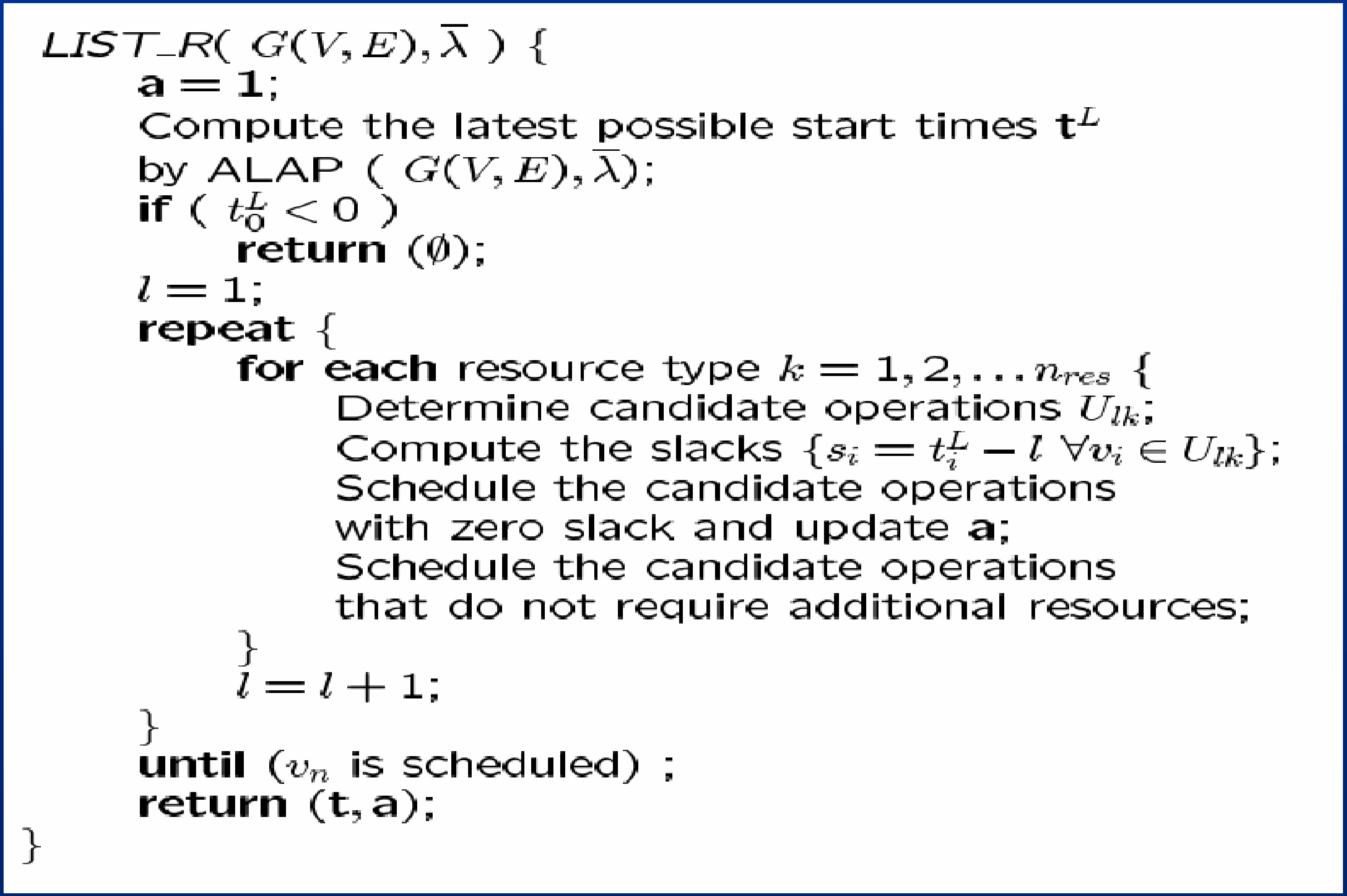






Operation		
Multiply	ALU	Start time
$\{v_1, v_2, v_6\}$	v_{10}	1
—	v_{11}	2
$\{v_3, v_7, v_8\}$	—	3
—	—	4
—	v_4	5
—	v_5	6
—	v_9	7

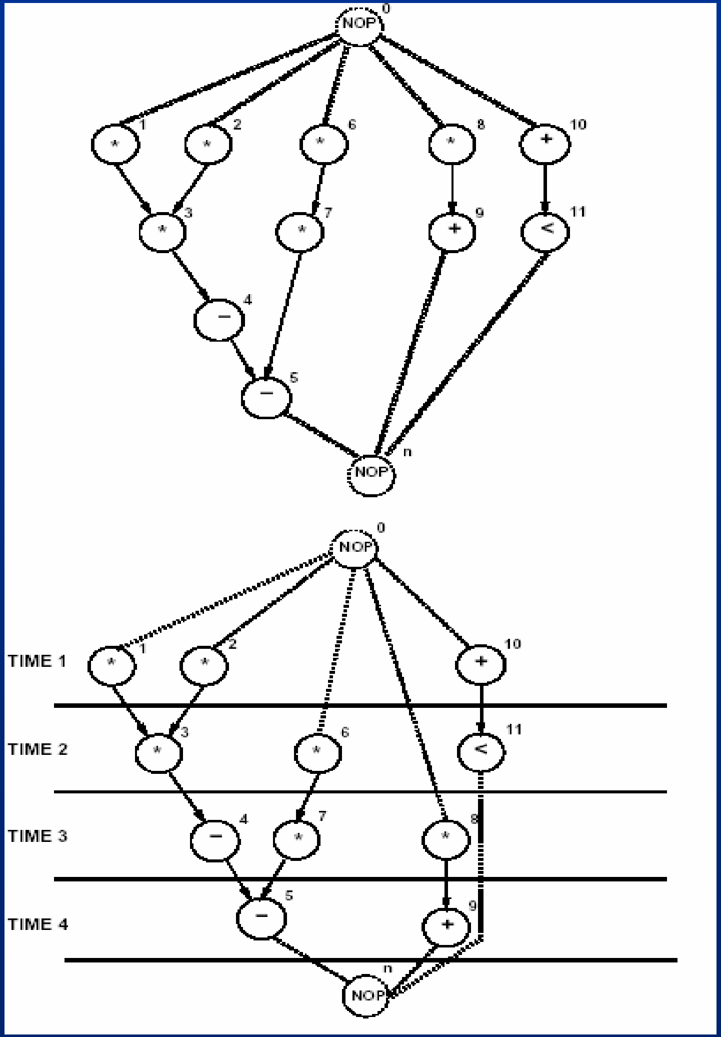
List Scheduling Algorithm for Minimum Resource Usage



```
$LIST\_R( G(V,E),\overline{\lambda} )$ {
    **a** $= \mathbf{1}$;
    Compute the latest possible start times $\mathbf{t}^L$
    by ALAP ( $G(V,E),\overline{\lambda}$);
    **if** ( $t_0^L < 0$ )
        **return** ($\emptyset$);
    $l = 1$;
    **repeat** {
        **for each** resource type $k = 1,2,\ldots n_{res}$ {
            Determine candidate operations $U_{lk}$;
            Compute the slacks $\{s_i = t_i^L - l \ \forall v_i \in U_{lk}\}$;
            Schedule the candidate operations
            with zero slack and update **a**;
            Schedule the candidate operations
            that do not require additional resources;
        }
        $l = l + 1$;
    }
    **until** ($v_n$ is scheduled) ;
    **return** ($\mathbf{t},\mathbf{a}$);
}
```

Example

- **Assume λ=4**
- **Let a = $[1, 1]^T$**
- **First Step**
  - $U_{1,1} = \{v_1, v_2, v_6, v_8\}$
  - Operations with zero slack $\{v_1, v_2\}$
  - a = $[2, 1]^T$
  - $U_{1,2} = \{v_{10}\}$
- **Second step**
  - $U_{2,1} = \{v_3, v_6, v_8\}$
  - Operations with zero slack $\{v_3, v_6\}$
  - $U_{2,2} = \{v_{11}\}$
- **Third step**
  - $U_{3,1} = \{v_7, v_8\}$
  - Operations with zero slack $\{v_7, v_8\}$
  - $U_{3,2} = \{v_4\}$
- **Fourth step**
  - $U_{4,2} = \{v_5, v_9\}$
  - Both have zero slack; a = $[2, 2]^T$

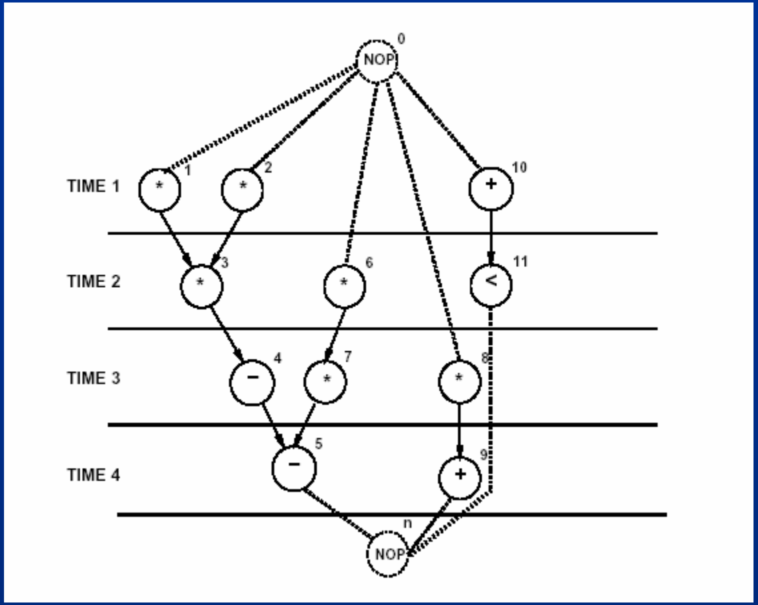


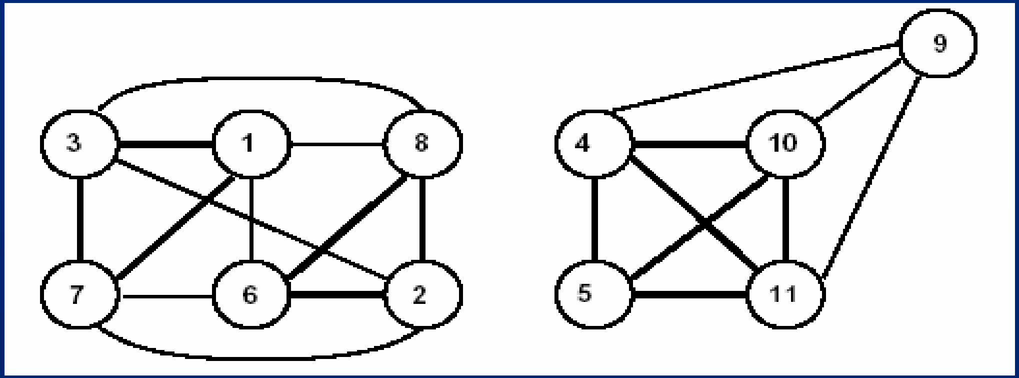
Allocation and Binding

- **Allocation**
  - Determine number of resources needed
- **Binding**
  - Mapping between operations and resources.
- **Sharing**
  - Assignment of a resource to more than one operation.
- **Optimum binding/sharing**
  - Minimize the resource usage.

Compatibility and Conflicts

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



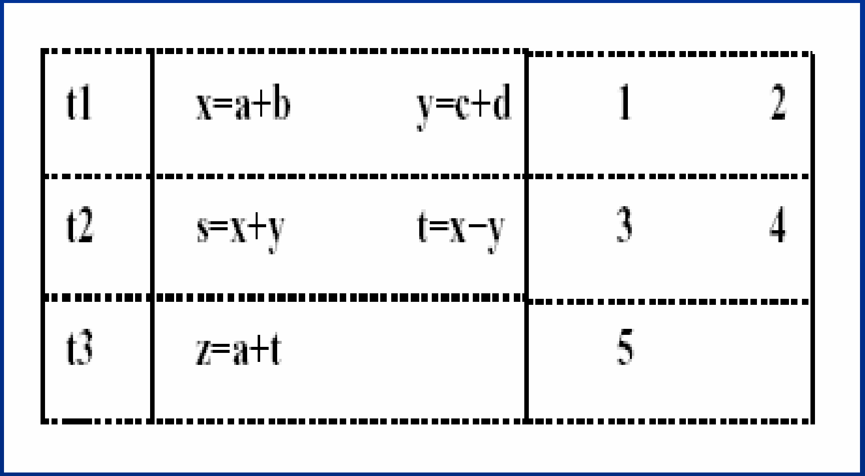


Multiplier *ALU*

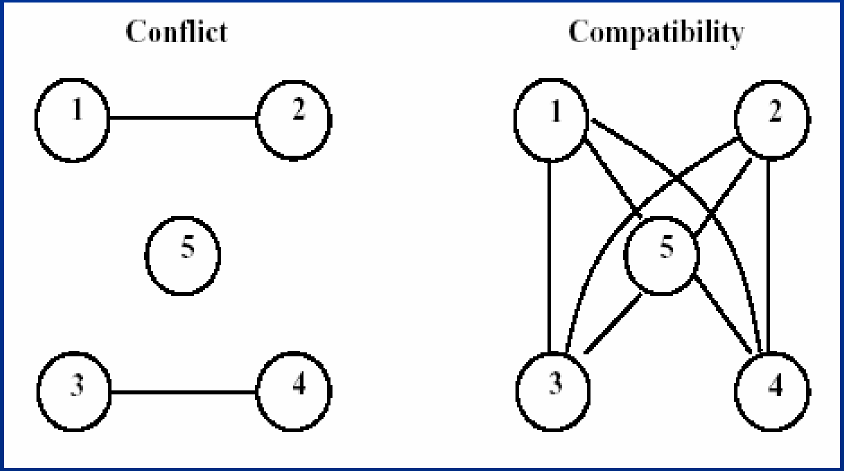
Algorithmic Solution to the Optimum Binding Problem

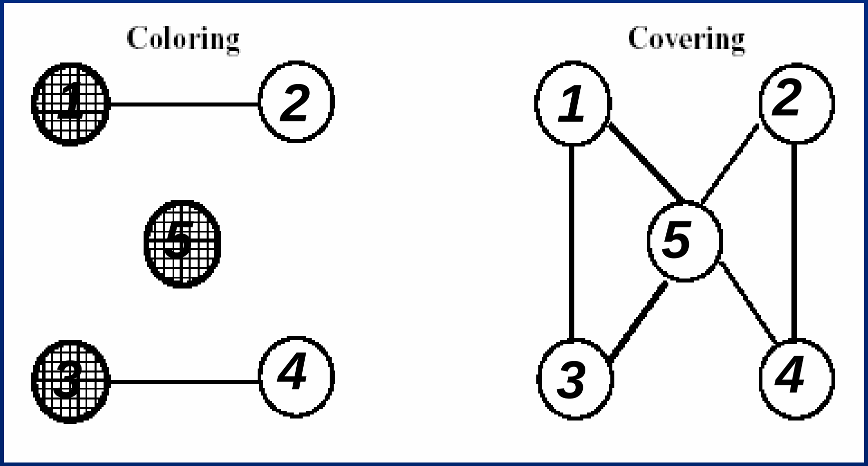
- **Compatibility graph.**
  - Partition the graph into a minimum number of cliques.
  - Find clique cover number.
- **Conflict graph.**
  - Color the vertices by a minimum number of colors.
  - Find chromatic number.
- **NP-complete problems - Heuristic algorithms.**

Example



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





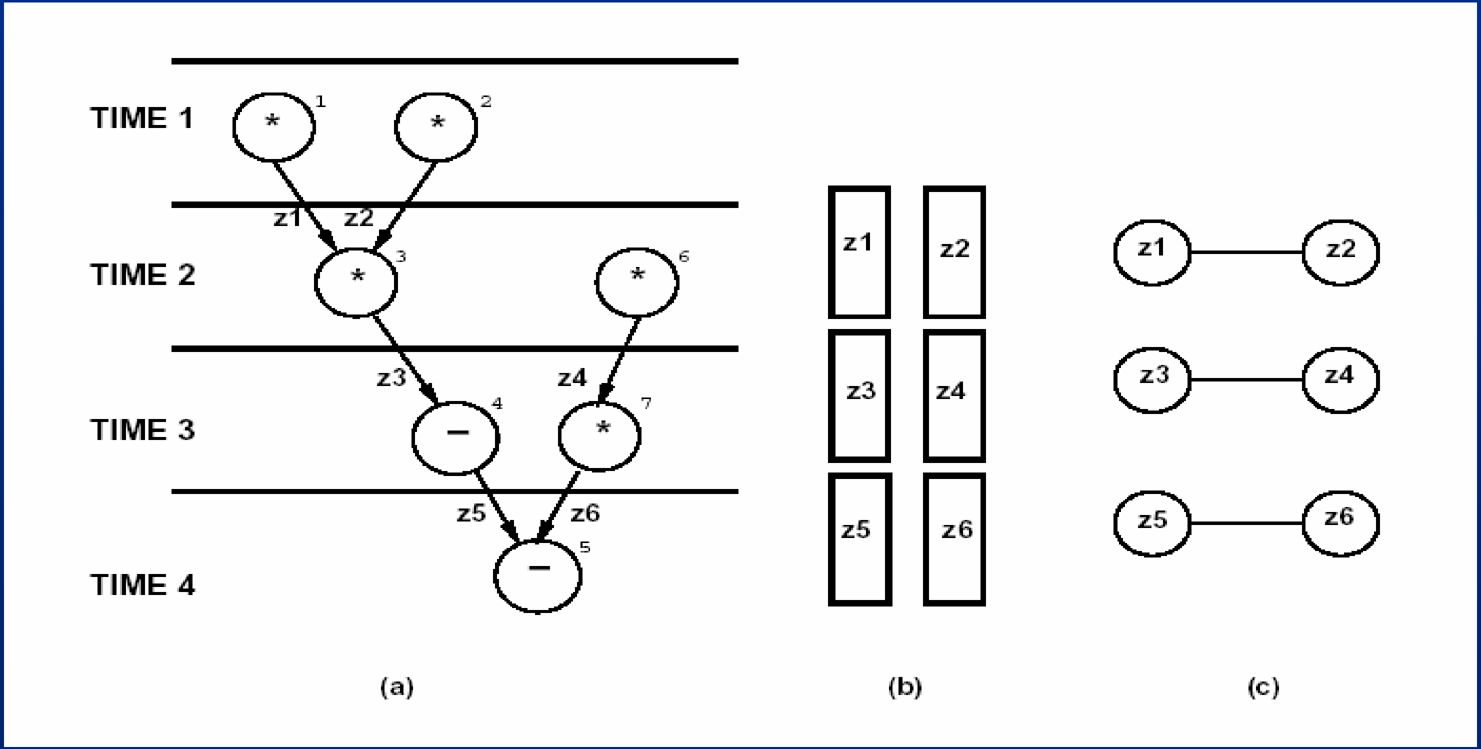
ALU1: 1, 3, 5
ALU2: 2, 4

Register Binding Problem

- **Given a schedule**
  - Lifetime intervals for variables.
  - Lifetime overlaps.
- **Conflict graph (interval graph).**
  - Vertices $\leftrightarrow$ variables.
  - Edges $\leftrightarrow$ overlaps.
- **Find minimum number of registers storing all the variables.**
- **Compatibility graph.**

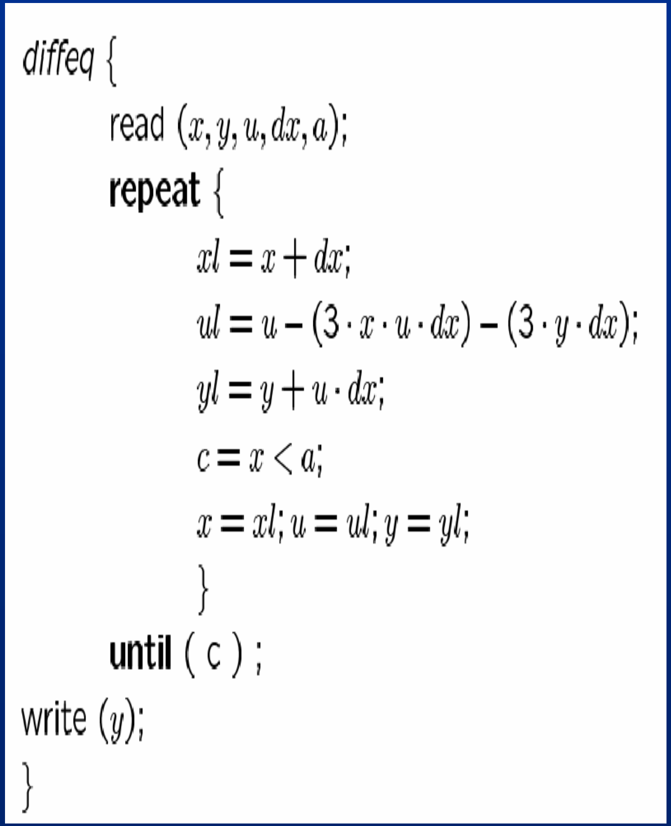
Example

- Six intermediate variables that need to be stored in registers {z1, z2, z3, z4, z5, z6}
- Six variables can be stored in two registers

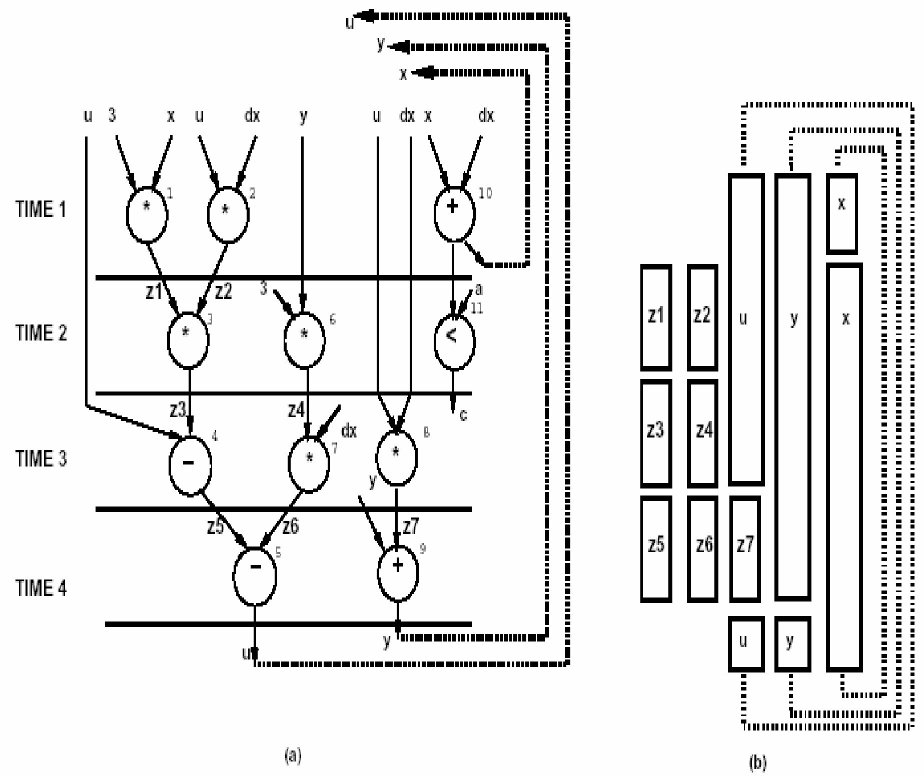


Example

- 7 intermediate variables, 3 loop variables, 3 loop invariants
- 5 registers suffice to store 10 intermediate loop variables



```
diffeq {
    read $(x, y, u, dx, a)$;
    repeat {
        $xl = x + dx$;
        $ul = u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx)$;
        $yl = y + u \cdot dx$;
        $c = x < a$;
        $x = xl; u = ul; y = yl$;
    }
    until ( c ) ;
write $(y)$;
}
```


TIME 1
TIME 2
TIME 3
TIME 4
u 3 x u dx y u dx x dx
z1 z2 z3 z4 z5 z6 z7
a
c
u
y
x
(a)
(b)