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]

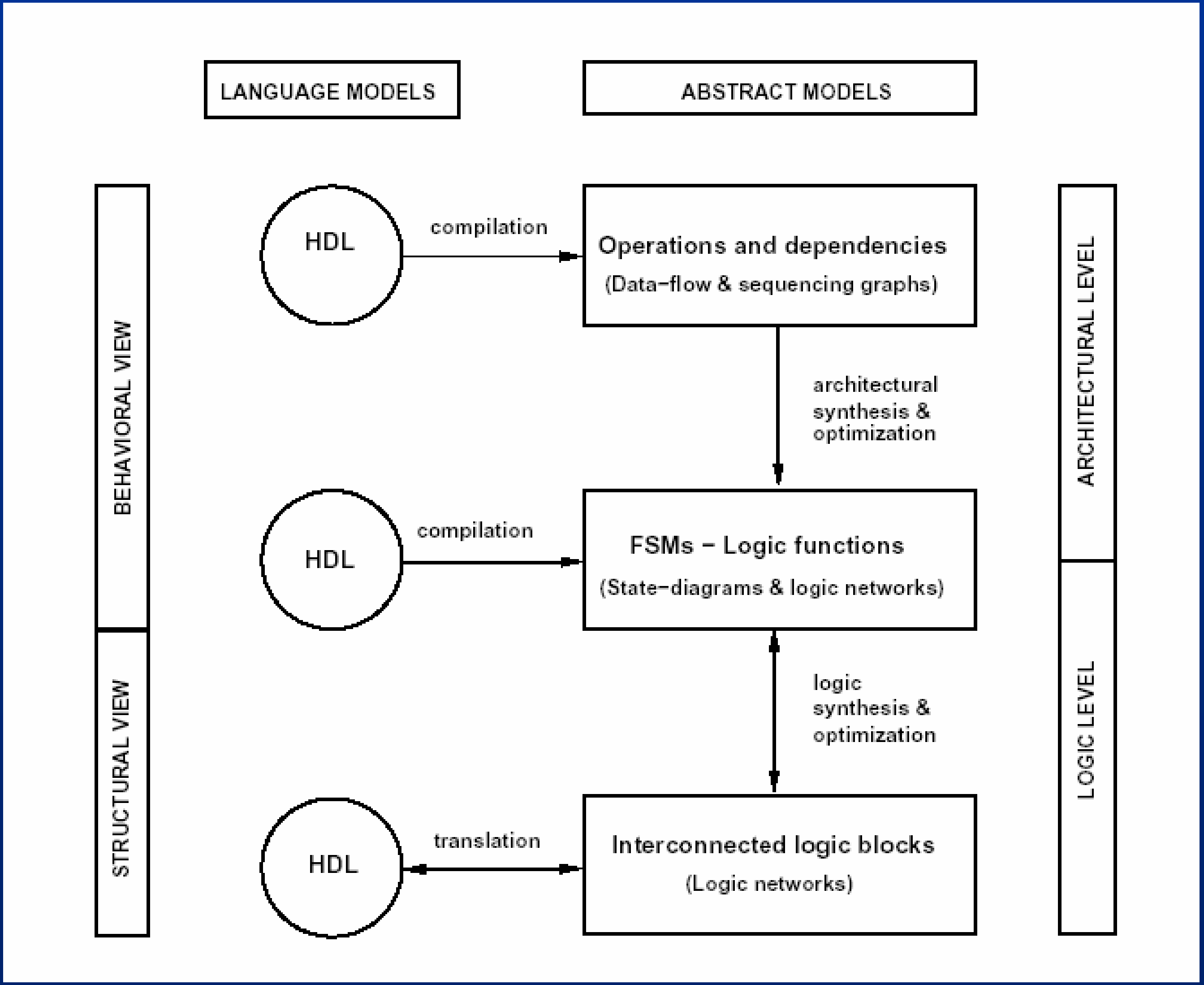
Outline

- Motivation
- Dataflow graphs
- Sequencing graphs
- Compilation and behavioral optimization
- Resources
- Constraints
- Synthesis in temporal domain: Scheduling
- Synthesis in spatial domain: Binding

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

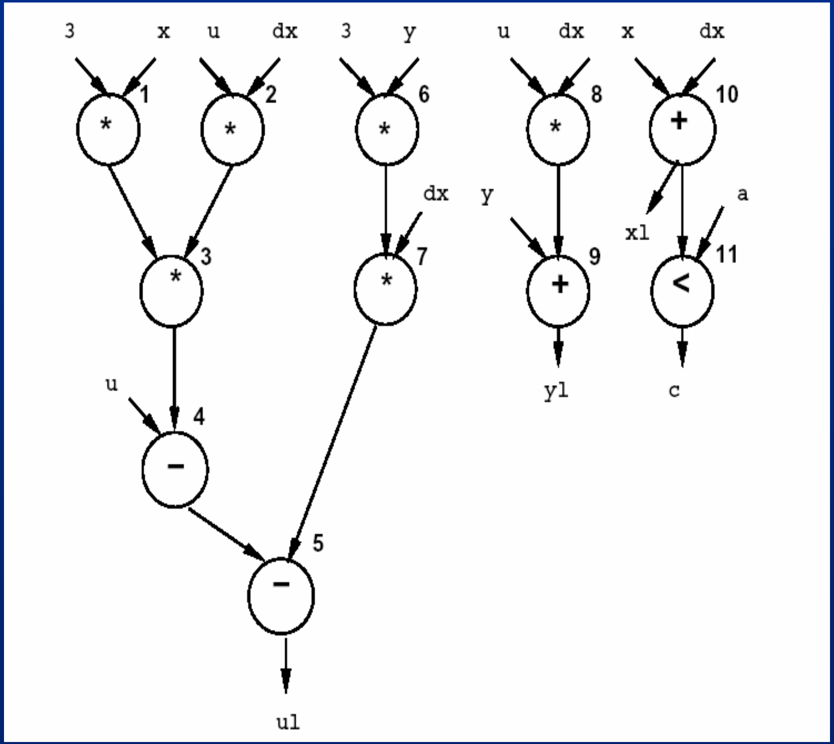


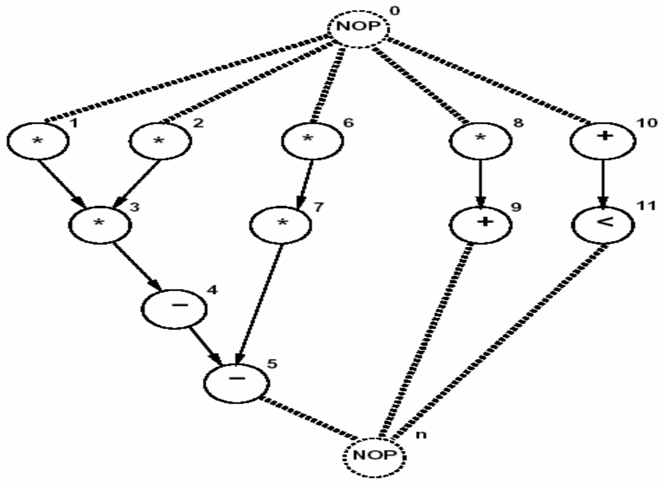
LANGUAGE MODELS
ABSTRACT MODELS
BEHAVIORAL VIEW
STRUCTURAL VIEW
HDL
compilation
Operations and dependencies
(Data-flow & sequencing graphs)
architectural synthesis & optimization
HDL
compilation
FSMs – Logic functions
(State-diagrams & logic networks)
logic synthesis & optimization
HDL
translation
Interconnected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

Architectural Design Space Example …

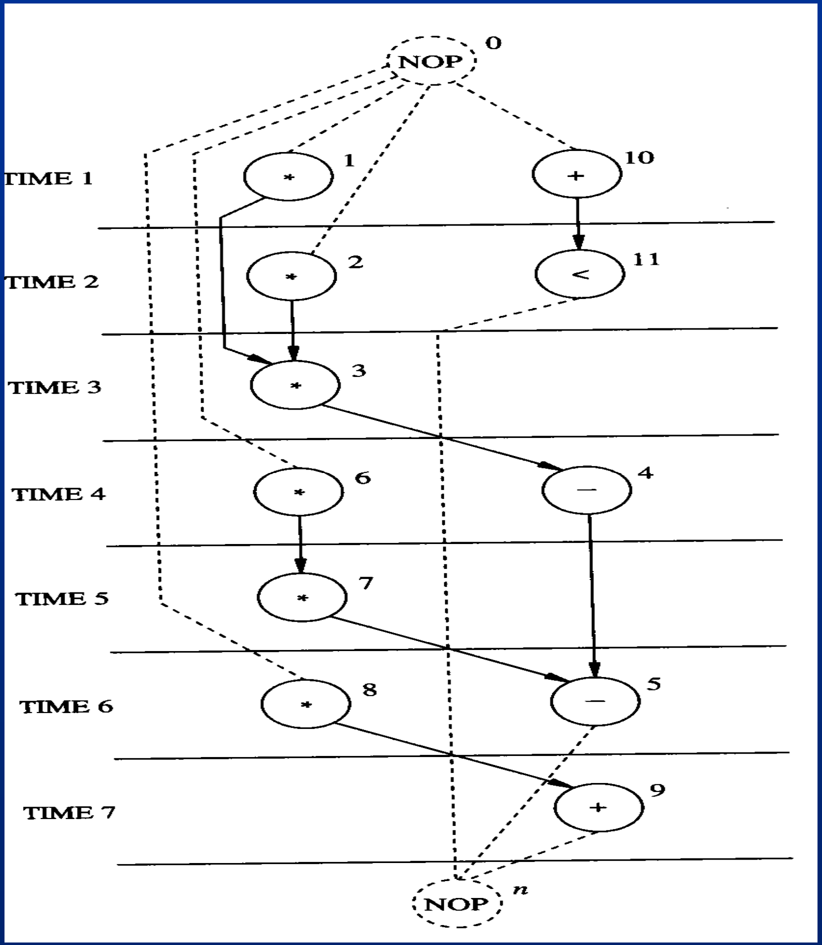
```
diffeq {
        read ($x, y, u, dx, a$);
        repeat {
                $xl = x + dx$;
                $ul = u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx)$;
                $yl = y + u \cdot dx$;
                $c = x < a$;
                $x = xl$; $u = ul$; $y = yl$;
            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

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

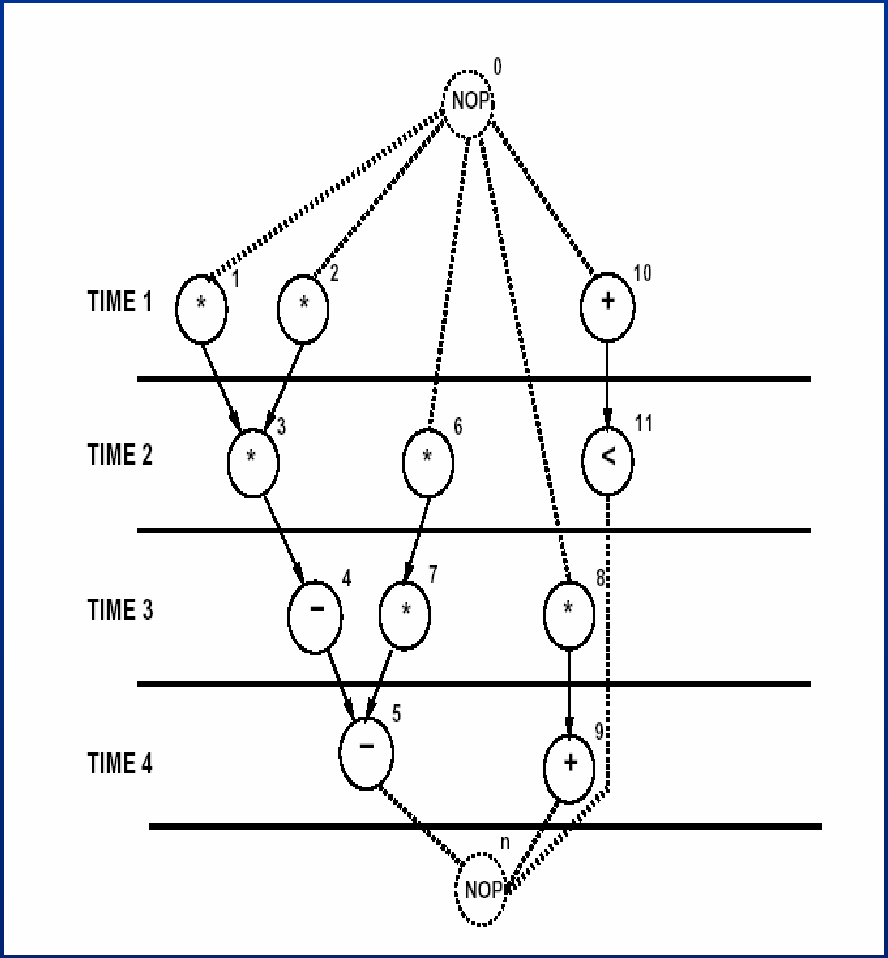




Different Design Solutions

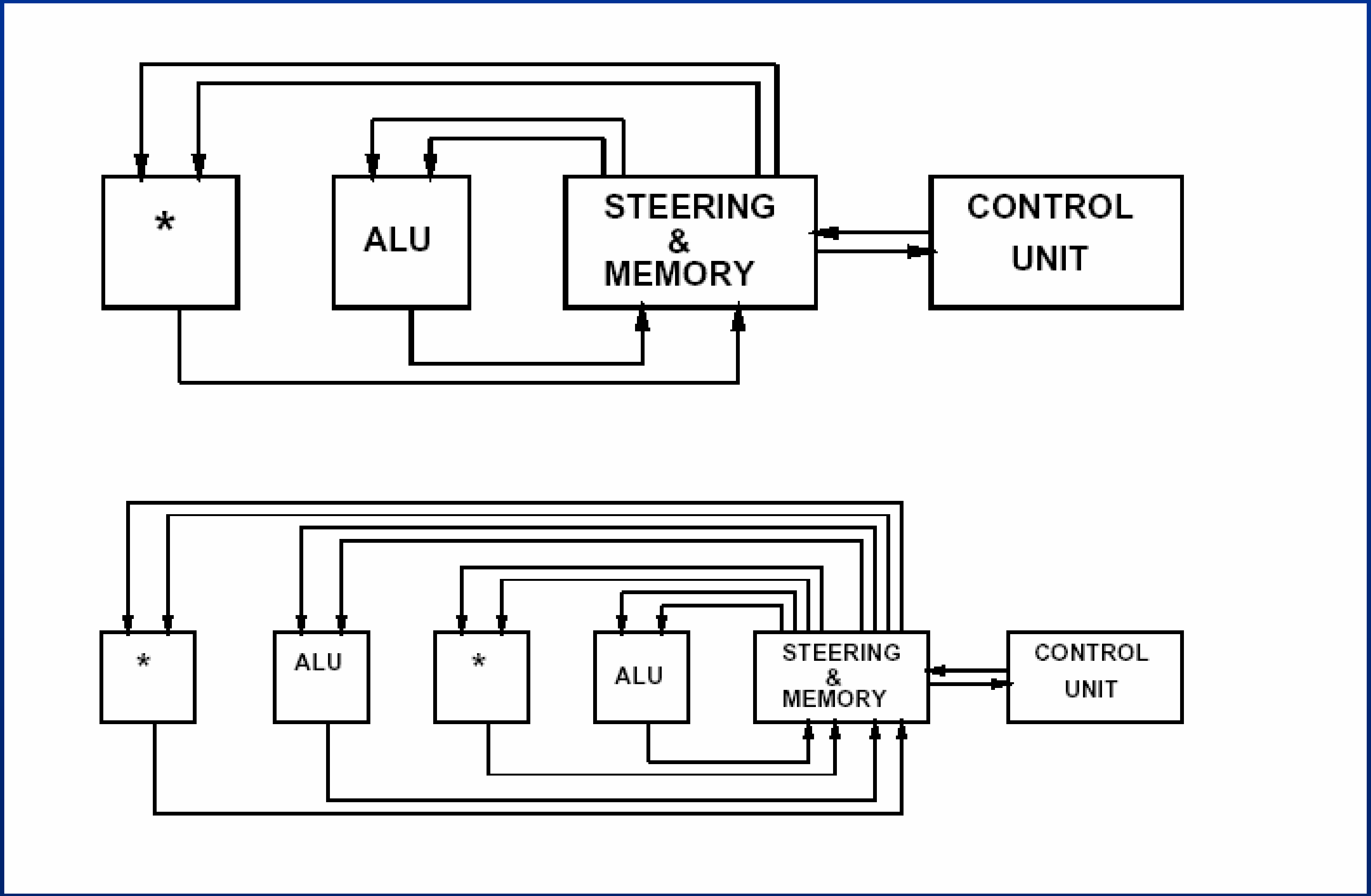


1 Multiplier , 1 ALU



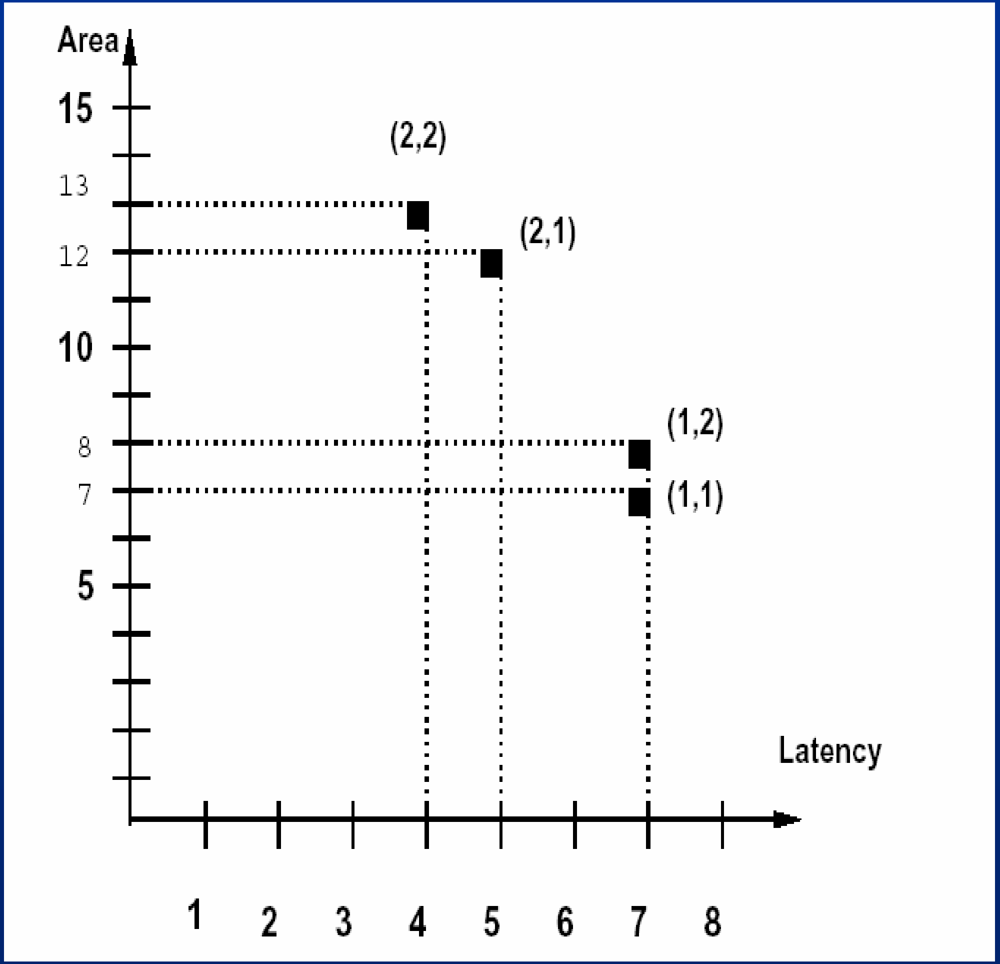
2 Multipliers, 2 ALUs

Example of Structures



Area vs. Latency Tradeoffs

Multiplier Area: 5
Adder Area: 1
Other logic Area: 1



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.

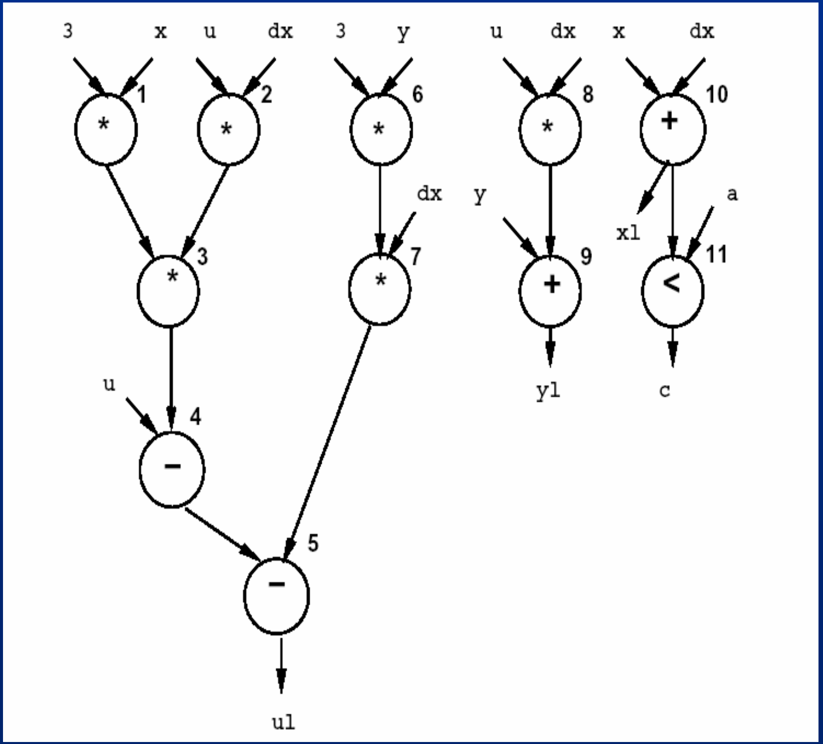
Architectural-Level Synthesis

- **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.

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

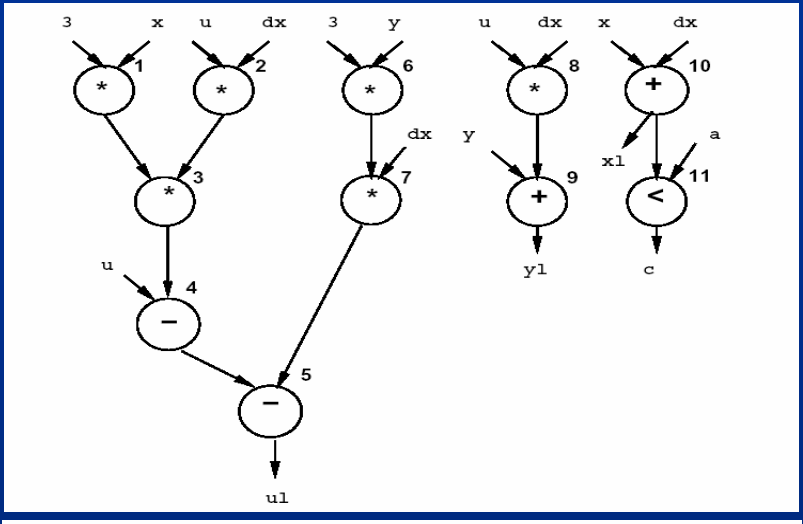


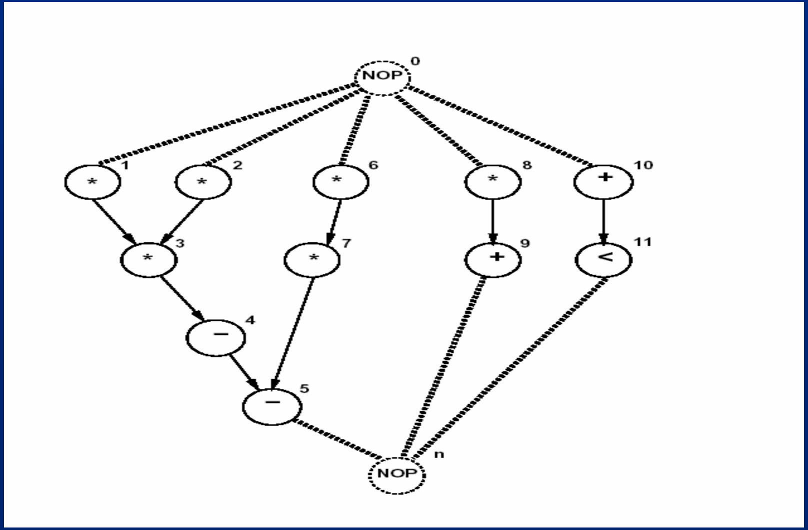
… Dataflow Graphs

- 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 operation.
- Values must be preserved during life-time.

Sequencing Graphs

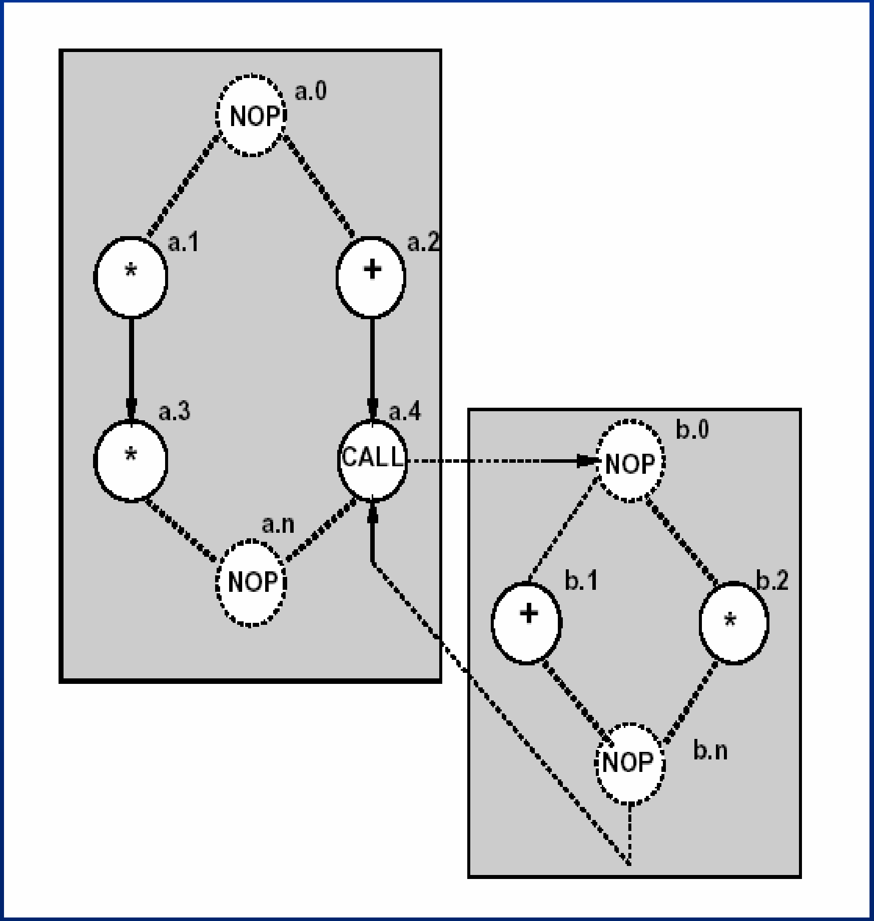
- **Useful to represent data-path and control.**
- **Extended dataflow graphs**
  - Control Data Flow Graphs (CDFGs).
  - Operation serialization.
  - Hierarchy.
  - Control-flow commands
    - branching and iteration.
  - Polar: source and sink.
- **Paths in the graph represent concurrent streams of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





Example of Hierarchy

- Two kinds of vertices
  - Operations
  - Links: linking sequencing graph entities in the hierarchy
    - Model call
    - Branching
    - Iteration
- **Vertex v_i is a *predecessor* of vertex v_j if there is a path with tail v_i and head v_j**
- **Vertex v_i is a *successor* of vertex v_j if there is a path with head v_i and tail v_j**

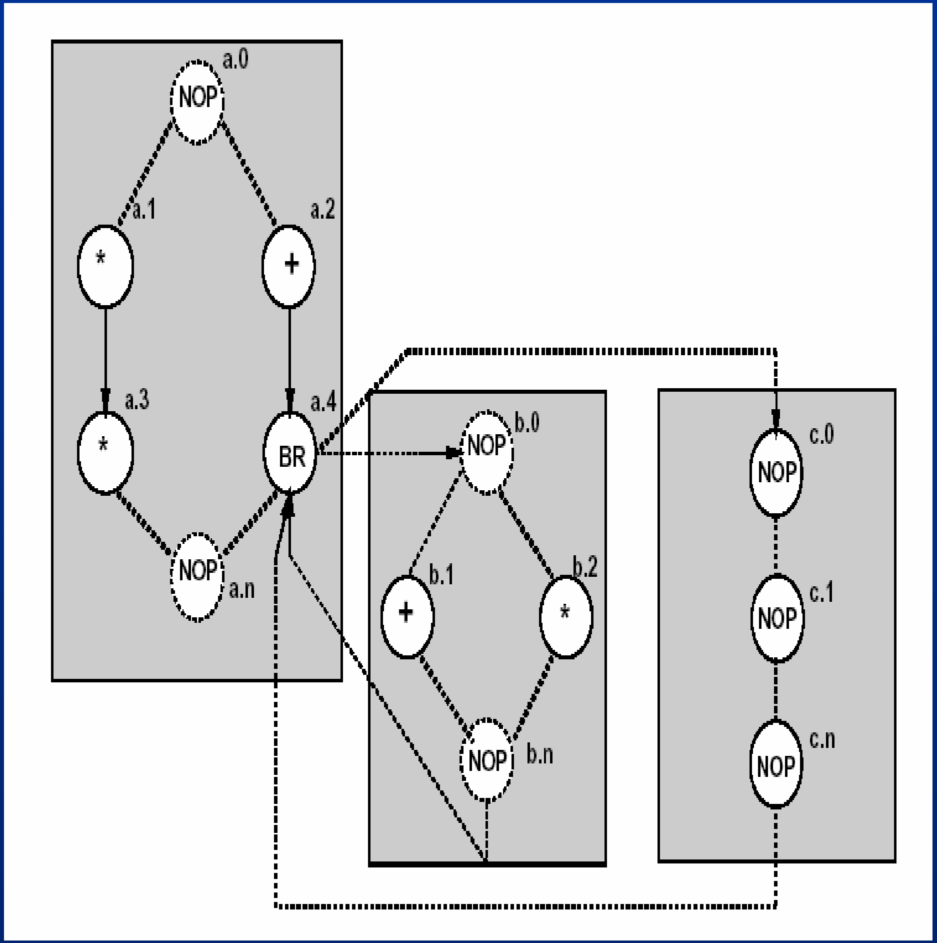


Example of Branching …

- **Branching modeled by**
  - Branching clause
  - Branching body
    - Set of tasks selected according to value of branching clause.
- **Several branching bodies**
  - Mutual exclusive execution.
- **A sequencing graph entity associated with each branch body.**
- **Link vertex models**
  - Branching clause.
  - Operation of evaluating clause and taking branch decision.

… Example of Branching

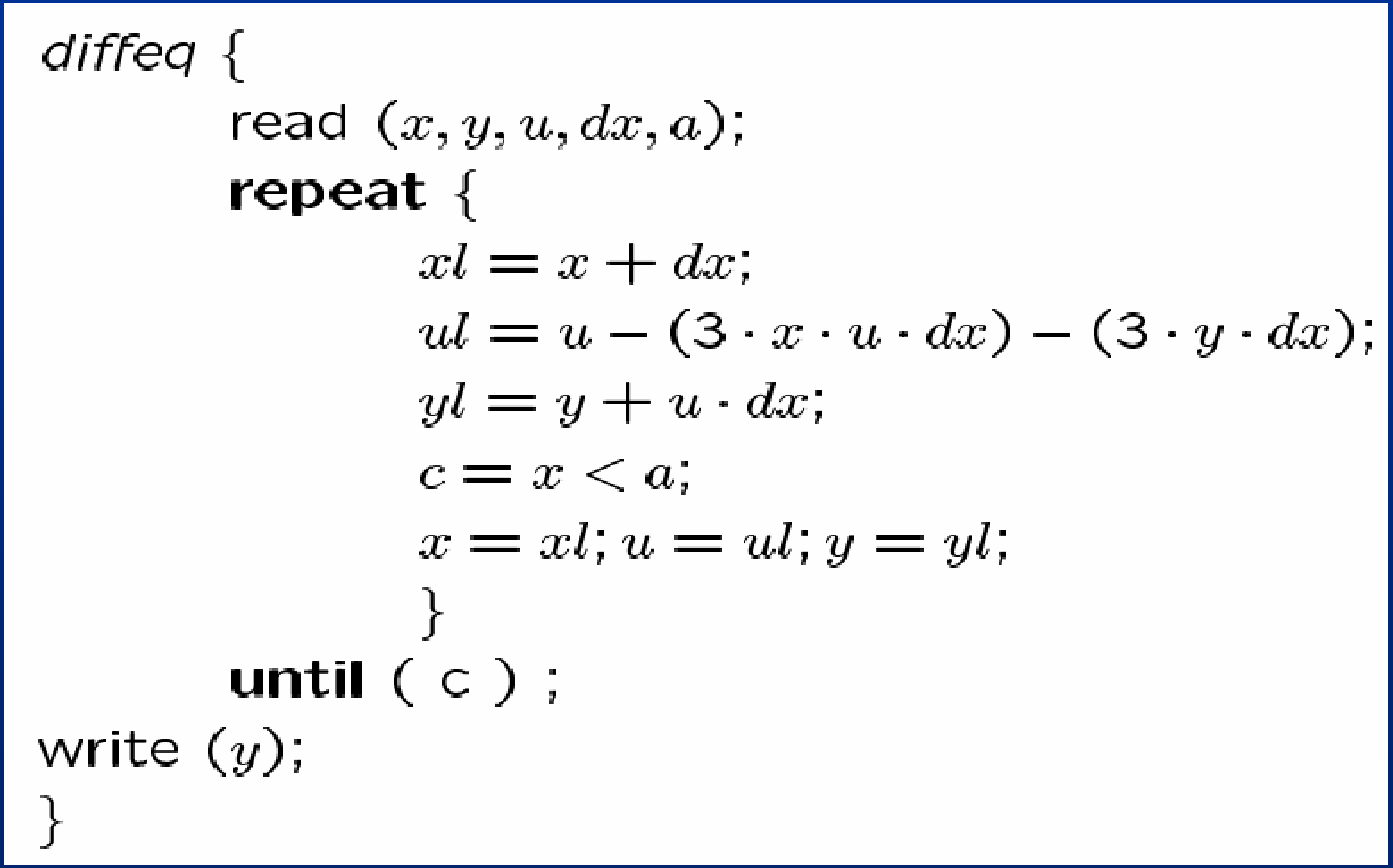
- x= a*b
- y=x*c
- z=a+b
- If (z ≥ 0)
  - {p=m+n; q=m*n}



Iterative Constructs

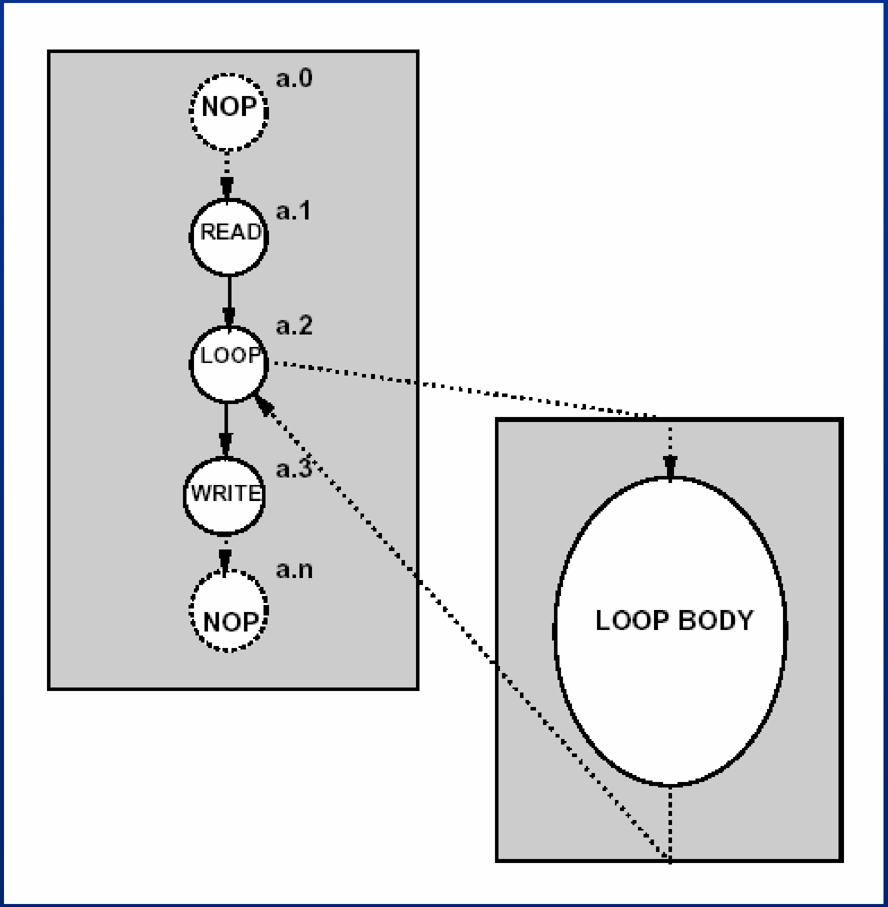
- **Iterative constructs modeled by**
  - Iteration clause
  - Iteration body
- **Iteration body is a set of tasks repeated as long as iteration clause is true.**
- **Iteration modeled through use of hierarchy.**
- **Iteration represented as repeated model call to sequencing graph entity modeling iteration body.**
- **Link vertex models the operation of evaluating the iteration cause.**

Example of Iteration …

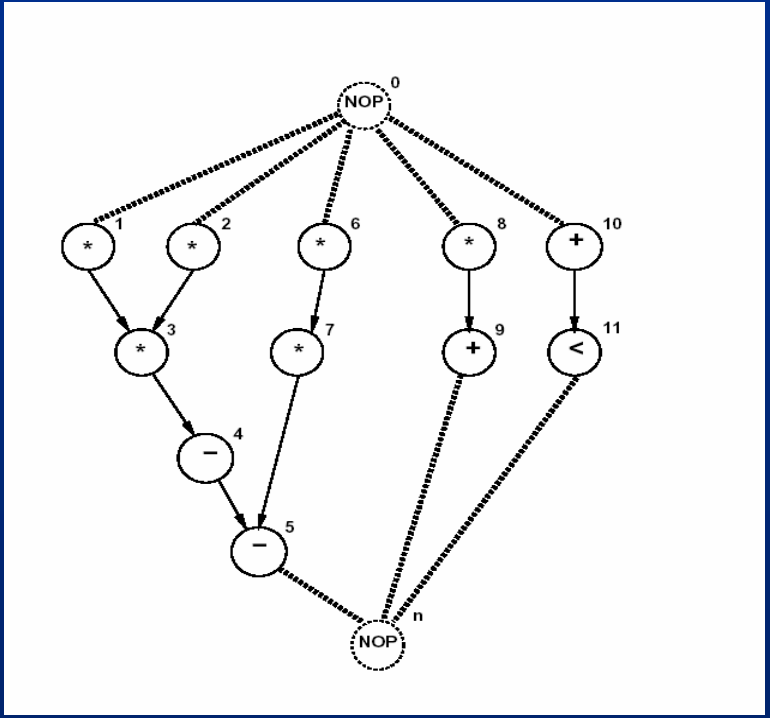


```
diffeq {
        read $(x, y, u, dx, a)$;
        repeat {
                $xl = x + dx$;
                $ul = u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx)$;
                $yl = y + u \cdot dx$;
                $c = x < a$;
                $x = xl; u = ul; y = yl$;
                }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

… Example of Iteration



Loop Body



Semantics of Sequencing Graphs

- **Marking of vertices**
  - Waiting for execution.
  - Executing.
  - Have completed execution.
- **Firing an operation means starting its execution.**
- **Execution semantics**
  - An operation can be fired as soon as all its immediate predecessors have completed execution.
- **Model can be reset by making all operations waiting for execution.**
- **Model can be fired (executed) by firing the source vertex.**
- **Model completes execution when sink completes execution.**

Vertex Attributes

- **Area cost.**
- **Delay cost**
  - Propagation delay.
  - Execution delay.
- **Data-dependent execution delays**
  - Bounded (e.g. branching).
    - Maximum and minimum delays can be computed
    - E.g. floating-point data normalization requiring conditional data alignment.
  - Unbounded (e.g. iteration, synchronization).

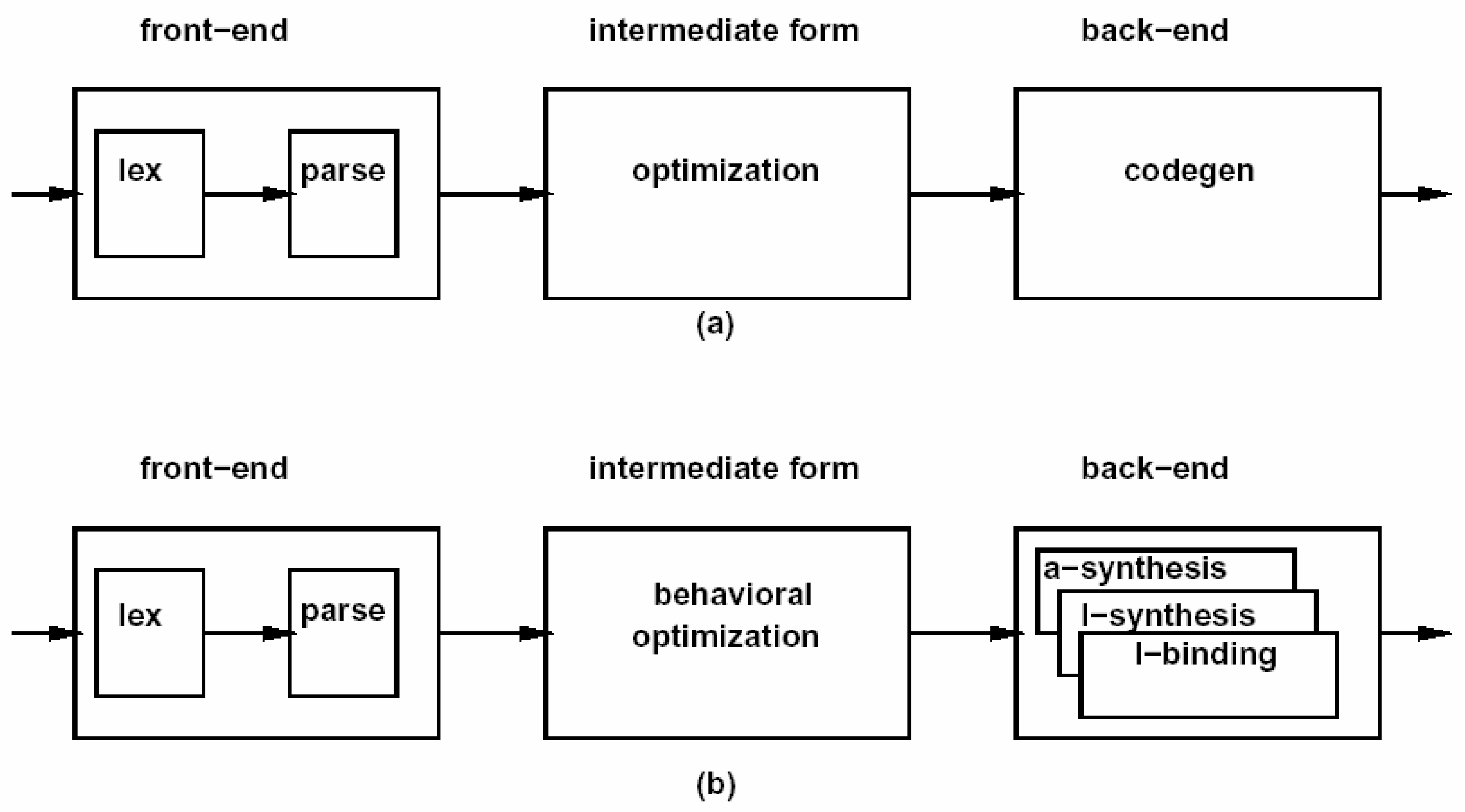
Properties of Sequencing Graphs

- **Computed by visiting hierarchy bottom-up.**
- **Area estimate**
  - Sum of the area attributes of all vertices.
  - Worst-case -- no sharing.
- **Delay estimate (latency)**
  - Bounded-latency graphs.
  - Length of longest path.

Compilation and Behavioral Optimization

- **Software compilation**
  - Compile program into intermediate form.
  - Optimize intermediate form.
  - Generate target code for an architecture.
- **Hardware compilation**
  - Compile HDL model into sequencing graph.
  - Optimize sequencing graph.
  - Generate gate-level interconnection for a cell library.

Hardware and Software Compilation

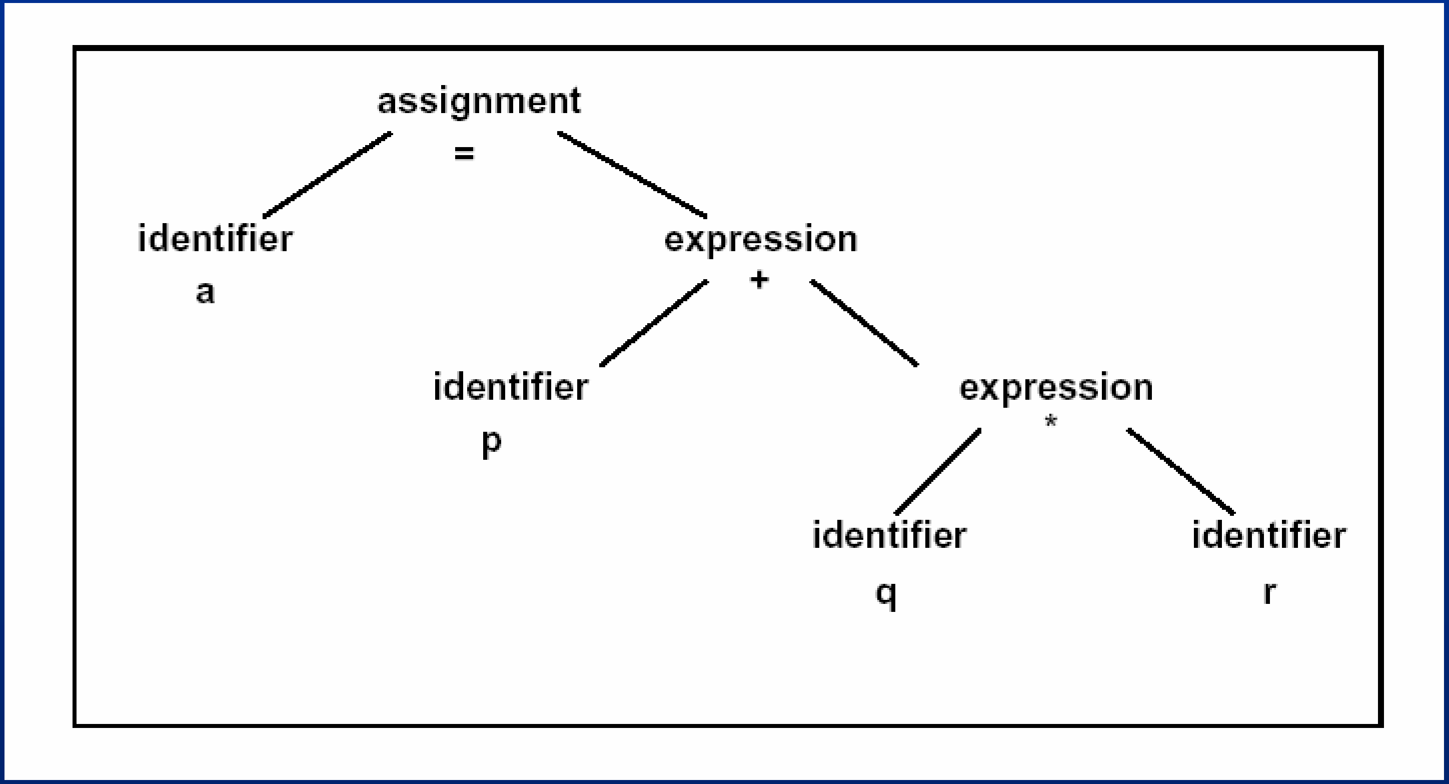


Compilation

- **Front-end**
  - Lexical and syntax analysis.
  - Parse-tree generation.
  - Macro-expansion.
  - Expansion of meta-variables.
- **Semantic analysis**
  - Data-flow and control-flow analysis.
  - Type checking.
  - Resolve arithmetic and relational operators.

Parse Tree Example

- a = p + q * r



Behavioral-Level Optimization

- **Semantic-preserving transformations aiming at simplifying the model.**
- **Applied to parse-trees or during their generation.**
- **Taxonomy**
  - Data-flow based transformations.
  - Control-flow based transformations.

Data-Flow Based Transformations

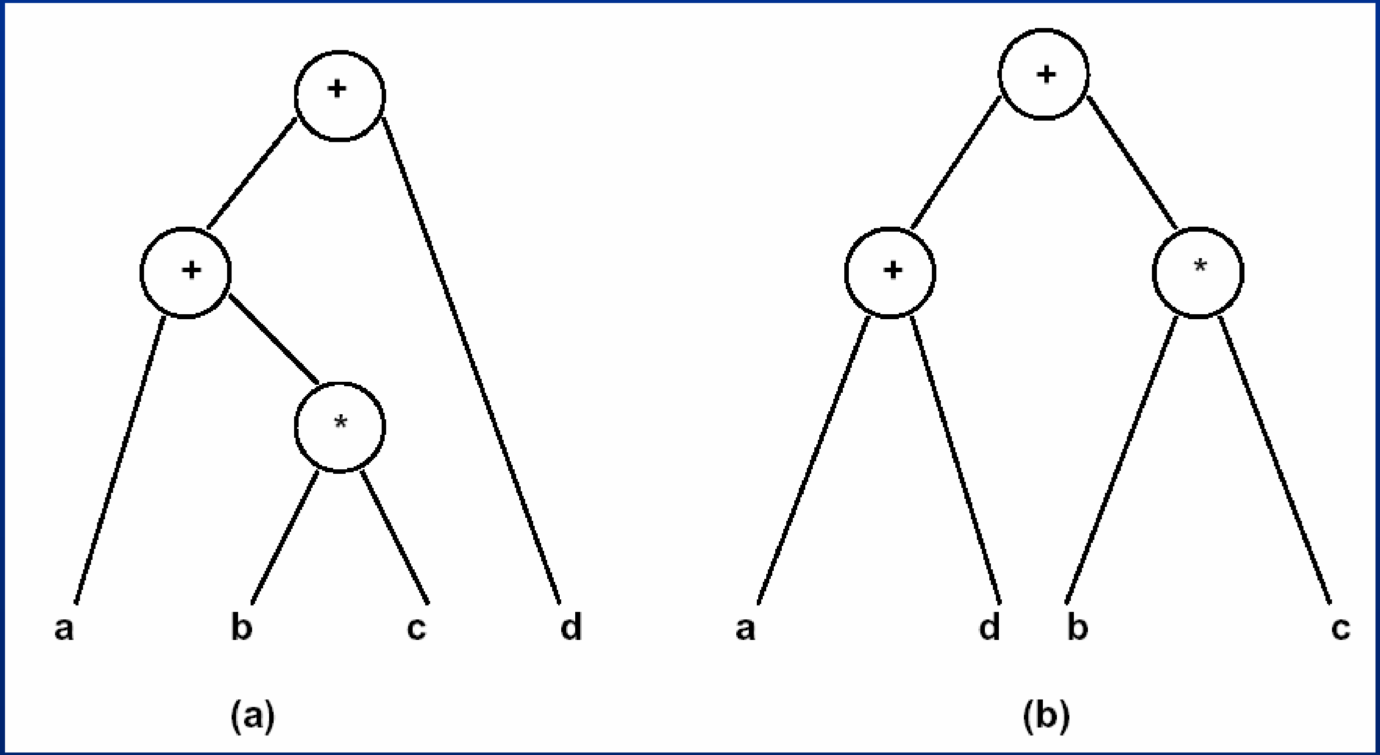
- Tree-height reduction.
- Constant and variable propagation.
- Common subexpression elimination.
- Dead-code elimination.
- Operator-strength reduction.
- Code motion.

Tree-Height Reduction

- **Applied to arithmetic expressions.**
- **Goal**
  - Split into two-operand expressions to exploit hardware parallelism at best.
- **Techniques**
  - Balance the expression tree.
  - Exploit commutativity, associativity and distributivity.

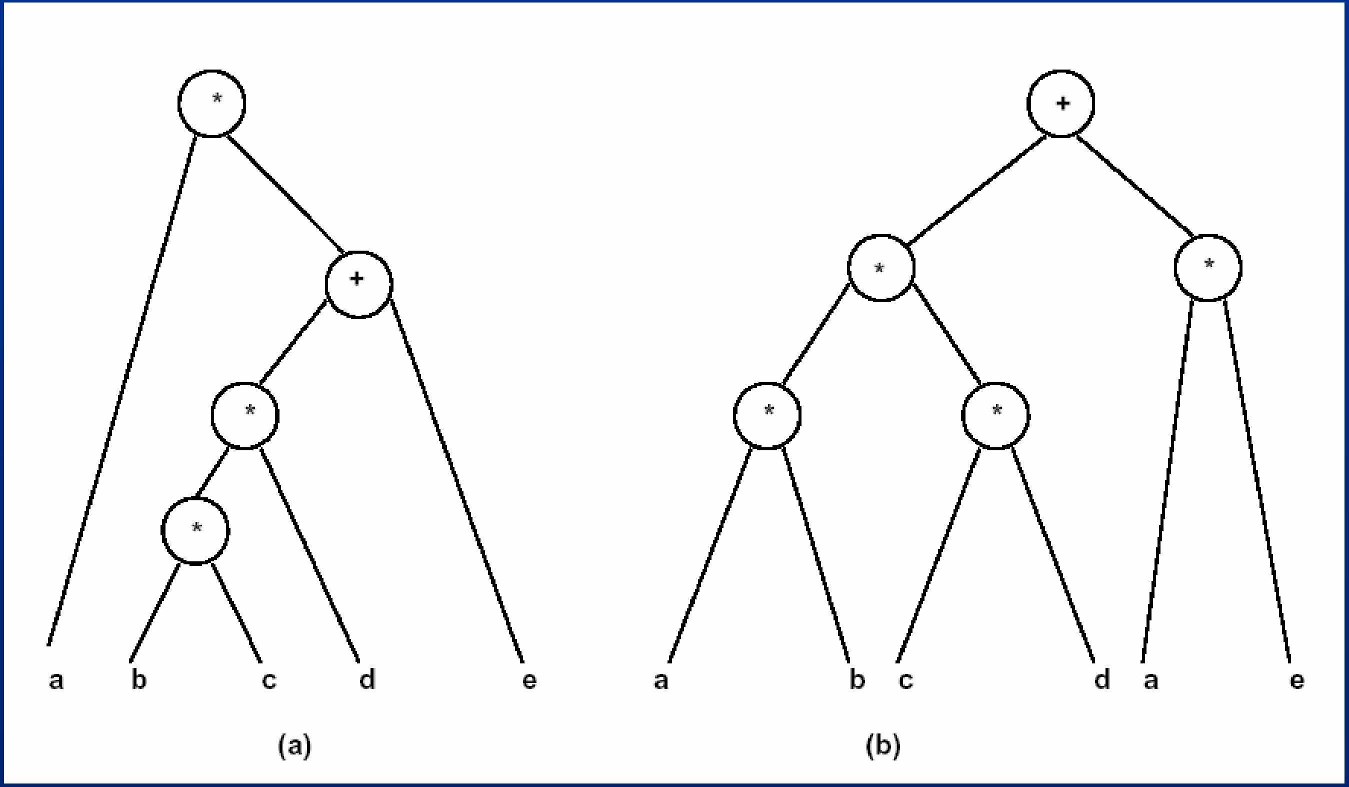
Example of Tree-Height Reduction using Commutativity and Associativity

- x = a + b * c + d => x = (a + d) + b * c



Example of Tree-Height Reduction using Distributivity

- x = a * (b * c * d + e) => x = a * b * c * d + a * e



Examples of Propagation & Subexpression Elimination

- **Constant propagation**
  - a = 0; b = a+1; c = 2 * b;
  - a = 0; b = 1; c = 2;
- **Variable propagation**
  - a = x; b = a+1; c = 2 * a;
  - a = x; b = x+1; c = 2 * x;
- **Subexpression elimination**
  - Search isomorphic patterns in the parse trees.
  - Example
    - a = x+y; b = a+1; c = x+y;
    - a = x+y; b = a+1; c = a;

Examples of Other Transformations

- **Dead-code elimination**
  - a = x; b = x+1; c = 2 * x;
  - a = x; can be removed if not referenced.
- **Operator-strength reduction**
  - a = x^2; b = 3 * x;
  - a = x * x; t = x << 1; b = x+t.
- **Code motion**
  - for (i = 1; i < a * b) { } ;
  - t = a * b; for (i = 1; i < t) { }.

Control-Flow Based Transformations

- **Model expansion.**
- **Conditional expansion.**
- **Loop expansion.**
- **Block-level transformations.**
- **Model Expansion**
  - Expand subroutine -- flatten hierarchy.
  - Useful to expand scope of other optimization techniques.
  - Problematic when routine is called more than once.
  - Example
    - x = a+b; y = a * b; z = foo(x; y);
    - foo(p; q){ t = q - p; return(t); }
    - By expanding foo
      - x = a+b; y = a * b; z = y - x

Conditional Expansion

- **Transform conditional into parallel execution with test at the end.**
- **Useful when test depends on late signals.**
- **May preclude hardware sharing.**
- **Always useful for logic expressions.**
- **Example**
  - If (A>B) { Y= A-B} Else {Y=B-A}.
- **Example**
  - y = ab; if (a) {x = b + d; } else {x = bd; }
    - can be expanded to: x = a (b+d) + a'bd
    - and simplified as: y = ab; x = y + d (a+b)

Loop Expansion

- **Applicable to loops with data-independent exit conditions.**
- **Useful to expand scope of other optimization techniques.**
- **Problematic when loop has many iterations.**
- **Example**
  - x = 0; for (i = 1; i ≤ 3; i++) {x = x+1; }
  - Expanded to:
    - x = 0; x = x+1; x = x+2; x = x+3

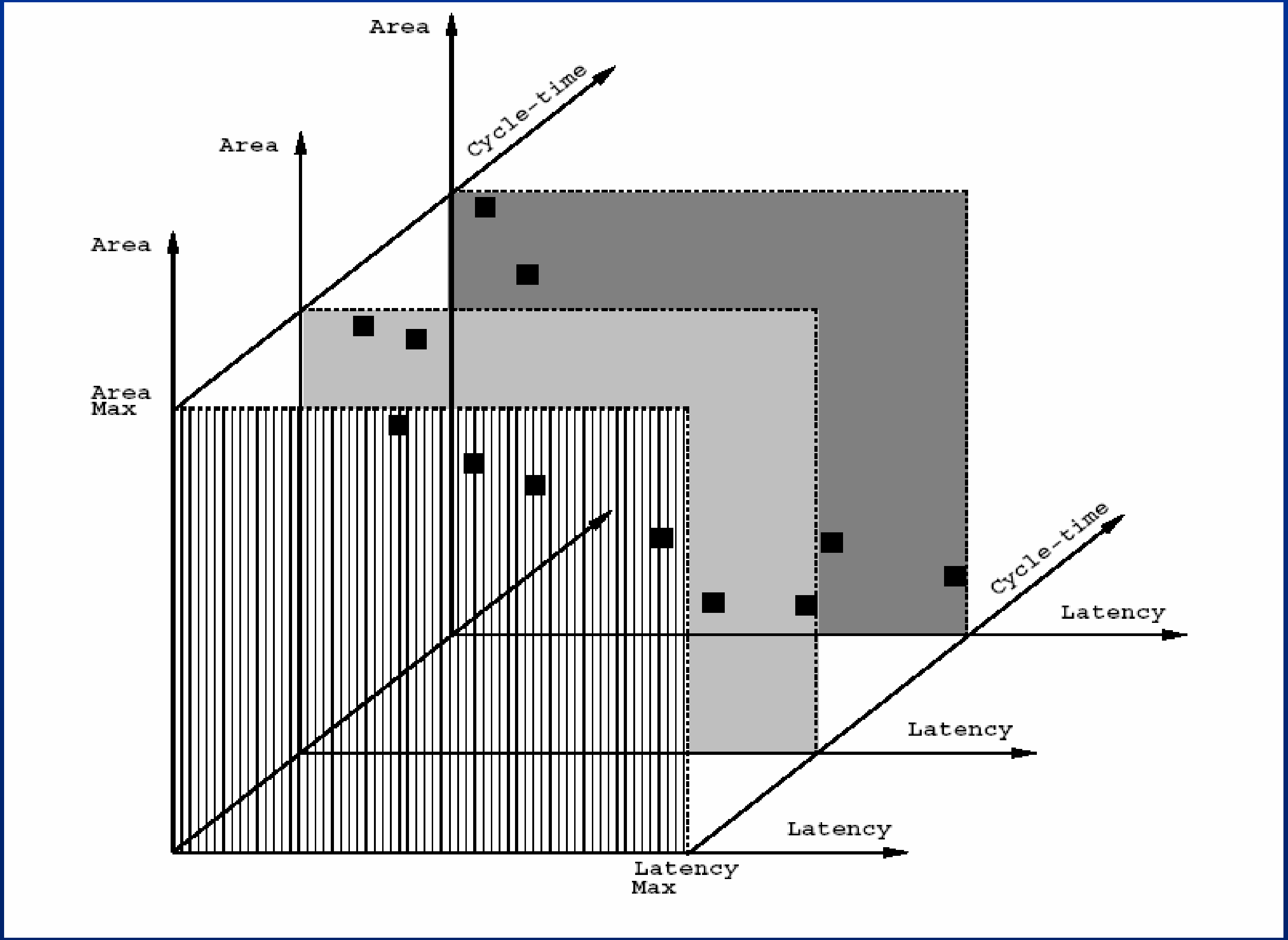
Architectural Synthesis and Optimization

- **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.**

Design Space and Objectives

- **Design space**
  - Set of all feasible implementations.
- **Implementation parameters**
  - Area.
  - Performance
    - Cycle-time,
    - Latency,
    - Throughput (for pipelined implementations).
  - Power consumption.

Design Evaluation Space



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.

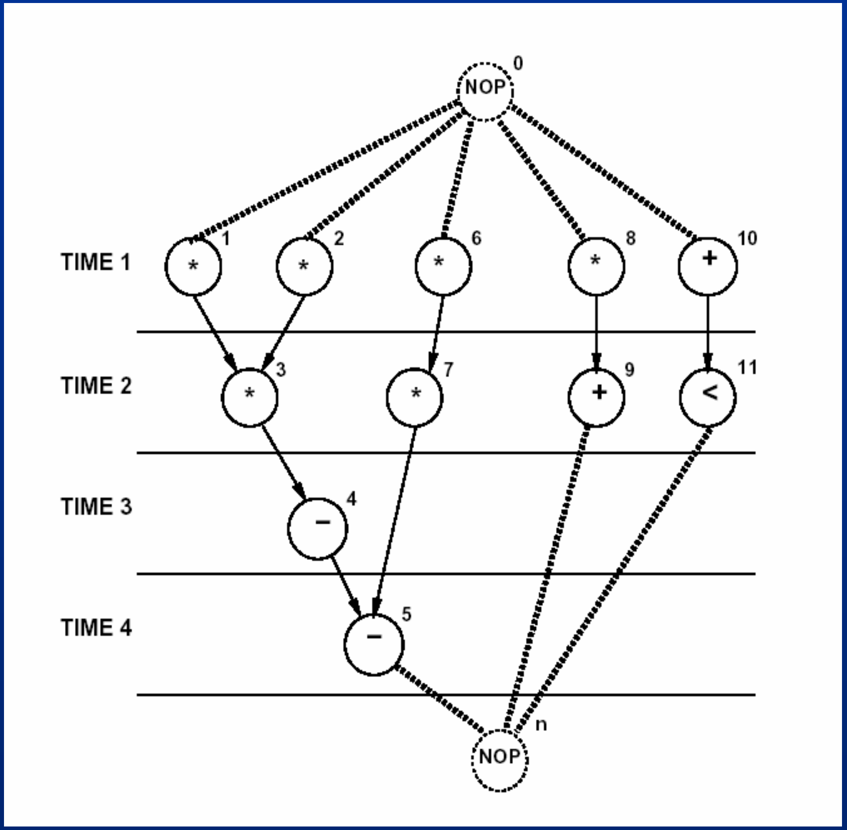
Resources and Circuit Families

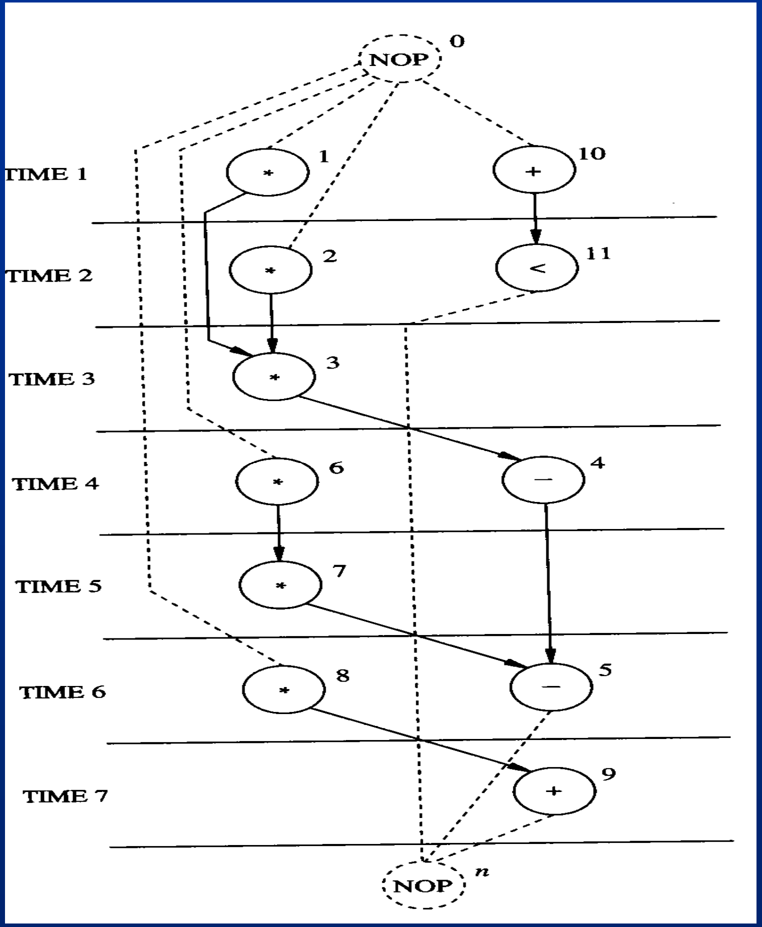
- **Resource-dominated circuits.**
  - Area and performance depend on few, well-characterized blocks.
  - Example: DSP circuits.
- **Non resource-dominated circuits.**
  - Area and performance are strongly influenced by sparse logic, control and wiring.
  - Example: some ASIC circuits.

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**
  - Latency.
  - Detailed timing constraints.
- **Scheduling with resource constraints.**

Scheduling ...

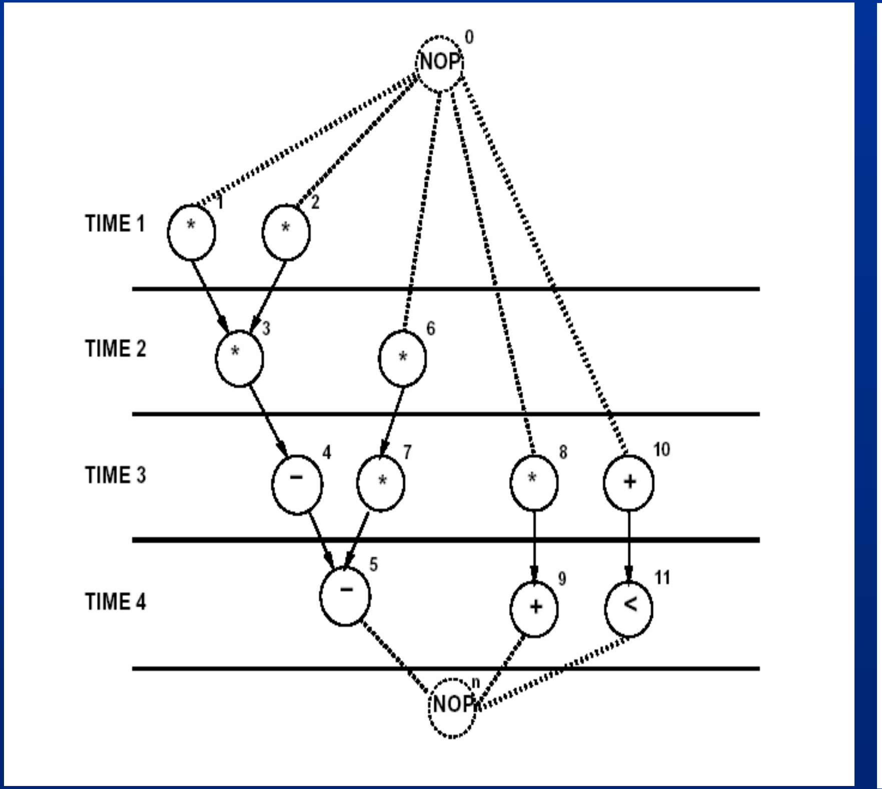




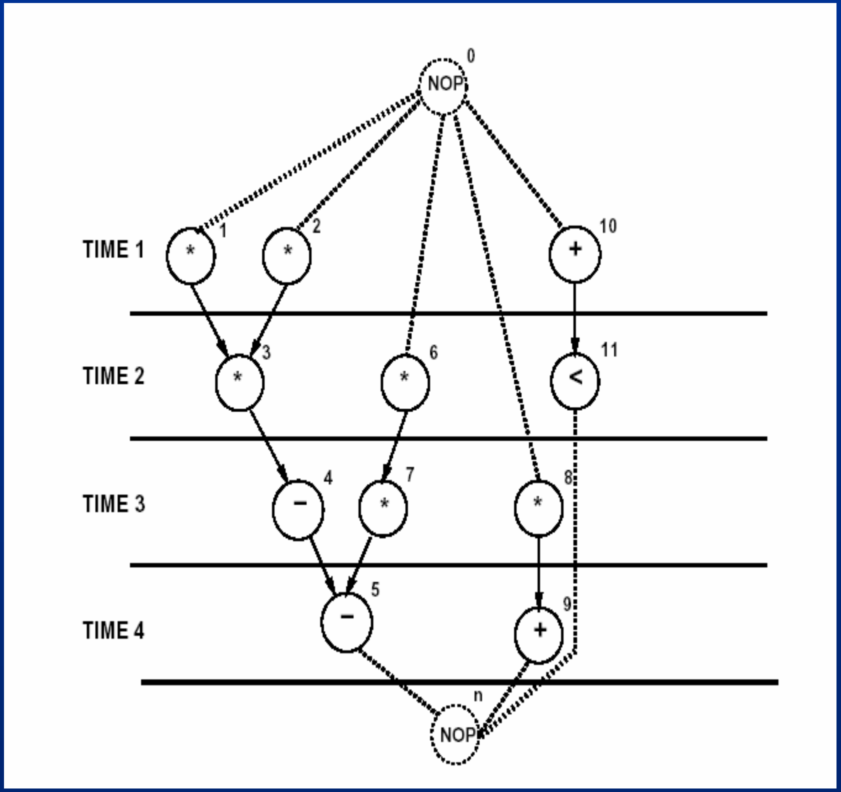
4 Multipliers, 2 ALUs

1 Multiplier , 1 ALU

… Scheduling



2 Multipliers, 3 ALUs

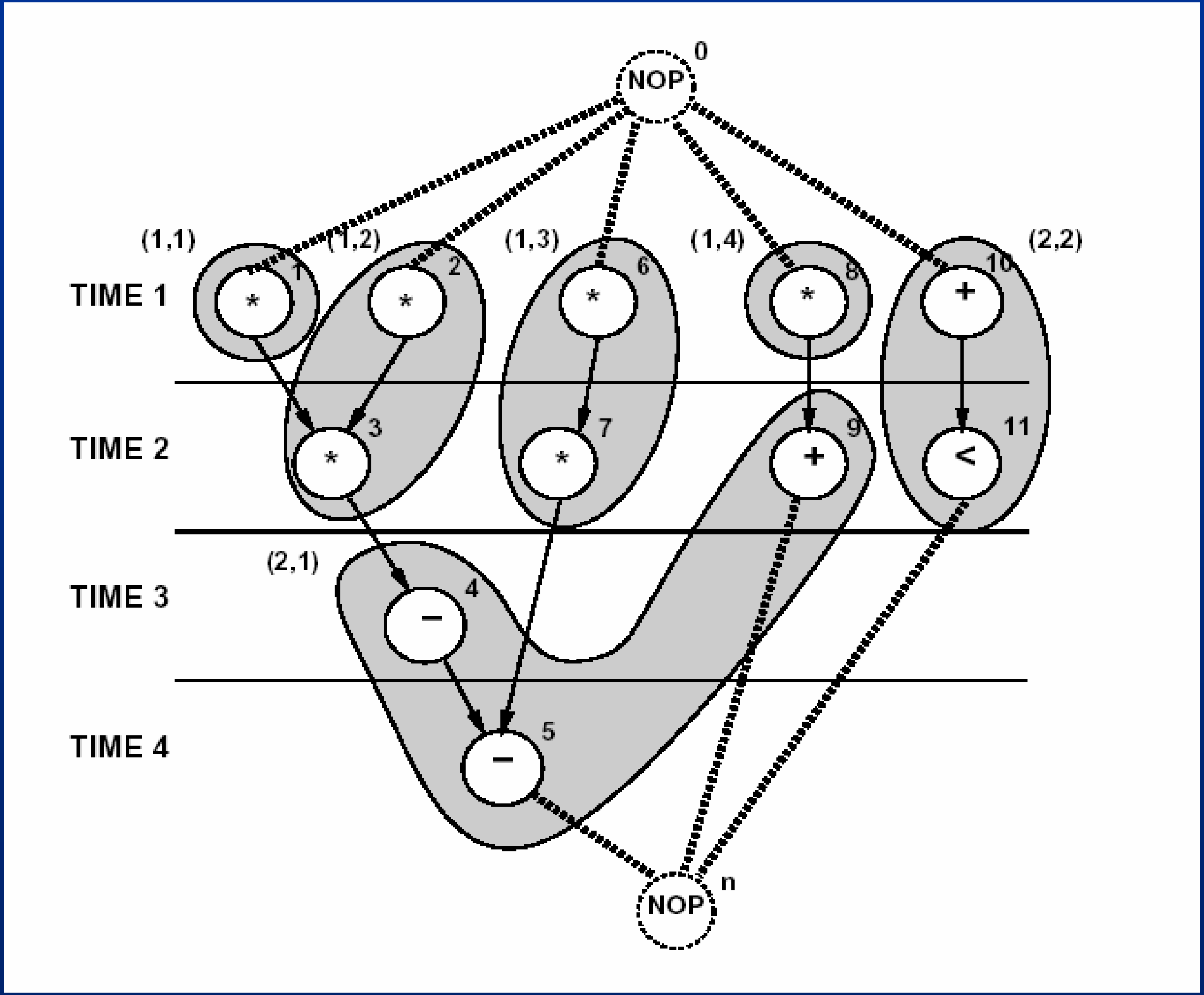


2 Multipliers, 2 ALUs

Synthesis in the Spatial Domain: Binding

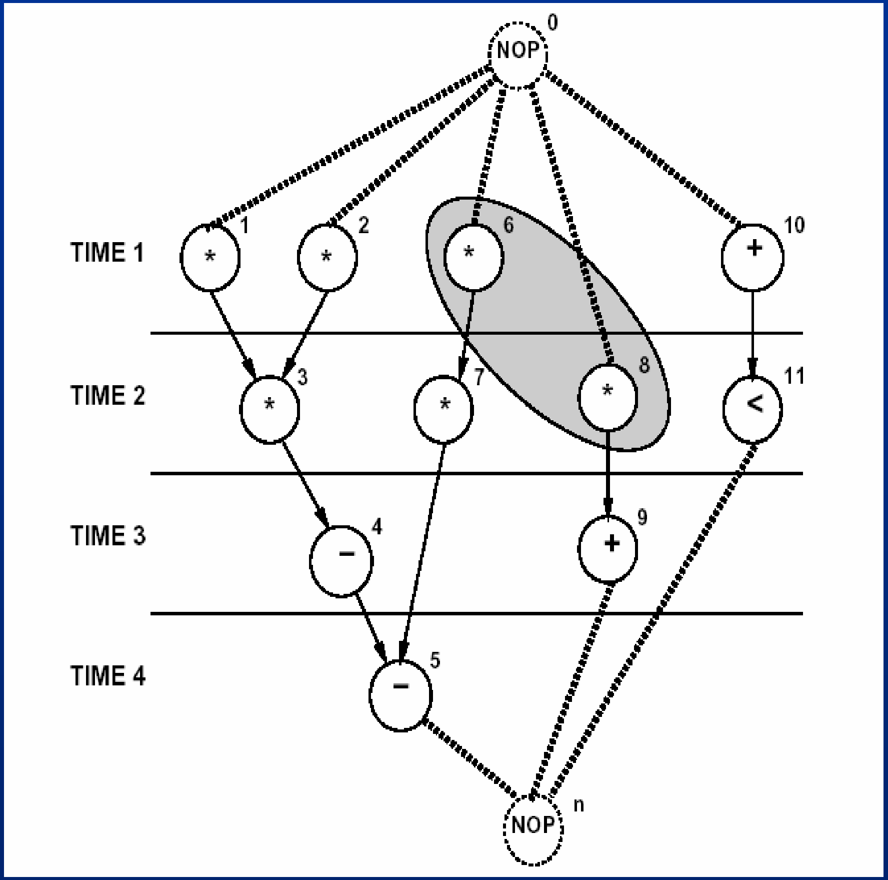
- **Binding**
  - Associate a resource with each operation with the same type.
  - Determine area of the implementation.
- **Sharing**
  - Bind a resource to more than one operation.
  - Operations must not execute concurrently.
- **Bound sequencing graph**
  - Sequencing graph with resource annotation.

Example: Bound Sequencing Graph



Binding Specification

- Mapping from the vertex set to the set of resource instances, for each given type.
- Partial binding
  - Partial mapping, given as design constraint.
- Compatible binding
  - Binding satisfying the constraints of the partial binding.



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.

Approaches to Architectural Optimization

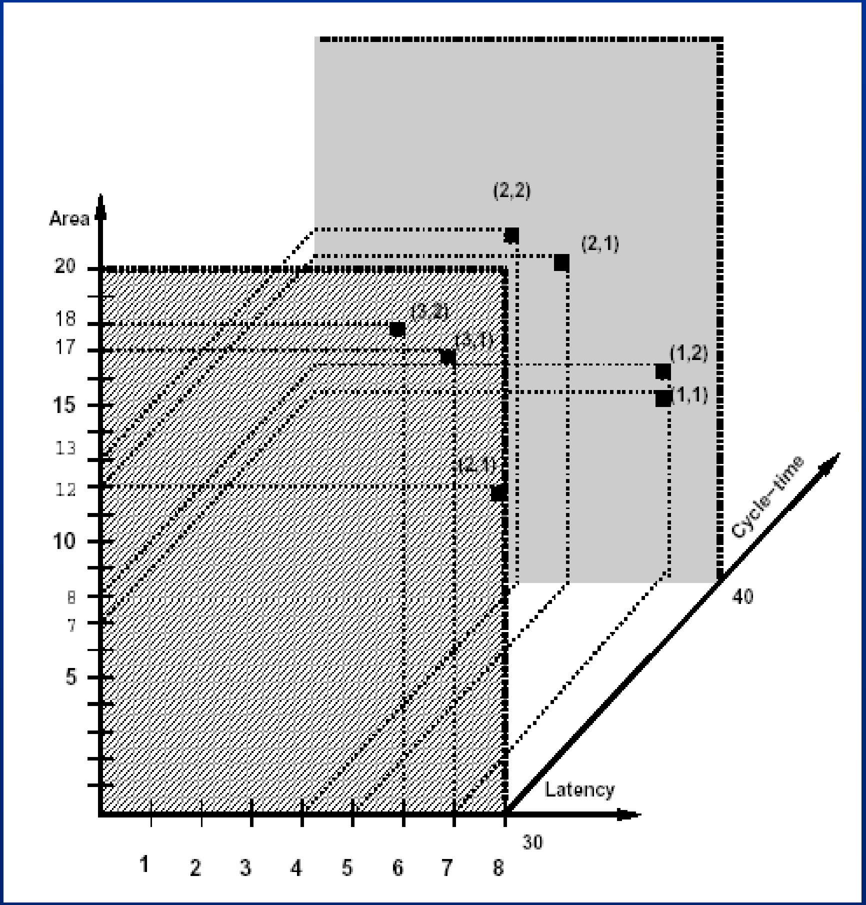
- **Multiple-criteria optimization problem**
  - area, latency, cycle-time.
- **Determine Pareto optimal points**
  - Implementations such that no other has all parameters with inferior values.
- **Draw trade-off curves**
  - discontinuous and highly nonlinear.
- Area/latency trade-off
  - **for some values of the cycle-time.**
- Cycle-time/latency trade-off
  - **for some binding (area).**
- Area/cycle-time trade-off
  - **for some schedules (latency).**

Area/Latency Trade-off …

- **Rationale**
  - Cycle-time dictated by system constraints.
- **Resource-dominated circuits**
  - Area is determined by resource usage.
- **General circuits**
  - Area and delay affected by registers, steering logic, wiring and control logic.
  - Complex dependency of area and delay on circuit structure.
- **Scheduling and binding are deeply interrelated.**
  - Most approaches perform scheduling before binding (fits well for CPU and DSP designs).
  - Performing binding before scheduling fits control dominated designs.
- **Approaches**
  - Schedule for minimum latency under resource constraints.
  - Schedule for minimum resource usage under latency constraints.

… Area/Latency Trade-off

- Areas smaller than 20 units.
- Latency less than 8 cycles.
- ALU area = 1 unit.
- MUL area = 5 units.
- Overhead area = 1 unit.
- ALU propagation delay 25ns.
- MUL propagation delay 35 ns.
- Cycle time = 40 ns
  - Resources have unit execution delay.
- Cycle time = 30ns
  - MUL has 2 unit execution delay.


Area
(2,2)
(2,1)
(3,2)
(3,1)
(1,2)
(1,1)
(2,1)
Cycle-time
Latency