ECE 667

Spring 2013

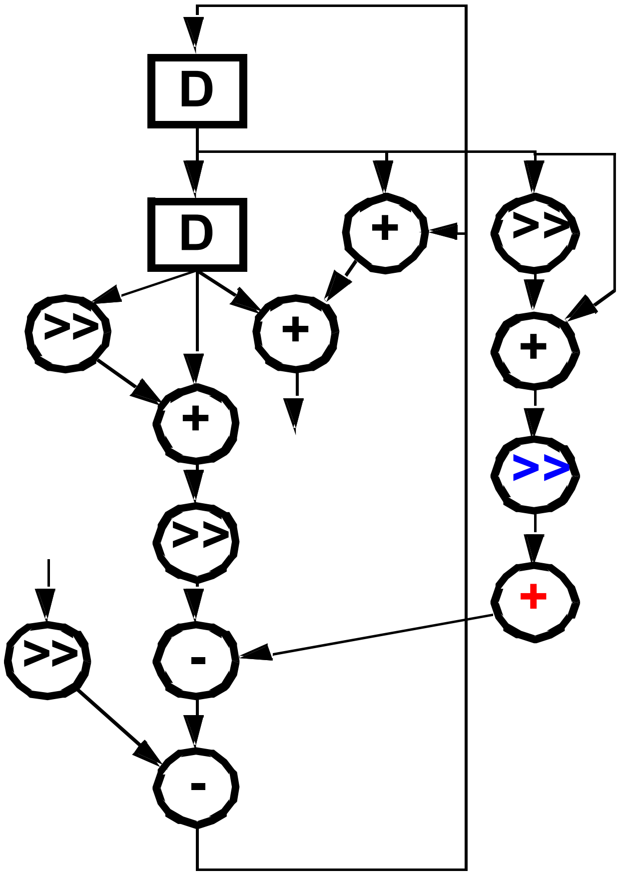
Synthesis and Verification of Digital Circuits

Scheduling Algorithms

Architectural Optimization

- Optimization in view of design space flexibility
- A multi-criteria optimization problem:
  - Determine schedule ϕ and binding β.
  - Under area A, latency L and cycle time τ objectives
- Solution space tradeoff curves:
  - Non-linear, discontinuous
  - Area / latency / cycle time (more?)
- Evaluate (estimate) cost functions
- Unconstrained optimization problems for resource dominated circuits:
  - Min area: solve for minimal binding
  - Min latency: solve for minimum L scheduling

Scheduling, Allocation and Assignment



Allocation: How Much?

2 adders
1 shifter
24 registers

Assignment: Where?

Shifter 1

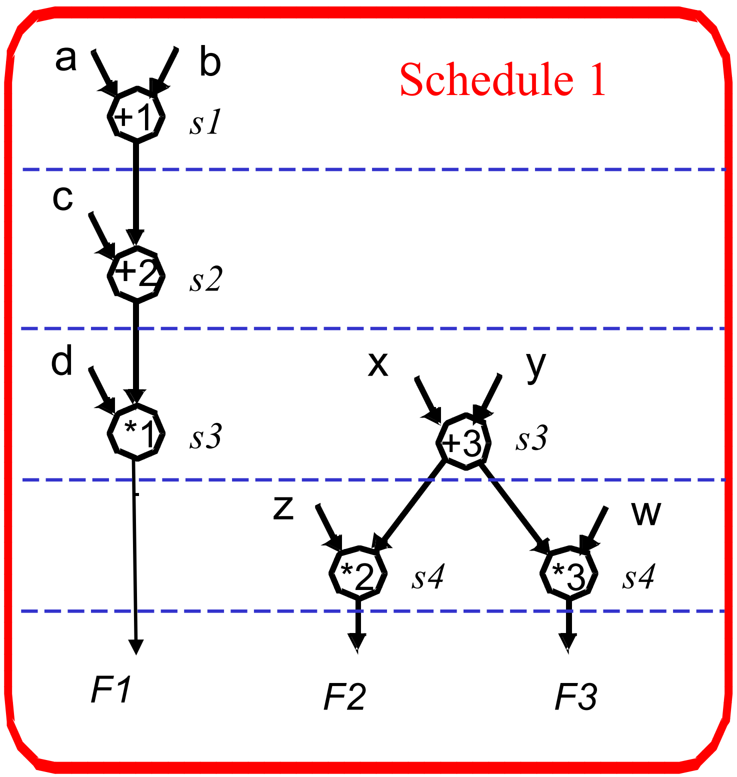
Schedule: When?

Time Slot 4

Techniques are well understood and mature

Scheduling and Assignment - Overview

*F1 = (a + b + c) * d   F2 = (x + y) * z   F3 = (x + y)* w*

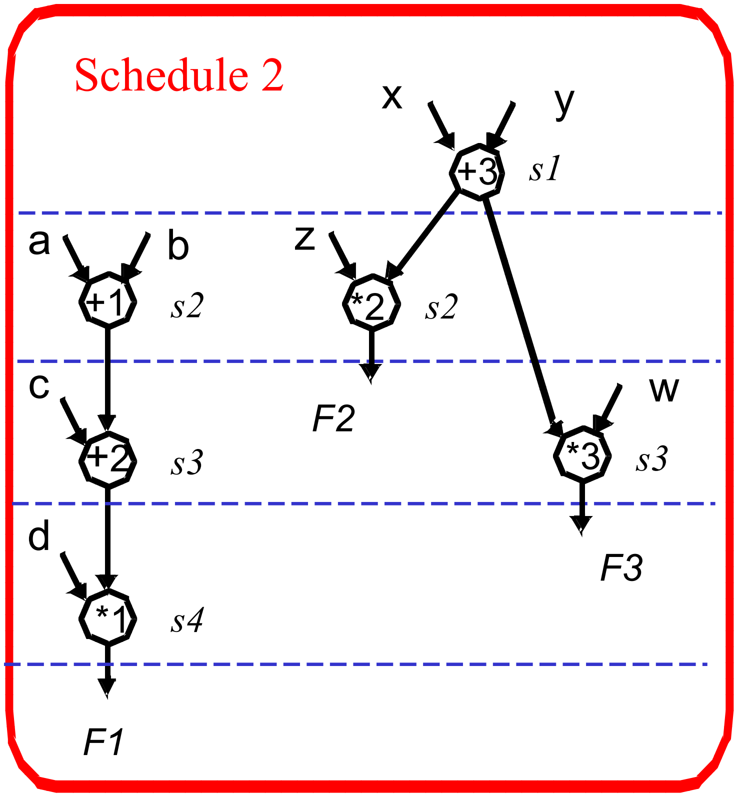


4 control steps, 1 Add, 2 Mult

Control Step	+	*	*
1	**+1**		
2	**+2**		
3	**+3**	***1**	
4		***2**	***3**

Scheduling and Assignment - Overview

*F1 = (a + b + c) * d   F2 = (x + y) * z   F3 = (x + y)* w*



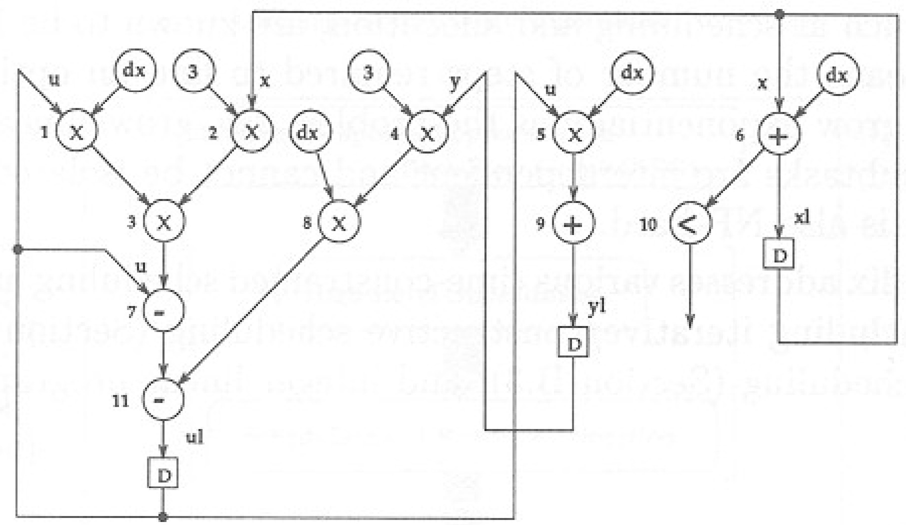
4 control steps, 1 Add, 1 Mult

Control Step	+	*
1	**+3**	
2	**+1**	***2**
3	**+2**	***3**
4		***1**

Algorithm Description → Data Flow Graph

$$y'' + 3xy' + 3y = 0 \qquad u = y' = \frac{dy}{dx}$$

$$\frac{du}{dx} = y'' = \frac{d^2y}{dx^2} = -3xy' - 3y = -3xu - 3y$$



while $(x < a)$ {

$\quad xl = x + dx;$

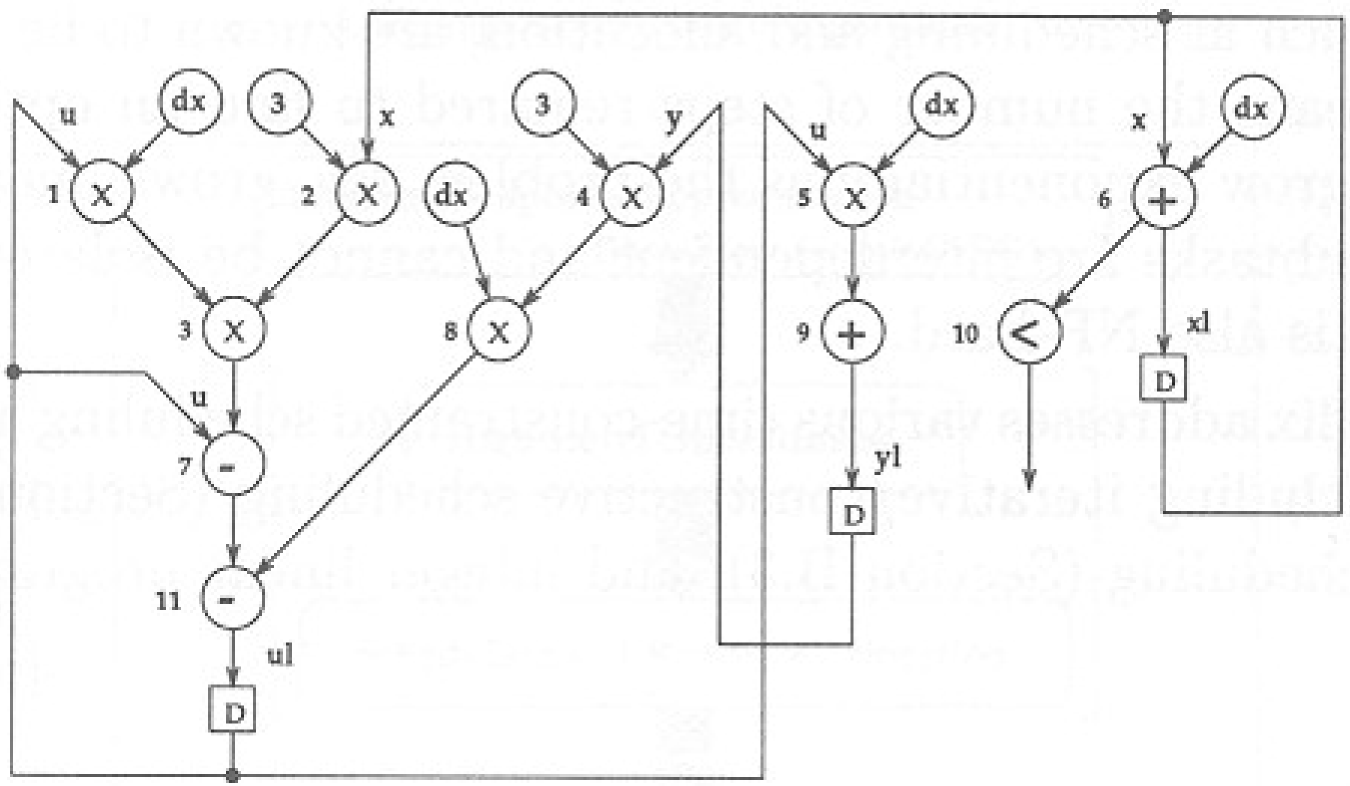
$\quad ul = u - (3 * x * u * dx) - (3 * y * dx);$

$\quad yl = y + u * dx;$

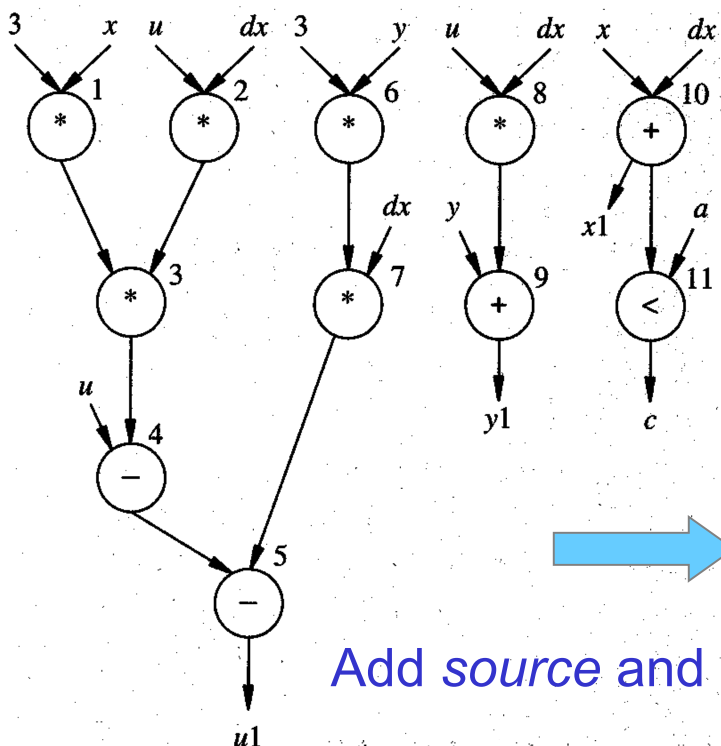
$\quad x = xl; \; y = yl; \; u = ul;$

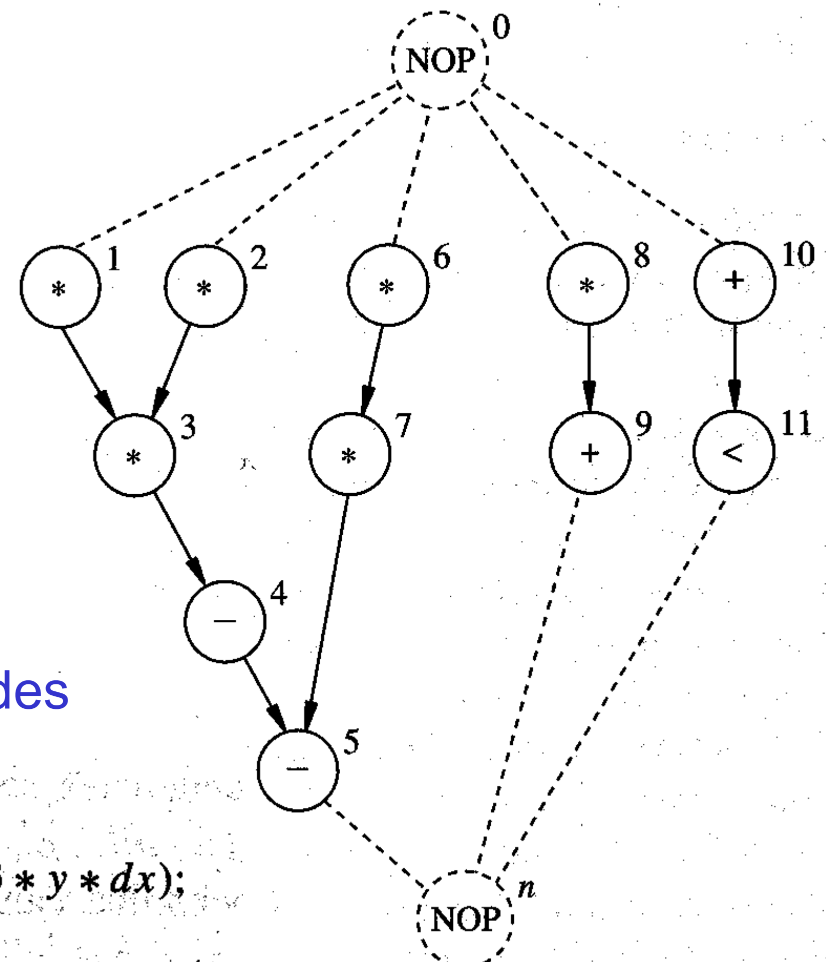
}

Data Flow Graph (DFG)



Sequencing Graph



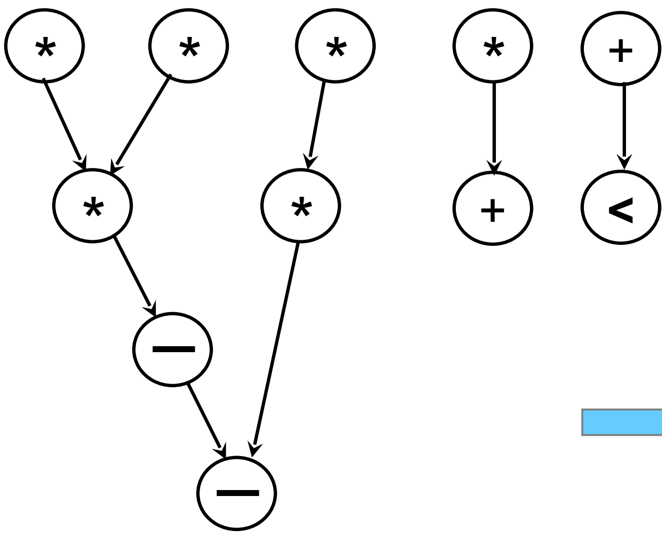


Add *source* and *sink* nodes

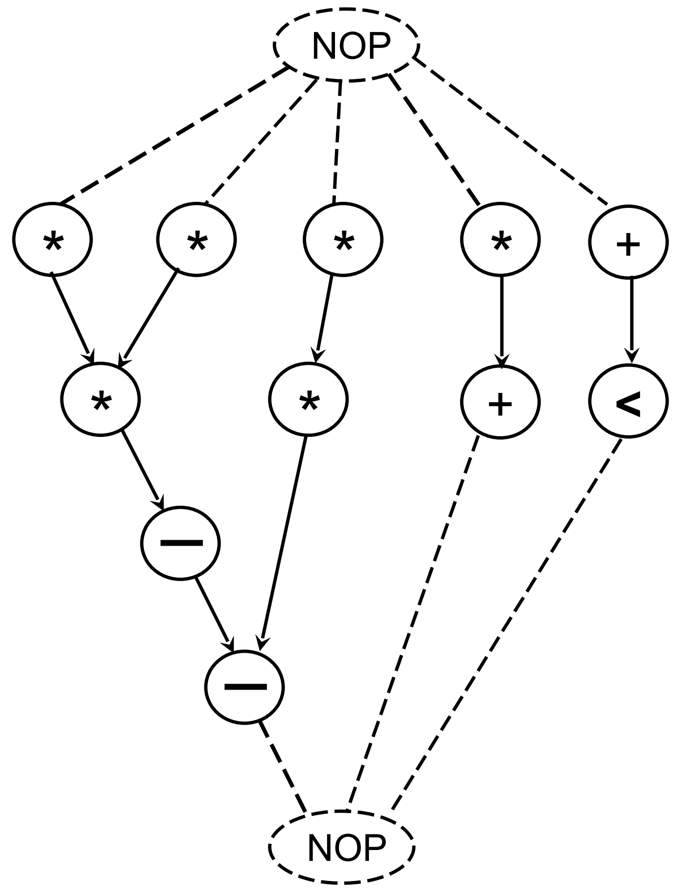
$$
\begin{aligned}
xl &= x + dx; \\
ul &= u - (3 * x * u * dx) - (3 * y * dx); \\
yl &= y + u * dx; \\
c &= xl < a;
\end{aligned}
$$

Sequencing Graph

- Add *source* and *sink* nodes (NOP) to the DFG



Data Flow Graph (DFG)



Sequencing Graph

ASAP Scheduling Algorithm

- As Soon as Possible scheduling
  - Unconstrained minimum latency scheduling
  - Uses topological sorting of the sequencing graph (polynomial time)
  - Gives *optimum* solution to scheduling problem
  - Schedule first the first node $n_o \rightarrow T1$ until last node n_v is scheduled
  - C_i = completion time (delay) of predecessor *i* of node *j*

Input: DFG $G = (N, E)$.
Output: ASAP Schedule.
1. $TS_0 = 1$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   - 2.1 Select a node n_j whose predecessors have already been scheduled;
   - 2.2 Schedule node n_j to time step $TS_j = \max\{TS_i + (C_i)\}$ $\forall\ n_i \rightarrow n_j$;

   }

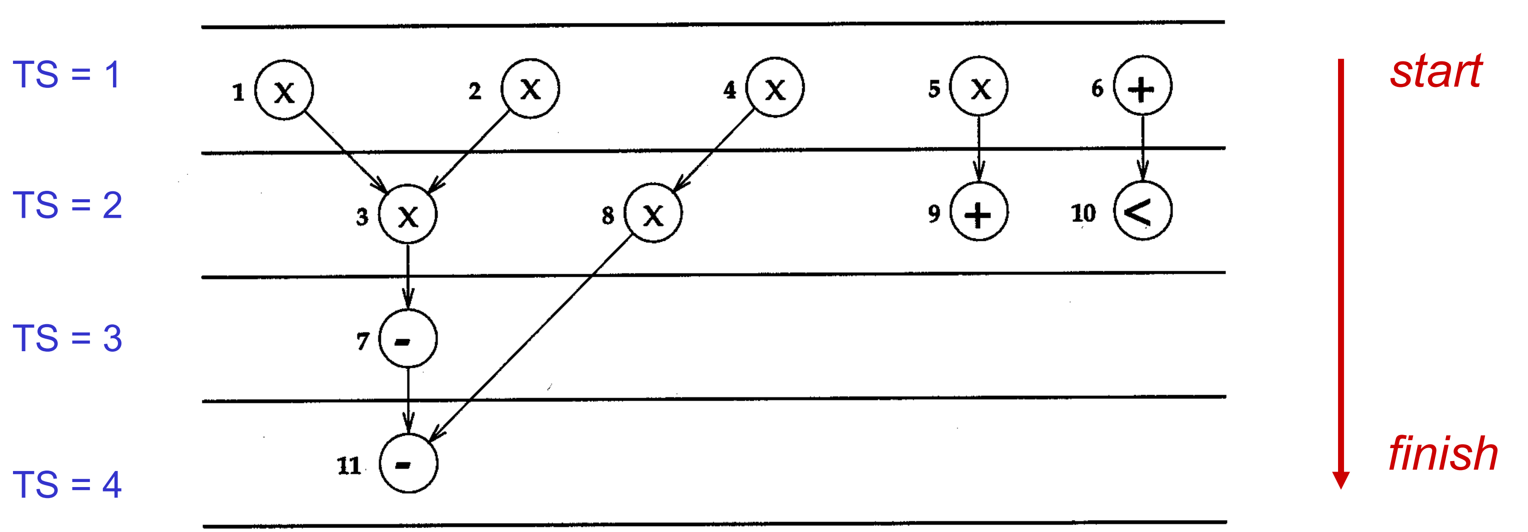
ASAP Scheduling Algorithm - Example

Input: DFG $G = (N, E)$.
Output: ASAP Schedule.

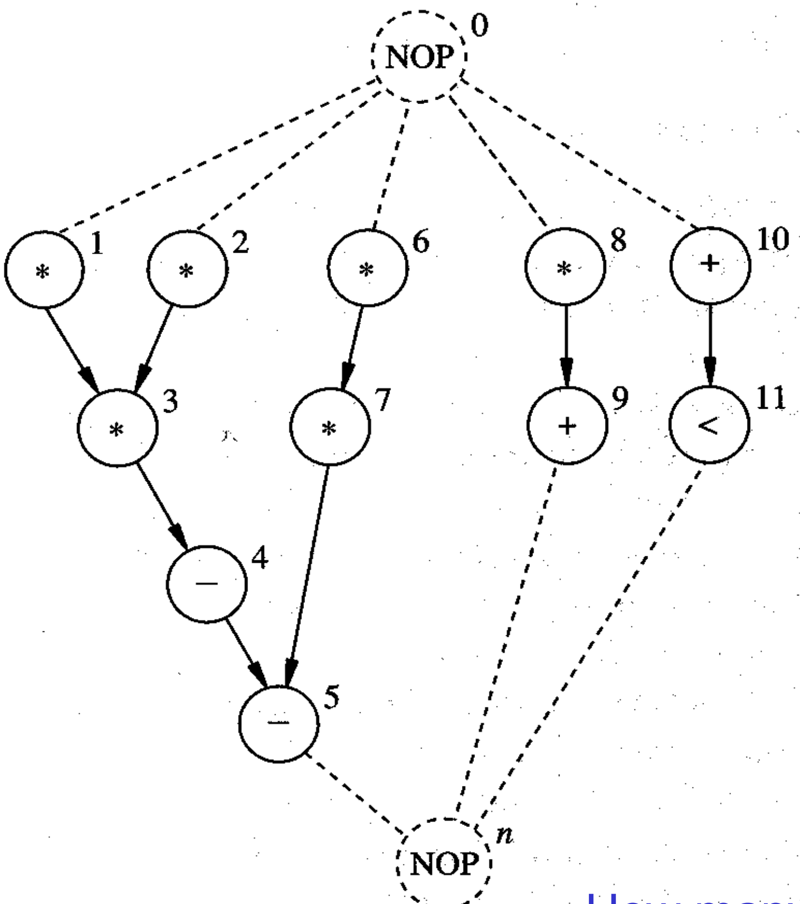
1. $TS_0 = 1$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   2.1 Select a node n_j whose predecessors have already been scheduled;
   2.2 Schedule node n_j to time step $TS_j = \max\{TS_i + (C_i)\}$ $\forall\ n_i \to n_j$;

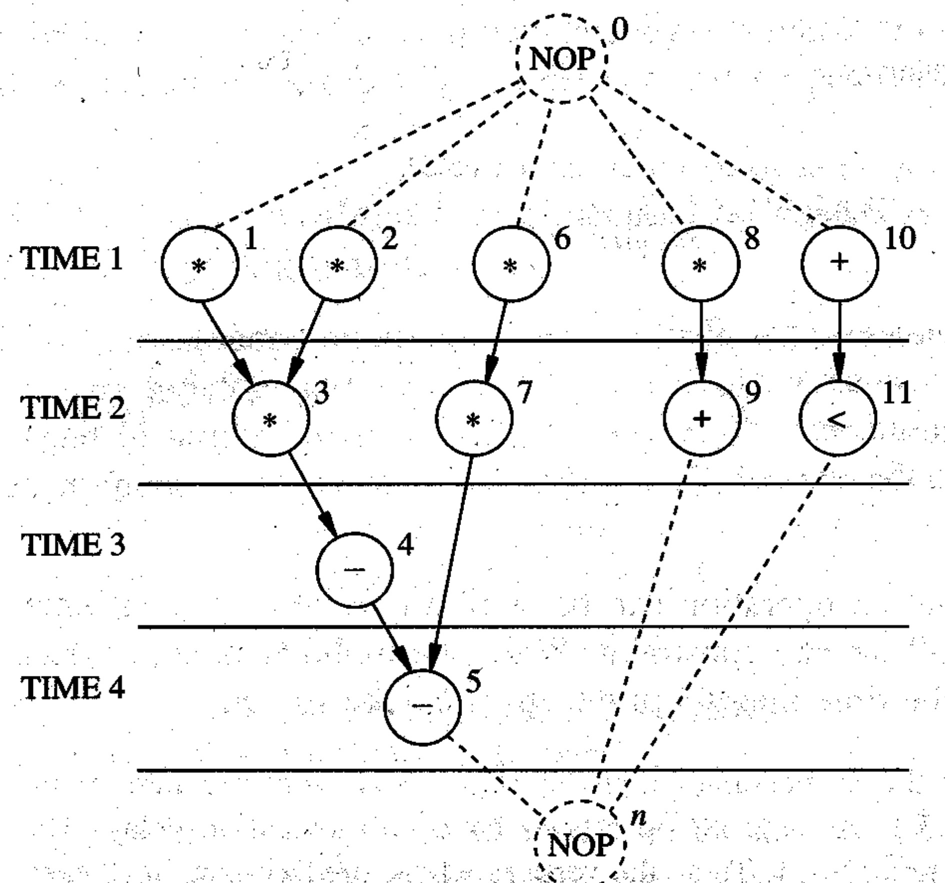
   }

Assume $C_i = 1$



ASAP Scheduling Example





How many Add, Mult are needed ?

Sequence Graph → ASAP Schedule

ALAP Scheduling Algorithm

- As late as Possible scheduling
  - Latency-constrained scheduling (latency is *fixed*)
  - Uses reversed topological sorting of the sequencing graph
  - If over-constrained (latency too small), solution *may not exist*
  - Schedule first the last node $n_v \rightarrow T$, until first node n_0 is scheduled
  - C_i = completion time (delay) of predecessor *i* of node *j*

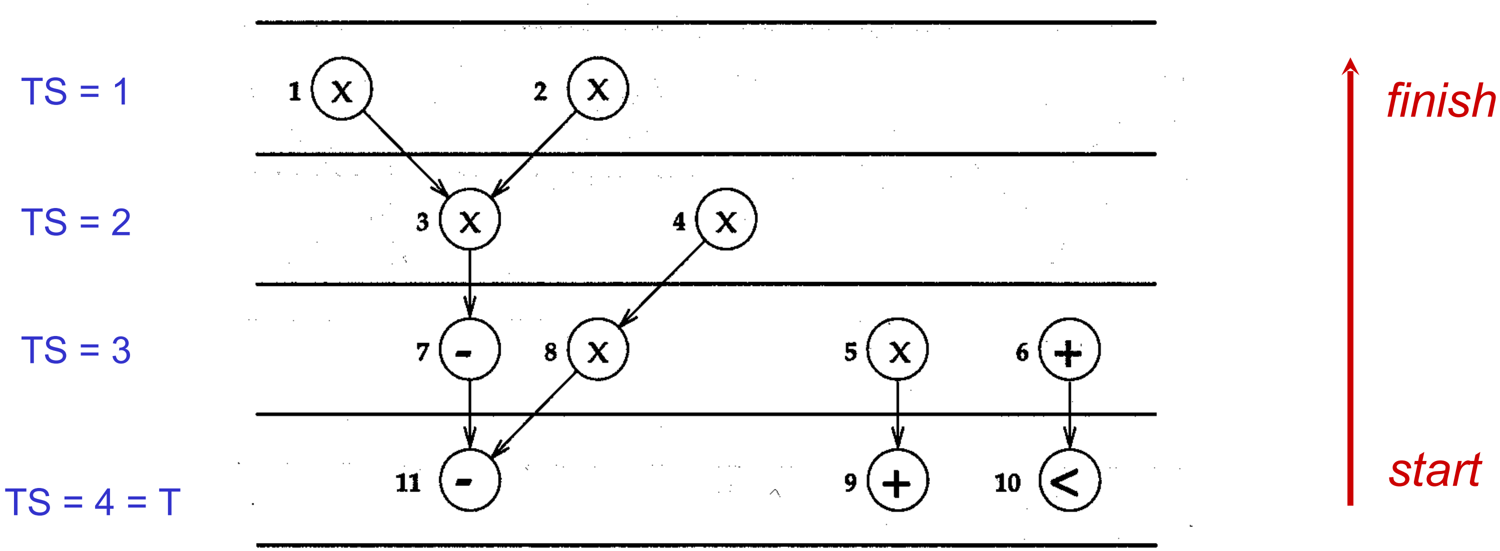
Input: DFG $G = (N, E)$, $IterationPeriod = T$. *(Latency)*
Output: ALAP Schedule.
1. $TS_0 = T$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
    2.1 Select a node n_i whose successors have already been scheduled;
    2.2 Schedule node n_i to time step $TS_i = \min\{TS_j - (C_i)\}$ $\forall\ n_i \rightarrow n_j$;

    }

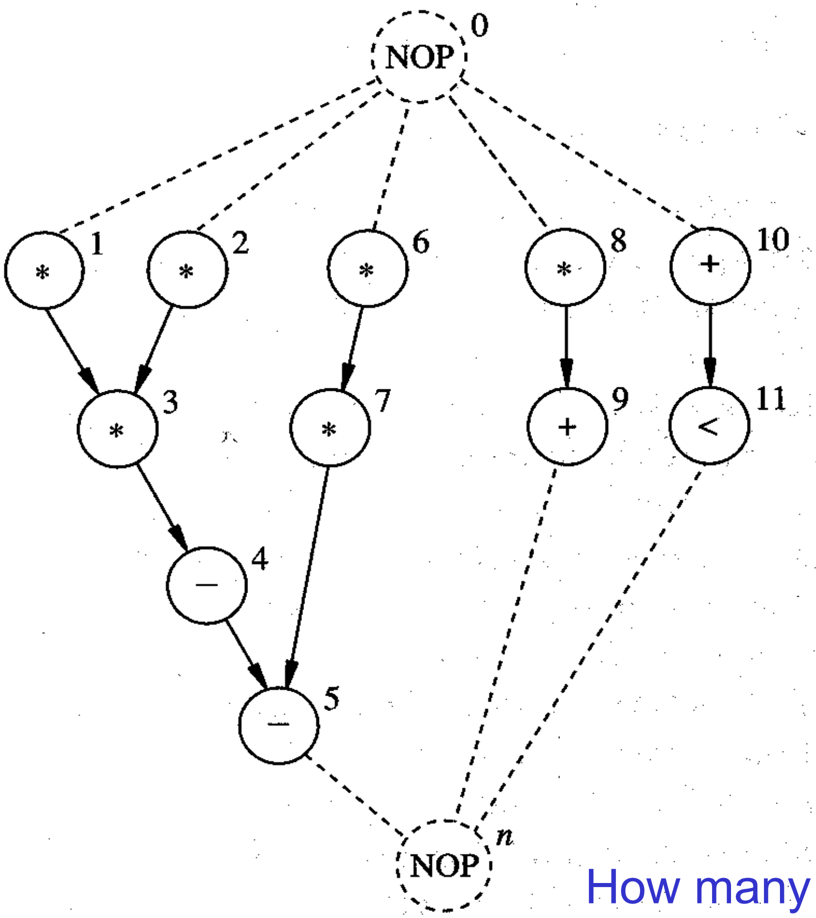
ALAP Scheduling Algorithm - example

Input: DFG $G = (N, E)$, $IterationPeriod = T$.
Output: ALAP Schedule.
1. $TS_0 = T$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   2.1 Select a node n_i whose successors have already been scheduled;
   2.2 Schedule node n_i to time step $TS_i = \min\{TS_j - (C_i)\}$ $\forall\ n_i \rightarrow n_j$;
   }

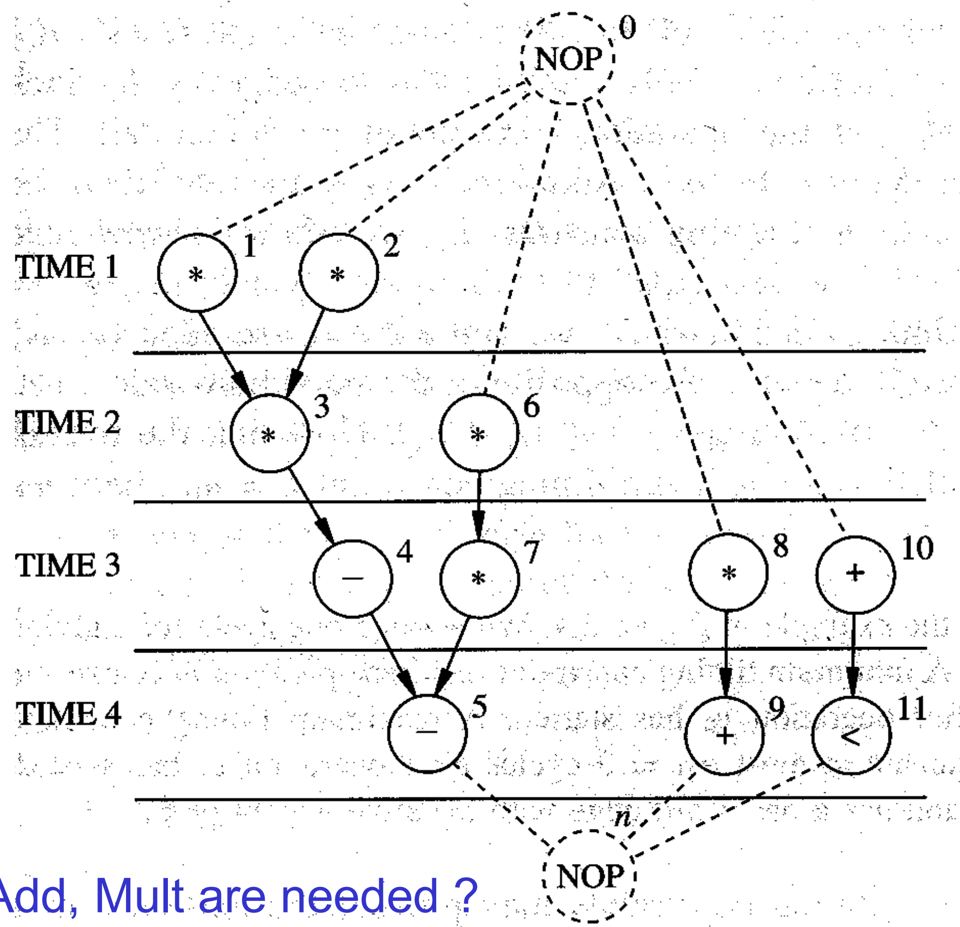
Assume $C_i = 1$



ALAP Scheduling Example



Sequence Graph

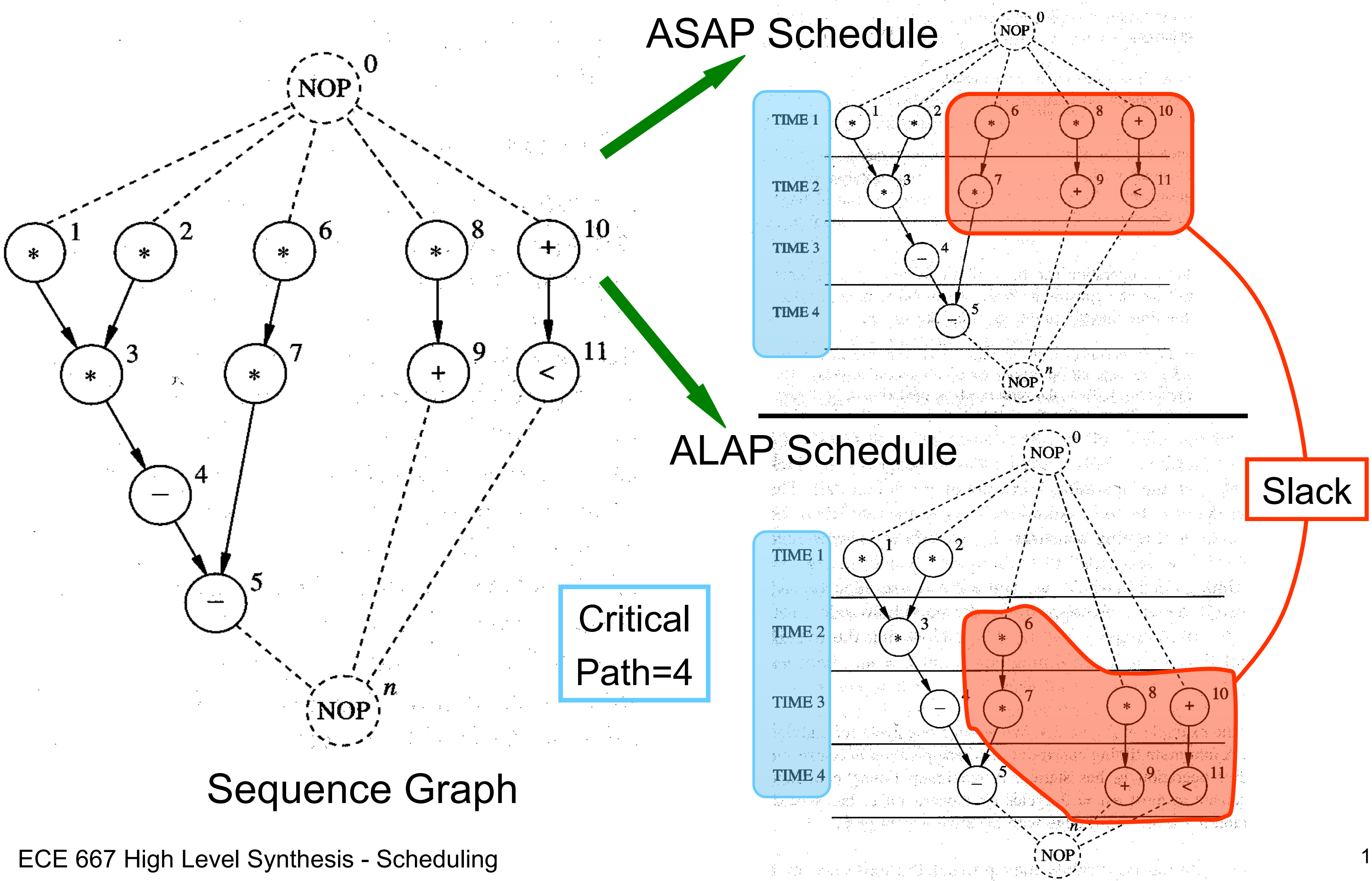


How many Add, Mult are needed ?

ALAP Schedule
(latency constraint = 4)

ASAP & ALAP Scheduling

No Resource Constraint

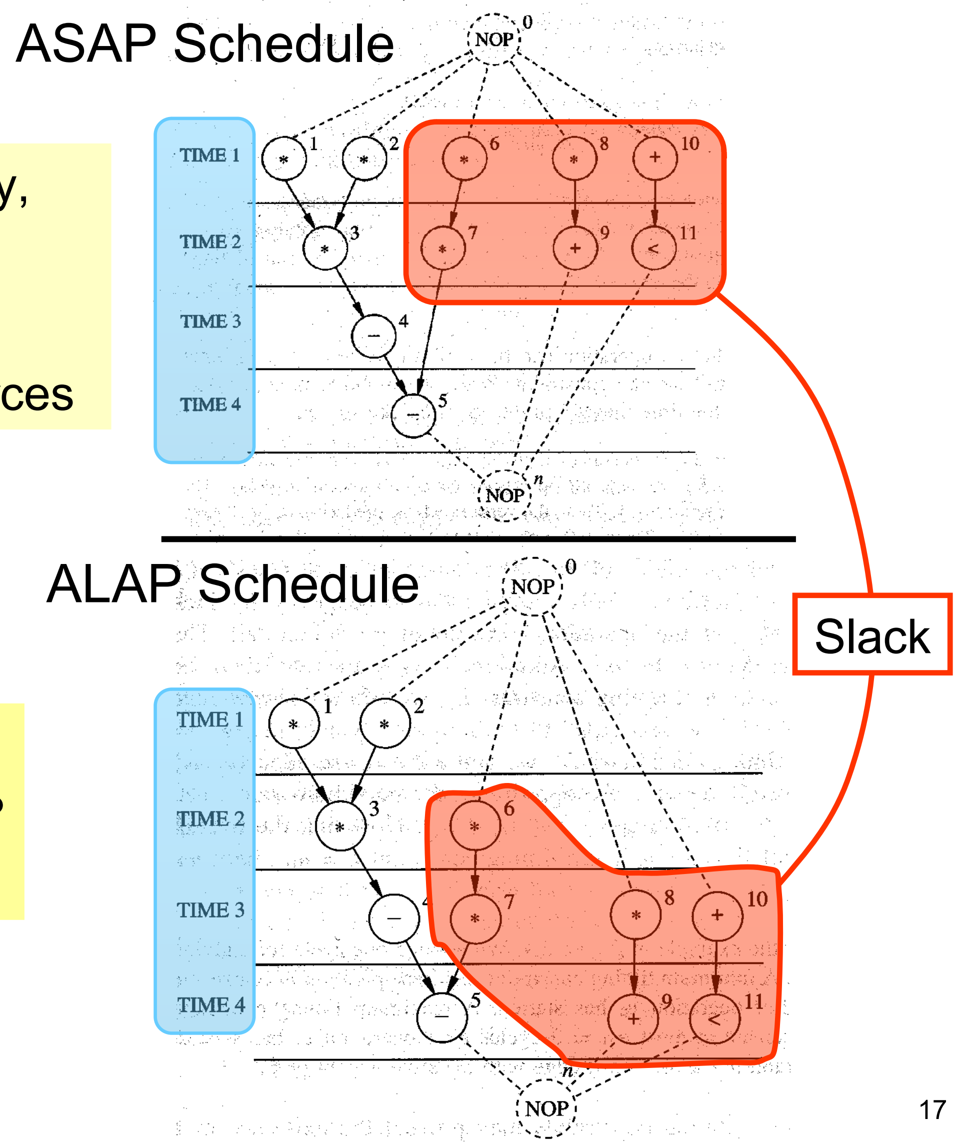


ASAP & ALAP Scheduling

No Resource Constraint

Having determined the minimum latency, what can we do with the slack?

- Adds flexibility to the schedule
- Determine the minimum # resources

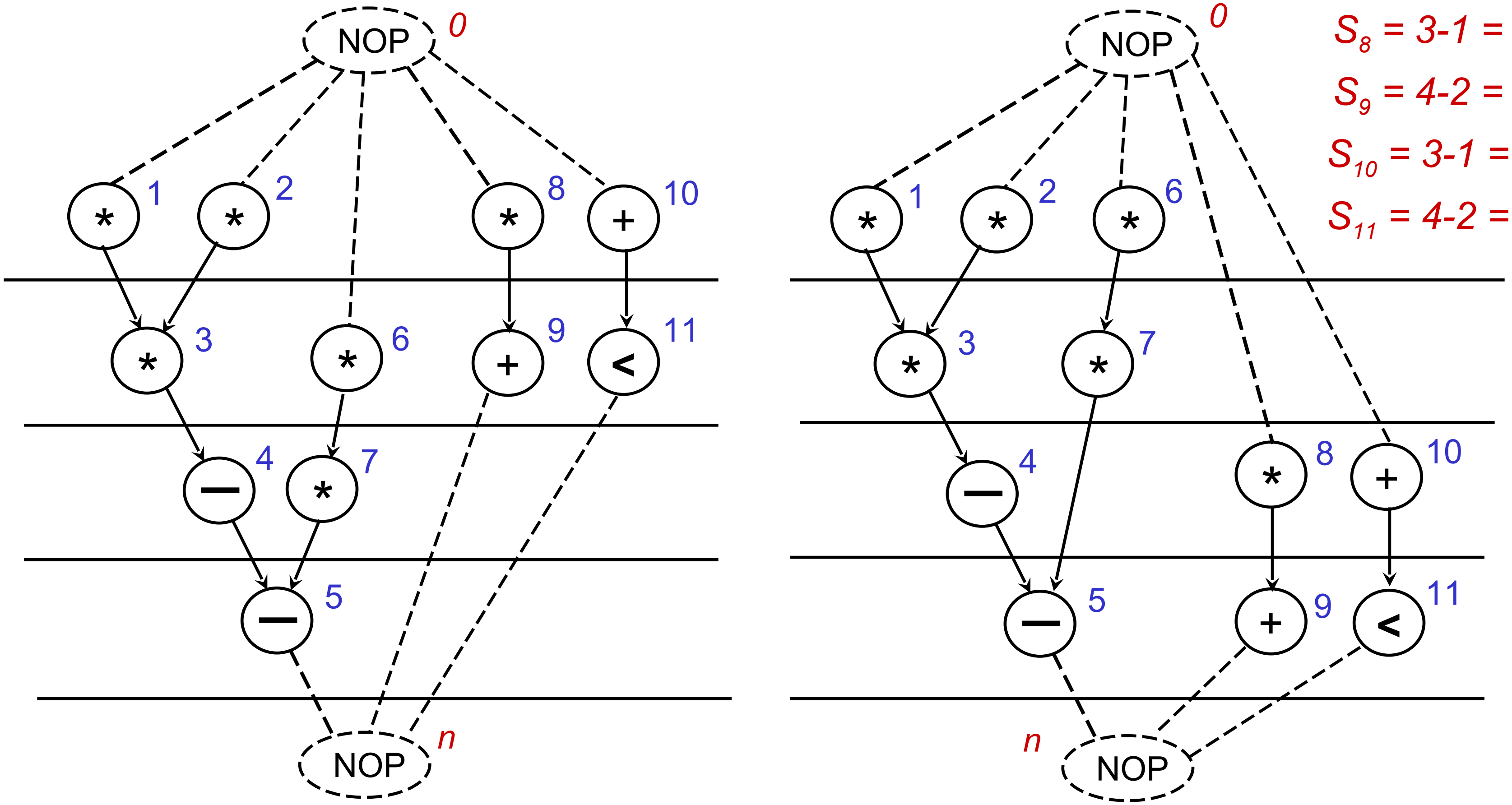


What if you want to find

- Min. latency given fixed resources?
- Min resources given a latency?

Computing Slack (mobility)

- Slack of Operator *i*: $S_i = TS_i^{ALAP} - TS_i^{ASAP}$
  - Defines mobility of the operators



$S_6 = 2\text{-}1 = 1$

$S_7 = 2\text{-}1 = 1$

$S_8 = 3\text{-}1 = 2$

$S_9 = 4\text{-}2 = 2$

$S_{10} = 3\text{-}1 = 2$

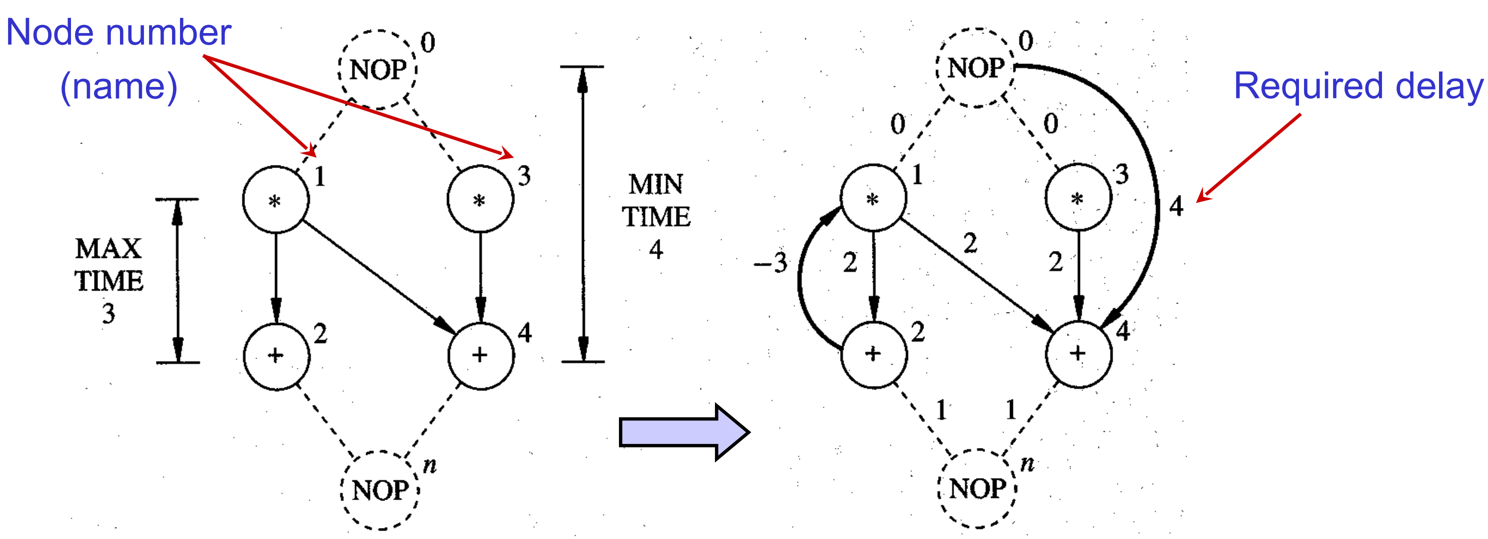
$S_{11} = 4\text{-}2 = 2$

Timing Constraints

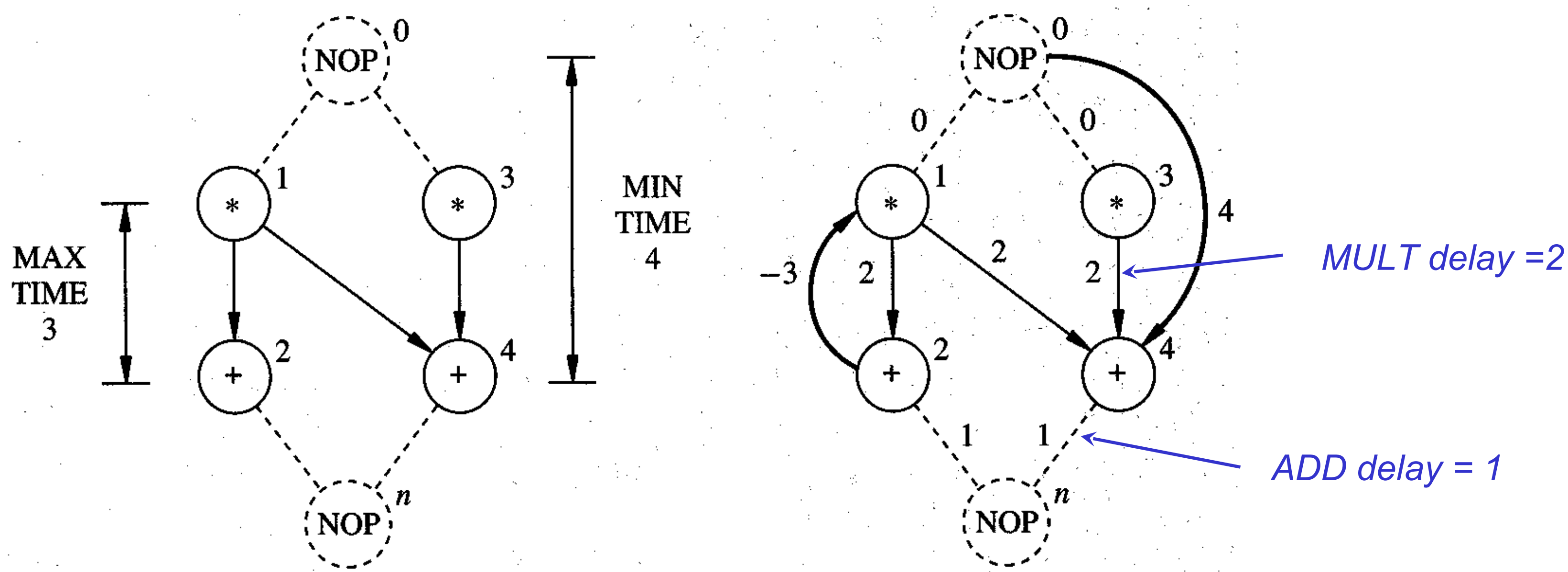
- Time measured in *cycles* or *control steps*
- Imposing relative timing constraints between operators *i* and *j*
  - max & min timing constraints

A **minimum** timing constraint $l_{ij} \geq 0$ requires: $t_j \geq t_i + l_{ij}$.

A **maximum** timing constraint $u_{ij} \geq 0$ requires: $t_j \leq t_i + u_{ij}$.

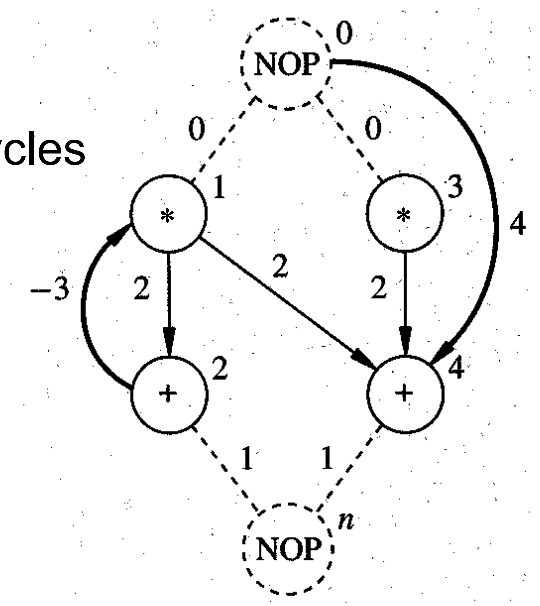


Constraint Graph $G_c(V,E)$



Existence of Schedule under Timing Constraints

- Upper bound (max timing constraint) is a problem
- Examine each *max* timing constraint (*i, j*):
  - Longest weighted path between nodes *i* and *j* must be $\leq$ *max* timing constraint u_{ij}.
  - Any cycle in G_c including edge (*i, j*) must be negative or zero
- Necessary and sufficient condition:
  - The constraint graph G_c must not have positive cycles



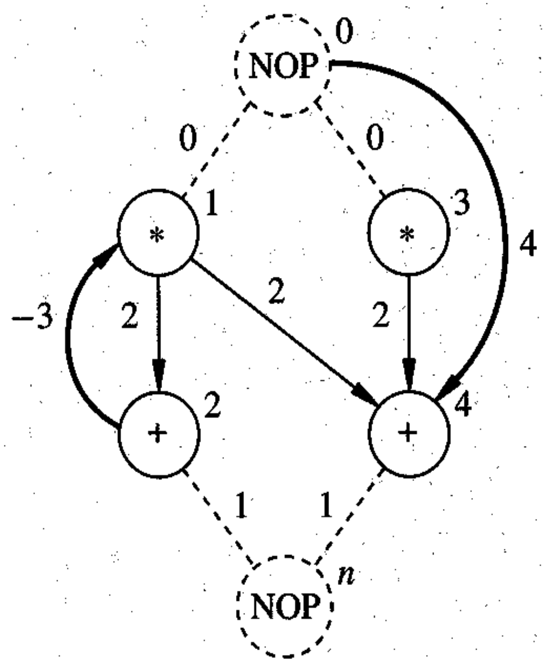
- Example:
  - Assume delays: ADD=1, MULT=2
  - Path {1 → 2} has weight $2 \leq u_{12}=3$, that is cycle {1,2,1} has weight = -1, OK
  - No positive cycles in the graph, so it has a consistent schedule

Existence of schedule under timing constraints

- Example: satisfying assignment
  - Assume delays: ADD=1, MULT=2
  - Feasible assignment:

    Vertex Start time

- $v_0 \rightarrow$ step 1
- $v_1 \rightarrow$ step 1
- $v_2 \rightarrow$ step 3
- $v_3 \rightarrow$ step 1
- $v_4 \rightarrow$ step 5
- $v_n \rightarrow$ step 6



Scheduling – a Combinatorial Optimization Problem

- NP-complete Problem
- Optimal solutions for special cases and ILP
- Heuristics - iterative Improvements
- Heuristics – constructive
- Various versions of the problem
  - Unconstrained, minimum latency
  - Resource-constrained, minimum latency
  - Timing-constrained, minimum latency
  - Latency-constrained, minimum resource
- If all resources are identical, problem is reduced to multiprocessor scheduling (Hu's algorithm)
  - Minimum latency multiprocessor problem is intractable

Observation about ALAP & ASAP

- No consideration given to resource constraints
- No priority is given to nodes on critical path
- As a result, less critical nodes may be scheduled ahead of critical nodes
  – No problem if unlimited hardware is available
  – However if the resources are limited, the less critical nodes may block the critical nodes and thus produce inferior schedules

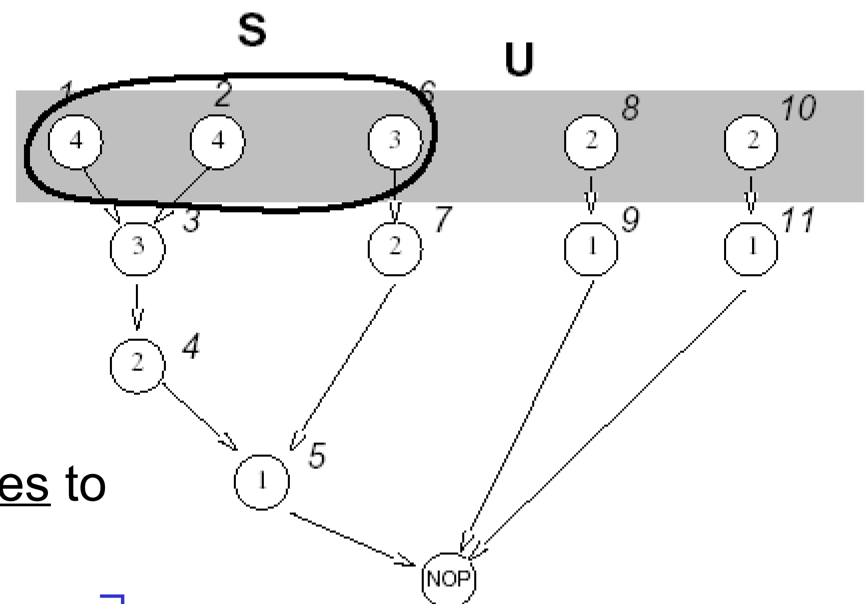
- *List scheduling* techniques overcome this problem by utilizing a more global node selection criterion

Hu's Algorithm

- Simple case of the scheduling problem
  - Each operation has unit delay
  - Each operation can be implemented by the same operator (*multiprocessor*)
- Hu's algorithm
  - Greedy, polynomial time
  - Optimal for trees and single type operations
  - Computes minimum number of <u>resources</u> for a given latency (MR-LCS), *or*
  - computes minimum <u>latency</u> subject to resource constraints (ML-RCS)
- Basic idea:
  - Label operations based on their distances from the sink
  - Try to schedule nodes with higher labels first (i.e., most "critical" operations have priority)

Hu's Algorithm

- Labeling of nodes
  – Label operations based on their distances from the sink
- Notation
  - α_i = label of node i
  - $\alpha = \max_i \alpha_i$
  – $p(j)$ = # vertices with label j



- Theorem (Hu)

  Lower bound on the <u>number of resources</u> to complete schedule with latency L is

$$a_{min} = \max_\gamma \left\lceil \sum^{\gamma}_{j=1} p(\alpha+1-j) \,/\, (\gamma+L-\alpha) \right\rceil$$

where γ is a positive integer ($1 \le \gamma \le \alpha + 1$)

- In this case: a_{min}= 3 (number of operators needed)

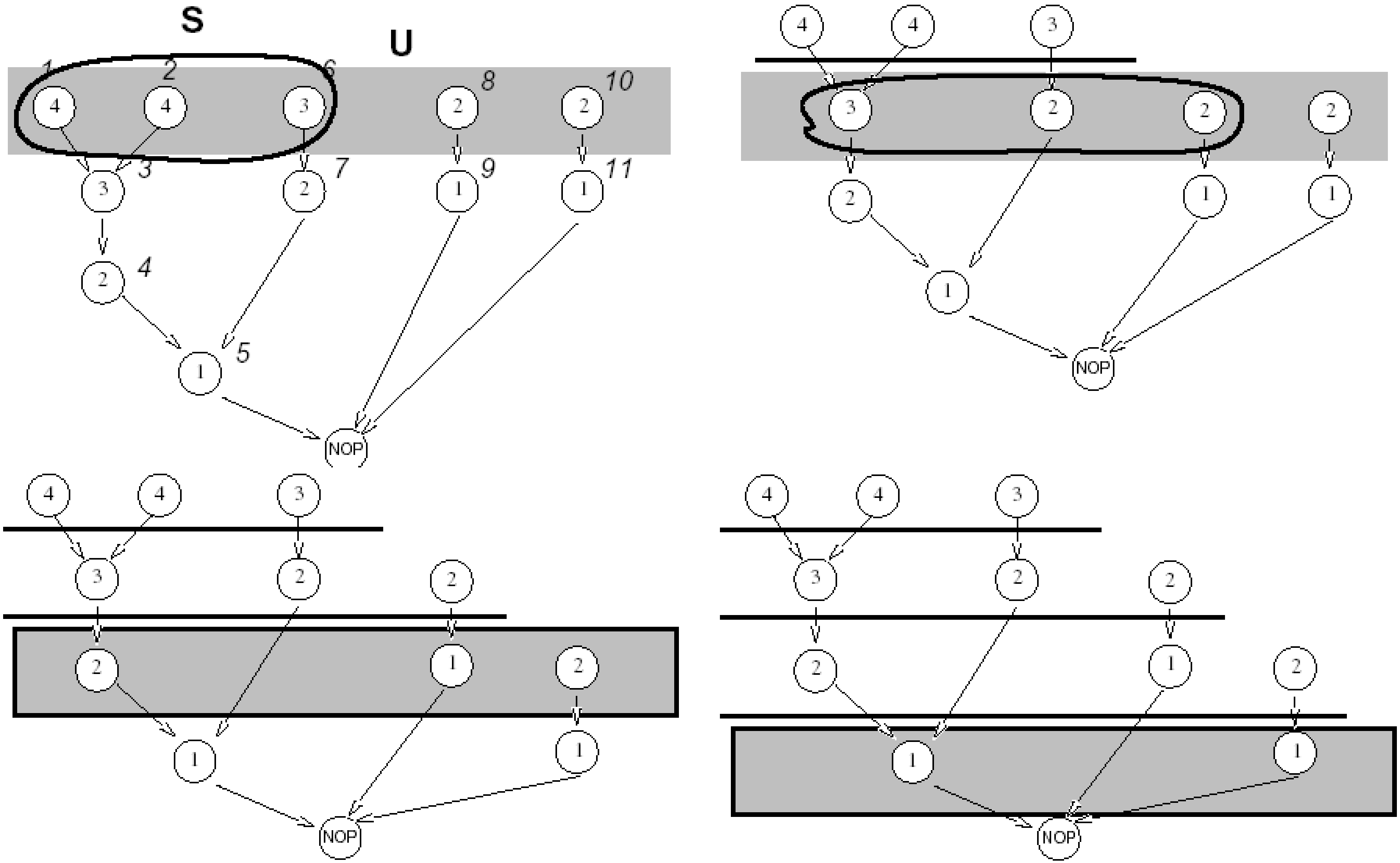
Hu's Algorithm (min Latency s.t. Resource constraint)

```
HU (G(V,E), a) {
    Label the vertices          // a = resource constraint
                                // label = length of longest path
                                          passing through the vertex

    l = 1
    repeat {
        U = unscheduled vertices in V whose
            predecessors have been already scheduled
              (or have no predecessors)

        Select S ⊆ U such that |S| ≤ a and labels in S are maximal
        Schedule the S operations at step l by setting
            t_i = l,  ∀ v_i ∈ S;
        l = l + 1;
    } until v_n is scheduled.
}
```

Hu's Algorithm: Example (a=3)



List Scheduling (arbitrary operators)

- Extend the idea to several operators
- Greedy algorithm for ML-RCS and MR-LCS
  - Does NOT guarantee optimum solution
- Similar to Hu's algorithm
  - Operation selection decided by criticality
  - *O(n)* time complexity
- Considers a more general case
  - Resource constraints with different resource types
  - Multi-cycle operations
  - Pipelined operations

List Scheduling Algorithms

- Algorithm 1: Minimize latency under resource constraint (ML-RC)
  - Resource constraint represented by vector ***a*** (indexed by resource type)
    - Example: two types of resources, MULT (a_1=1), ADD (a_2=2)
- The candidate operations $U_{l,k}$
  - those operations of type k whose predecessors have already been scheduled early enough so that they are completed at step l:

    $U_{l,k} = \{ v_i \subseteq V: type(v_i) = k$ and $t_j + d_j \leq l$, for all j: $(v_j, v_i) \subseteq E$
- The unfinished operations $T_{l,k}$
  - those operations of type k that started at earlier cycles but whose execution has not finished at step l:

    $T_{l,k} = \{ v_i \subseteq V: type(v_i) = k$ and $t_i + d_i > l$
- Priority list
  - List operators according to some heuristic urgency measure
  - Common priority list: labeled by position on the longest path in decreasing order
- Algorithm 2: Minimize resources under latency constraint (MR-LC)

List Scheduling Algorithm 1: ML-RC

Minimize latency under resource constraint

LIST_L (G(V,E), **a**) {   // resource constraints specified by vector **a**

    $l = 1$

    repeat {

    for each resource type k {

    $U_{l,k}$ = candidate operations available in step l

    $T_{l,k}$ = unfinished operations (in progress)

    Select $S_k \subseteq U_{l,k}$ such that $|S_k| + |T_{l,k}| \leq a_k$

    Schedule the S_k operations at step l

    }

    $l = l + 1$

    } until v_n is scheduled

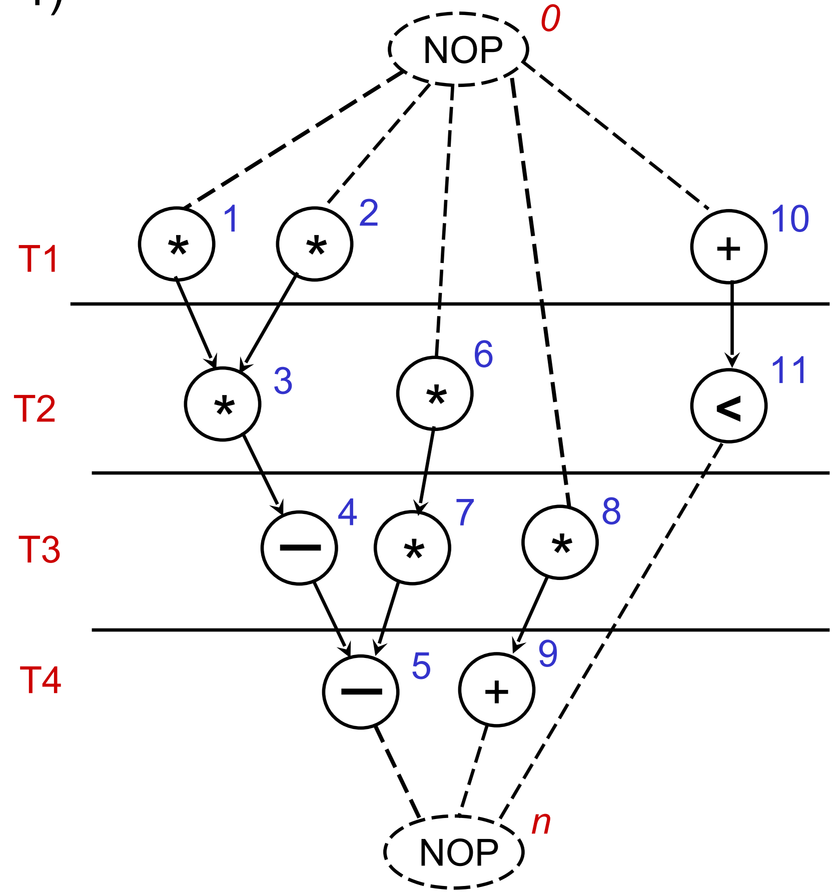
}

Note: If for all operators *i*, $d_i = 1$ (unit delay), the set $T_{l,k}$ is empty

List Scheduling – Example 1 (a=[2,2])

MLRC Minimize latency under resource constraint (with d = 1)

- Assumptions
  - All operations have unit delay (d_i=1)
  - Resource constraints:
    MULT: $a_1 = 2$, ALU: $a_2 = 2$
- Step 1:
  - $U_{1,1} = \{v_1, v_2, v_6, v_8\}$, select $\{v_1, v_2\}$
  - $U_{1,2} = \{v_{10}\}$, select + schedule
- Step 2:
  - $U_{2,1} = \{v_3, v_6, v_8\}$, select $\{v_3, v_6\}$
  - $U_{2,2} = \{v_{11}\}$, select + schedule
- Step 3:
  - $U_{3,1} = \{v_7, v_8\}$, select + schedule
  - $U_{3,2} = \{v_4\}$, select + schedule
- Step 4:
  - $U_{4,2} = \{v_5, v_9\}$, select + schedule



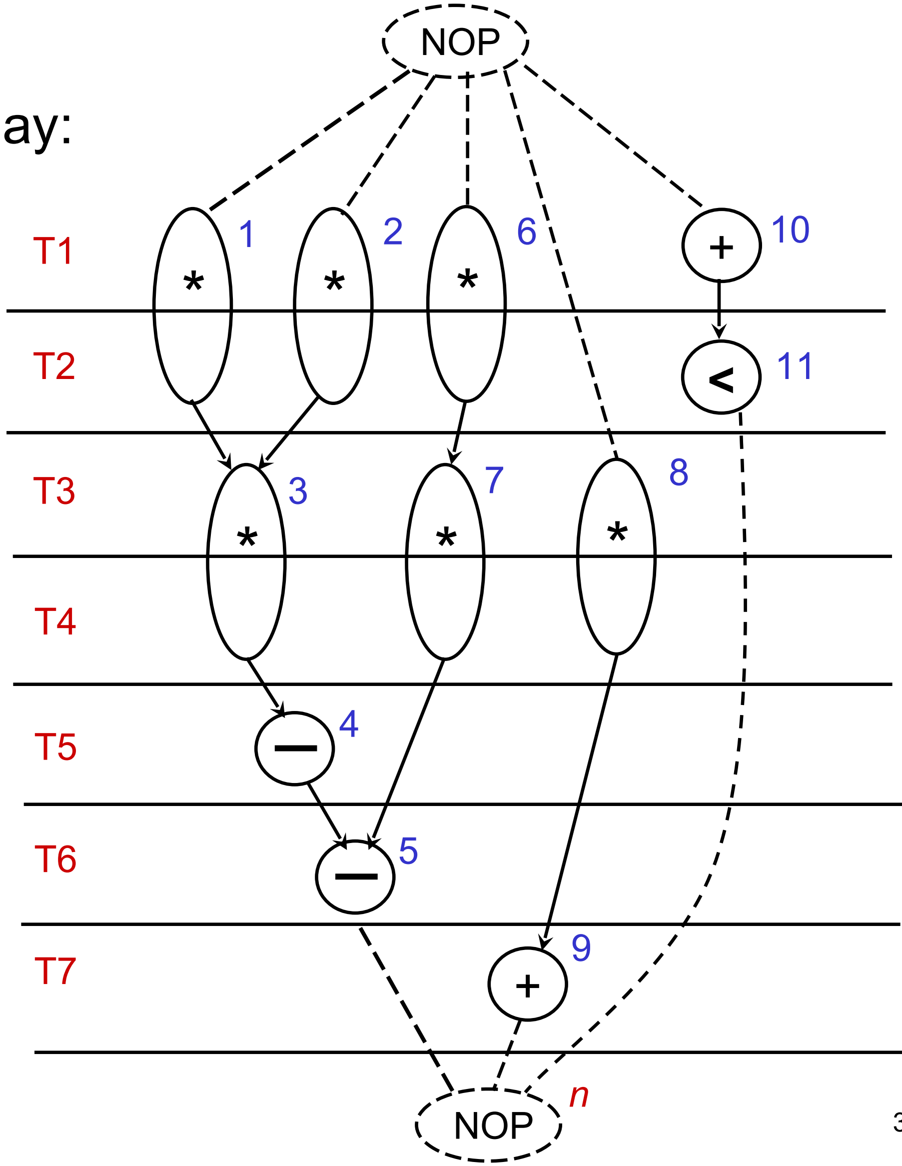
List Scheduling – Example 2 (a = [3,1])

MLRC Minimize latency under resource constraint (with d_1=2, d_2=1)

- Assumptions
  - Operations have different delay:
    $del_{MULT} = 2$, $del_{ALU} = 1$
  - Resource constraints:
    MULT: $a_1 = 3$, ALU: $a_2 = 1$

	MUTL	ALU	start time
U=	$\{v_1, v_2, v_6\}$	v_{10}	1
T=	$\{v_1, v_2, v_6\}$	v_{11}	2
U=	$\{v_3, v_7, v_8\}$	--	3
T=	$\{v_3, v_7, v_8\}$	--	4
	--	v_4	5
	--	v_5	6
	--	v_9	7

Latency L = 8



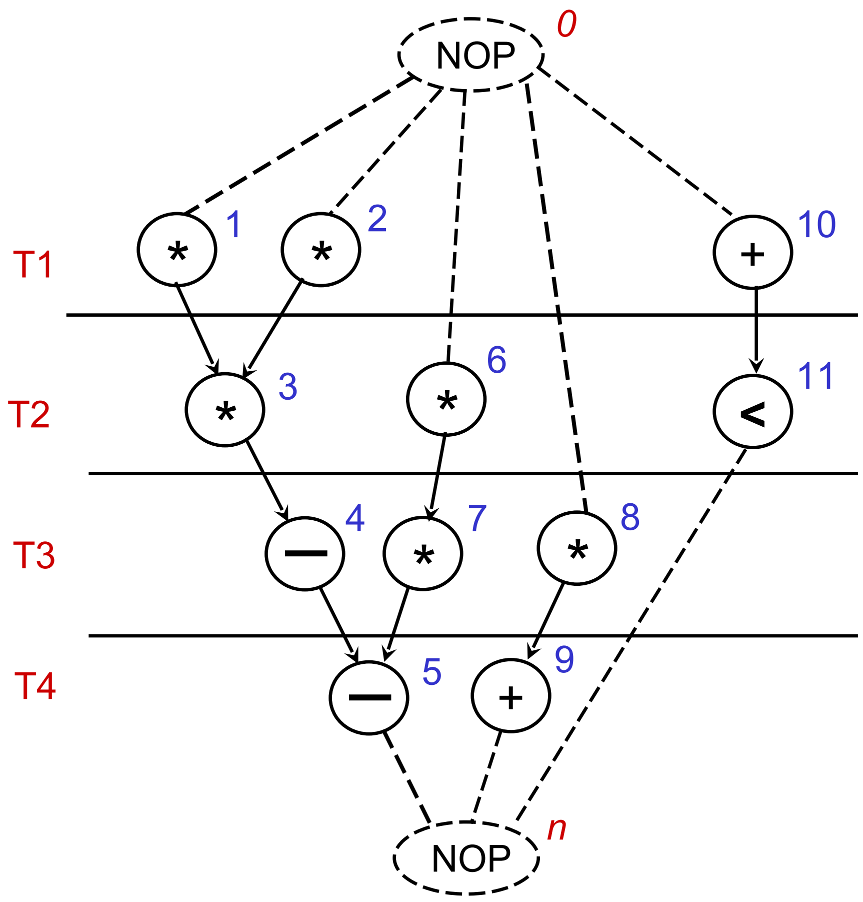
List Scheduling Algorithm 2: MR-LC

```
LIST_R (G(V,E), λ') {
    a = 1,    l = 1
    Compute latest possible starting times t^L using ALAP algorithm
    repeat {
        for each resource type k {
            U_{l,k} = candidate operations;
            Compute slacks { s_i = t_i^L - l, ∀ v_i ∈ U_{l,k} };
            Schedule operations with zero slack, update a;
            Schedule additional S_k ⊆ U_{l,k} requiring no additional resources;
        }
        l = l + 1
    } until v_n is scheduled.
}
```

List Scheduling – Example 3

MRLC Minimize resources under latency constraint

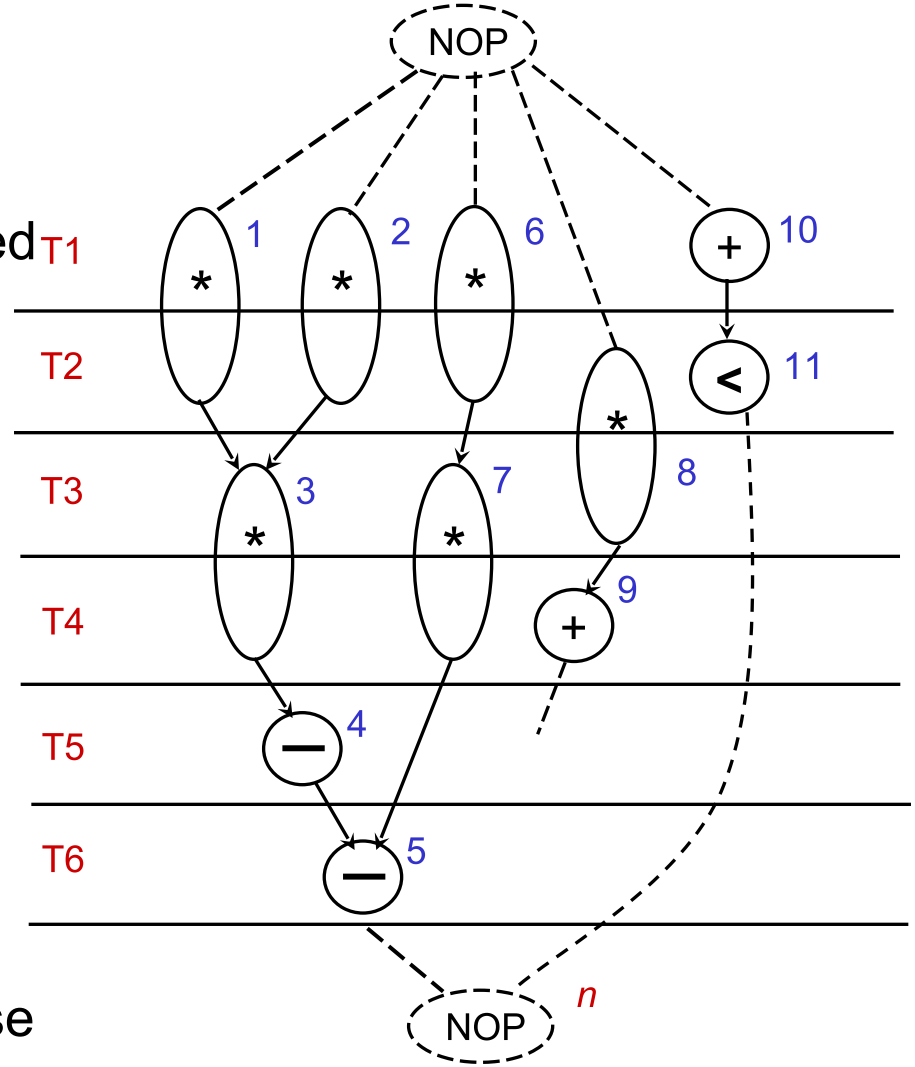
- Assumptions
  - All operations have unit delay (d_i=1)
  - Latency constraint: L = 4
- Use slack information to guide the scheduling
  - Schedule operations with slack=0 first
  - Add other operations only if resource limit allows
  - The lower the slack the more urgent it is to schedule the operation



List Scheduling – Pipelined Operations

Minimize latency under resource constraint (*a_1=3 Mults, a_2=3 ALUs*)

- Assumptions
  - Multipliers are pipelined
  - Sharing between first and second pipeline stage allowed for different multipliers



-

MUTL	ALU	start time
$\{v_1, v_2, v_6\}$	v_{10}	*1*
v_8	v_{11}	*2*
$\{v_3, v_7\}$	--	*3*
--	v_9	*4*
--	v_4	*5*
--	v_5	*6*

- *L=7,* Compare to multi-cycle case

Scheduling – a Combinatorial Optimization Problem

- NP-complete Problem
- Optimal solutions for special cases (trees) and ILP
- Heuristics
  - iterative Improvements
  - constructive
- Various versions of the problem
  - Minimum latency, unconstrained (ASAP)
  - Latency-constrained scheduling (ALAP)
  - Minimum latency under resource constraints (ML-RC)
  - Minimum resource schedule under latency constraint (MR-LC)
- If all resources are identical, problem is reduced to multiprocessor scheduling (Hu’s algorithm)
  - Minimum latency multiprocessor problem is intractable for general graphs
  - For trees greedy algorithm gives optimum solution