Introduction to Scheduling

Dr. Chandan Karfa

Department of Computer Science and Engineering

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

High-level Synthesis

Example: 2nd order differential equation solver

```
Diffeq: (x, dx, u, a, clock, y)
input: x, dx, u, a, clock;
output: y
while(x < a)
        u1 = u-(3*x*u*dx)-(3*y*dx);
        y1 = y+(u*dx);
        x1 = x+dx;
        x = x1, y = y1, u = u1;
end
```

High-level Behaviour

HLS →

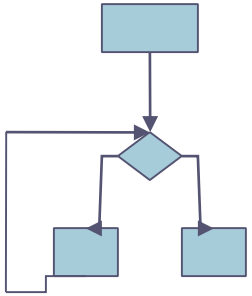
```
always @(posedge ap_clk) begin
   if(1'b1 == ap_CS_fsm_state5) begin
    j_reg_126 <= j_4_reg_293;
   end else if((1'b1 == ap_CS_fsm_state1) & (ap_start == 1'b1)) begin
    j_reg_126 <= 3'd0;
   end
  end

 assign tmp_108_fu_235_p1_temp_6 = tmp_108_fu_235_p1 & 63'd12;
  assign statemt_addr_28_reg_324_temp_7 = statemt_addr_28_reg_324 &
4'd19;
  assign tmp_108_fu_235_p1_temp_6_temp_8 = tmp_108_fu_235_p1_temp_6
| statemt_addr_28_reg_324_temp_7;

ap_ST_fsm_state2: begin
      if((exitcond_fu_175_p2 == 1'd1) & (1'b1 == ap_CS_fsm_state2)) begin
        ap_NS_fsm = ap_ST_fsm_state1;
      end else begin
        ap_NS_fsm = ap_ST_fsm_state3;
      end
     end
```

Register Transfer Level Description

High-level Synthesis Steps

```
while (x_var < a_var) loop
    t1 := u_var * dx_var;
    t2 := 3 * x_var;
    t3 := 3 * y_var;
    t4 := t1 * t2;
    t5 := dx_var * t3;
    t6 := u_var - t4;
    u_var := t6 - t5;
    y1 := u_var * dx_var;
    y_var := y_var + y1;
    x_var := x_var + dx_var;
 end loop;
X <= x_var;
Y <= y_var;
U <= u_var;
```

pre-processing



Input behaviour

scheduling

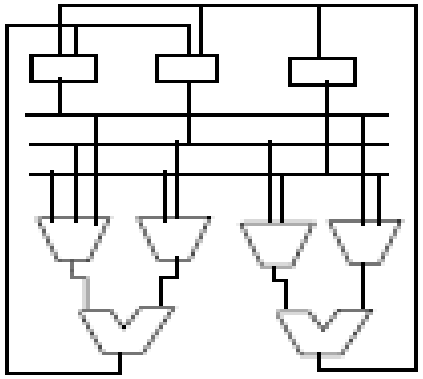
1.	< 1 *>		
2.		<2 + >	<0 * >
3.		<4 * >	| * |
4.		| * |	<3 *>
5.	<7 *>	<6 *>	| * |
6.	| * |	| * |	<5 - >
7.		<9 +>	<8 - >

Allocation & binding

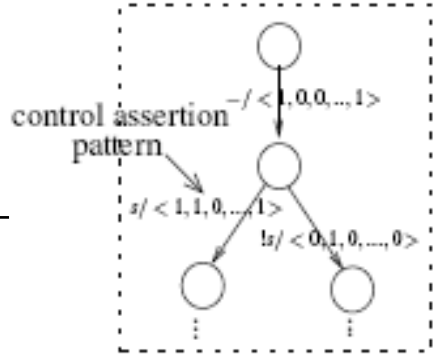
R1 : 3, v1
R2 : x u, v5
R3 : v0, v6
R4 : v3

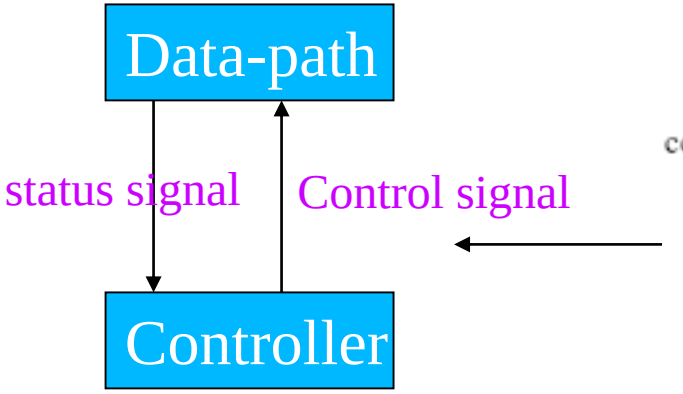
FU1: op1, on3. ..
FU2: op2, op5, …
FU3: …

Data-path generation



Controller generation





RTL behaviour

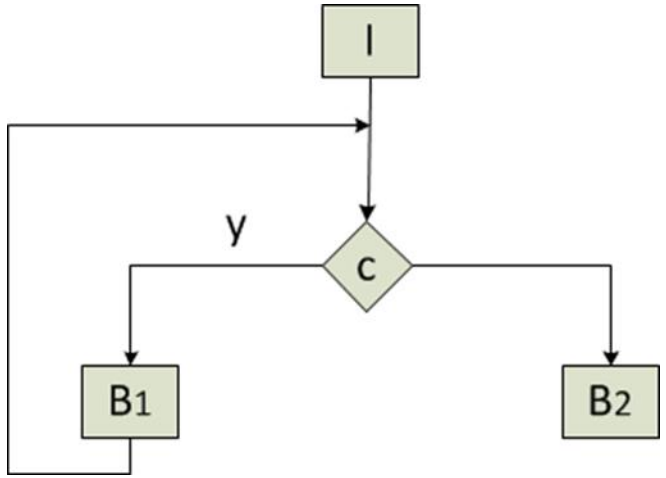
High-level Synthesis Steps

- Preprocessing: Intermediate representation (CDFG) construction, data-dependency, live variable analysis, compiler optimization.
- Scheduling: Assigns control step to the operations of the input behaviour.
- Allocation: Computes minimum number of functional units and registers.
- Binding: Variables are mapped to registers, operation to functional units, data transfers to the interconnection units.
- Data path & Controller design: controller is designed based on inter connections among the data path elements, data transfer required in different control steps.

Working with an example

Example: 2nd order differential equation solver

```
Diffeq: (x, dx, u, a, clock, y)
input: x, dx, u, a, clock;
output: y
while(x < a)
        u1 = u-(3*x*u*dx)-(3*y*dx);
        y1 = y+(u*dx);
        x1 = x+dx;
        x = x1, y = y1, u = u1;
end
```



CDFG

Preprocessing

Example: 2nd order differential equation solver

```
Diffeq: (x, dx, u, a, clock, y)
input: x, dx, u, a, clock;
output: y
while(x < a)
        u1 = u-(3*x*u*dx)-(3*y*dx);
        y1 = y+(u*dx);
        x1 = x+dx;
      x = x1, y = y1, u = u1;
end
```

I

Read(p1, dx)

Read(p2, x)

Read(p3, a)

Read(p1,y)

Read(p2, u)

c = x < a

B2

Write(p1, y)

B_1

$V_1 : t_1 = u * dx$

$V_2 : t_2 = 3 * x$

$V_3 : t_3 = 3 * y$

$V_4 : t_4 = u * dx$

$V_5 : t_5 = t_1 * t_2$

$V_6 : t_6 = t_3 * dx$

$V_7 : t_7 = u - t_5$

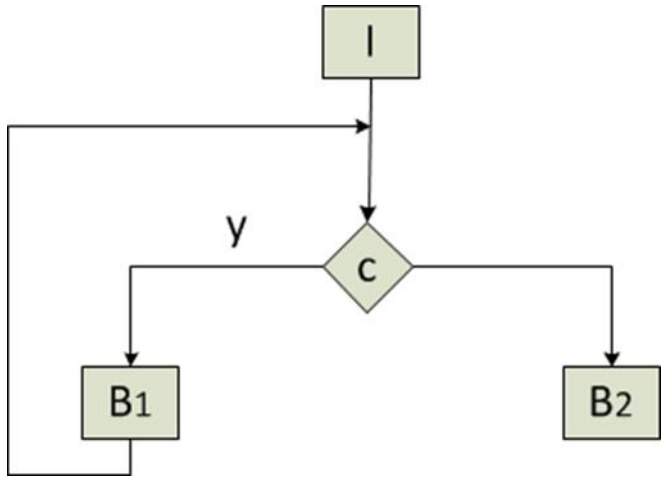
$V_8 : u = t_7 - t_6$

$V_9 : y = y + t_4$

$V_{10} : x = x + dx$

$V_{11} : c = x < a$

Basic Blocks with 3-address codes



Control and Dataflow graph (CDFG)

Preprocessing

B_1

$V_1 : t_1 = u * dx$

$V_2 : t_2 = 3 * x$

$V_3 : t_3 = 3 * y$

$V_4 : t_4 = u * dx$

$V_5 : t_5 = t_1 * t_2$

$V_6 : t_6 = t_3 * dx$

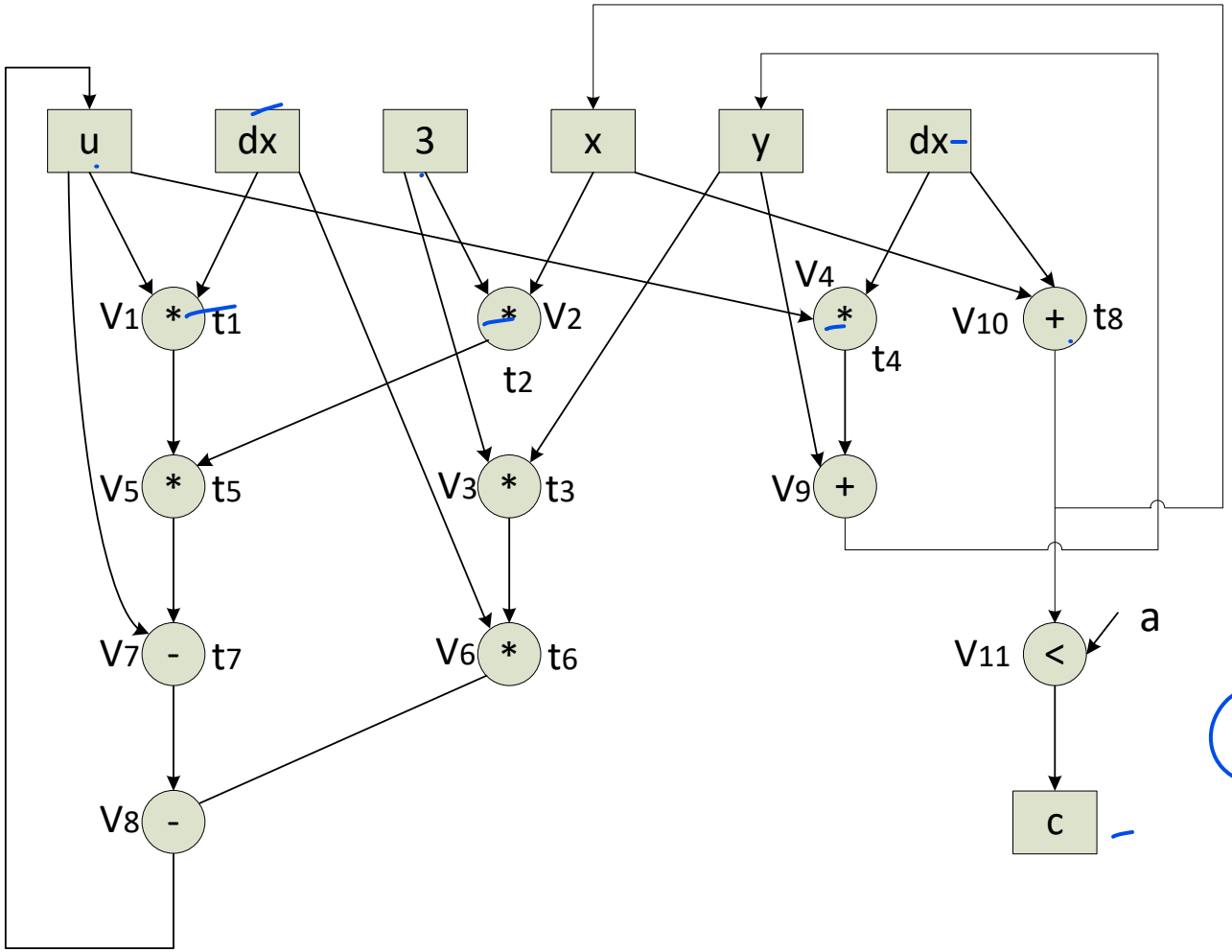
$V_7 : t_7 = u - t_5$

$V_8 : u = t_7 - t_6$

$V_9 : y = y + t_4$

$V_{10} : x = x + dx$

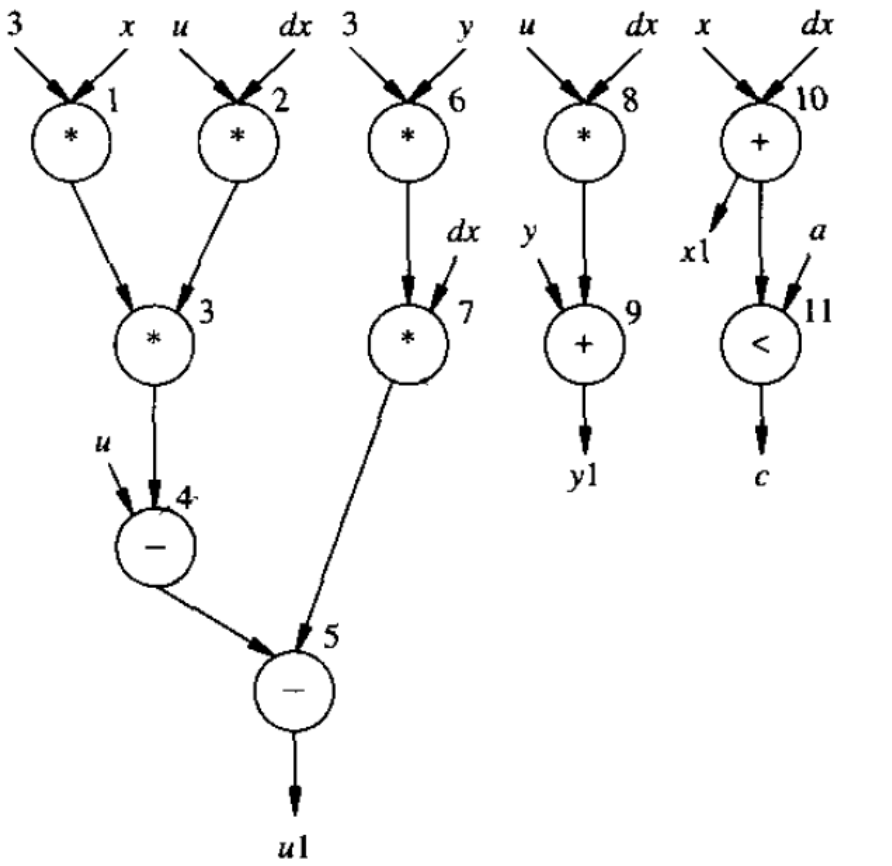
$V_{11} : c = x < a$



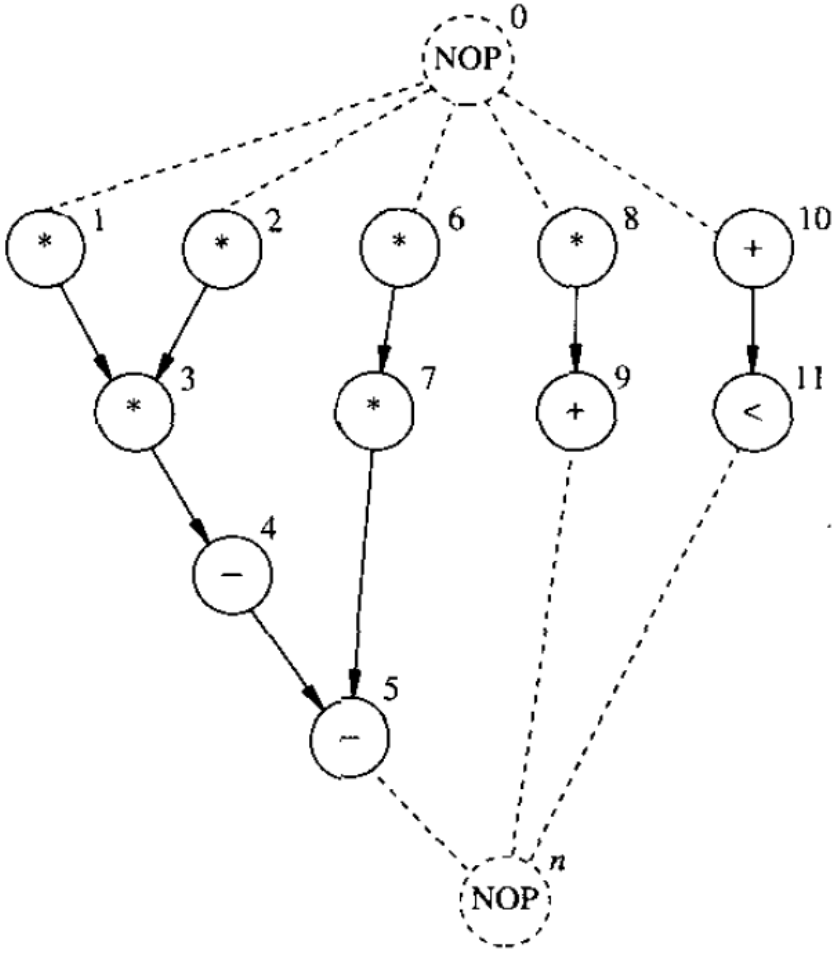
Date dependency graph

12

Sequence Graph

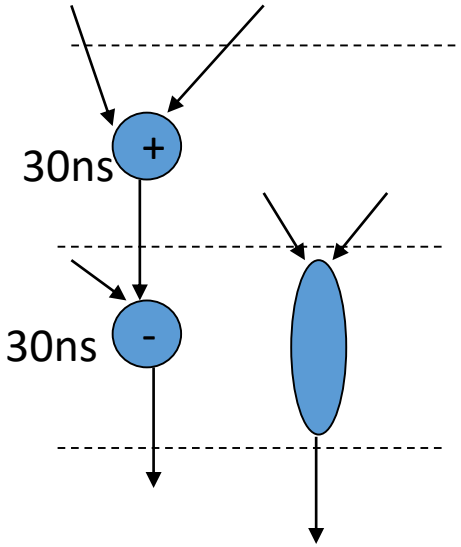


Dataflow graph



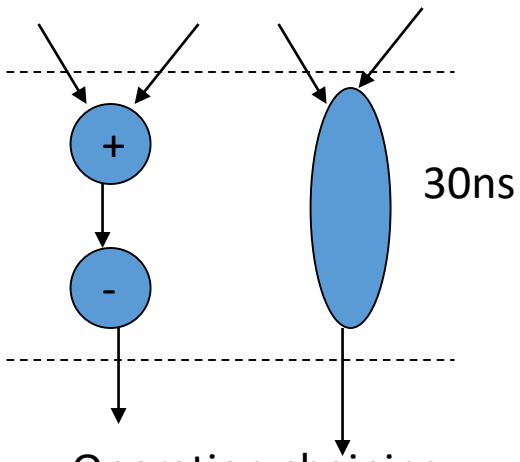
Sequence Graph

Operation Delay

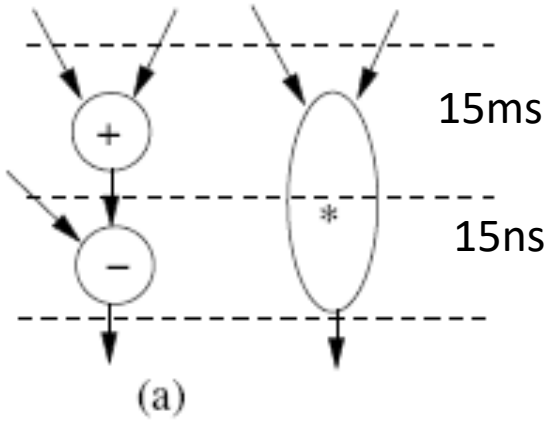


Multiplication: 30ns

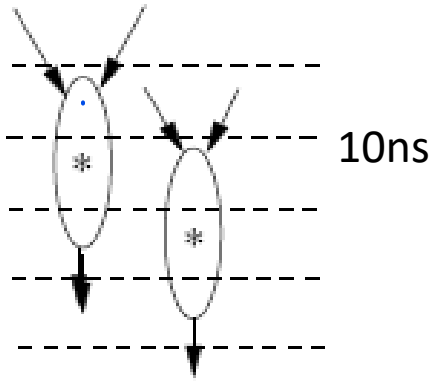
Add/Sub: 10ns



Operation chaining



Multi-cycling

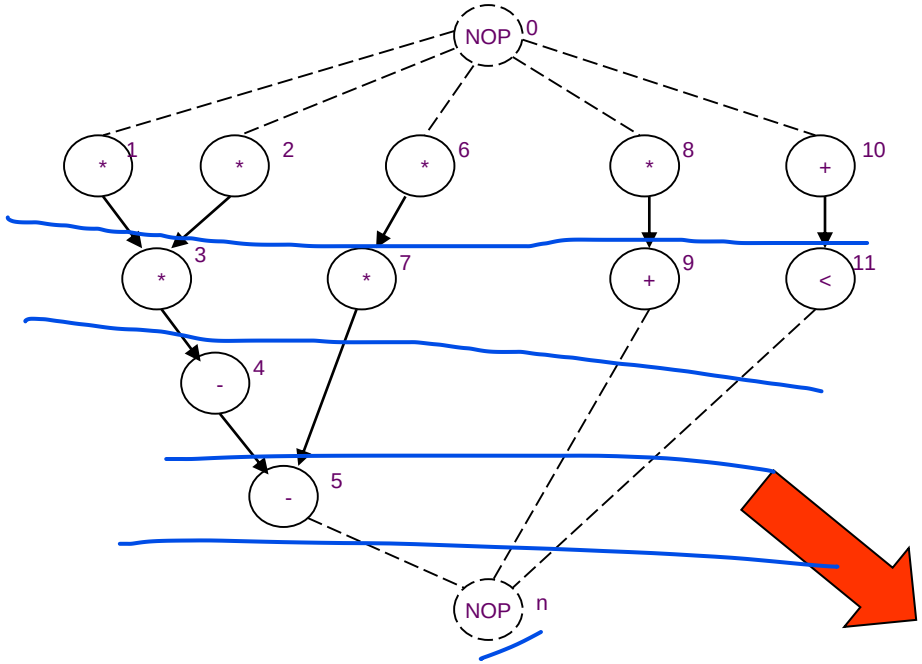


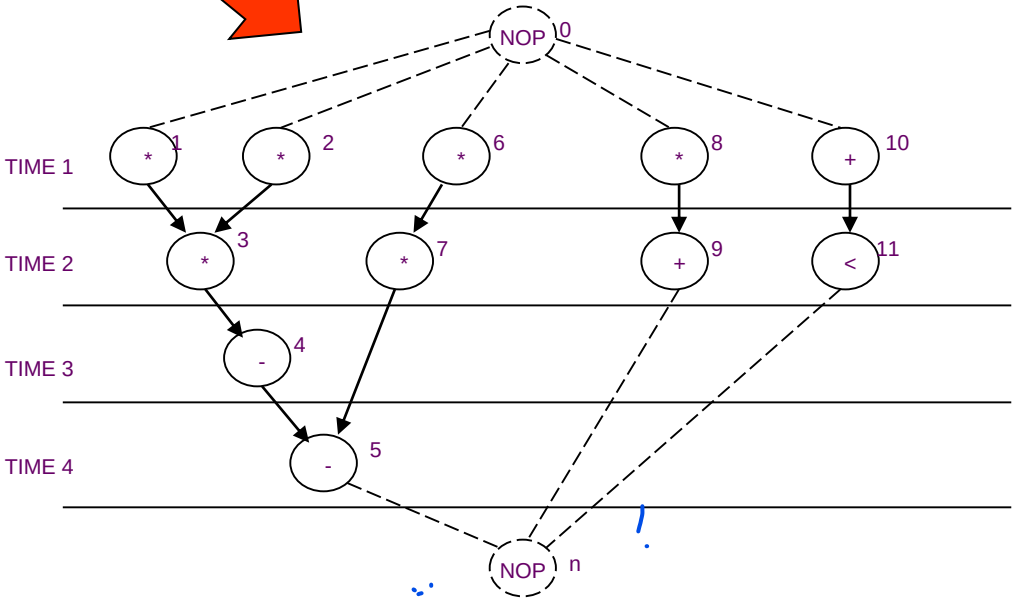
Pipelining

Scheduling

- Circuit model:
  - Sequencing graph
  - Cycle-time is fixed
  - 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





$\lambda = 5$

Scheduling Problem Formulation

Input:

- Sequence Graph G = (V, E), |V| = n
- Delay of each node. D = {d_i, i = 0, 1, …, n}
- Resource or Timing Constraints (optional)

→ ASAP ALAP

Output:

- The start time of each node T = {t_i, i=0, 1, 2, …, n}
- Latency: number of cycles to execute the entire schedule. Difference of start time of source node and sink node; latency = $t_n - t_0$

The start time of an operation is at least as large as the start time of each of its direct predecessor plus its execution delay

$$t_i \geq t_j + d_j \quad \forall\, i, j \quad : \; (v_j, v_i) \in E$$

Scheduling Problems

- Minimum Latecny Unconstrained minimum-latency scheduling problem (Unconstraint)
- Minimum latency under resource constraints (MLRC)
- Minimum resource under latency constraints (MRLC)

Required for getting λ

Unconstraint Scheduling: Minimize t_n.

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 φ, : V -> Z+ such that Dependency constraints are satisfied

$t_i + d_j$

t_j

t_i

t_i = φ(v_i), t_i >= t_j + d_j for all i, j: (v_j, v_i) in E and t_n is minimum.

Minimum latency under resource constraints (MLRC)

- There are n_{res}, resource types, T : V → {I, 2, . . . , n_{res},].
- A bound on number of resource is given: (a_k, 1, 2, …, n_{res}}
- MLRC: The operations are scheduled in such a way that the number of operations of any given type executing in any schedule step does not exceed the bound.
  - Objective: Minimize t_n such that
    - Dependency constraints are satisfied
    - Resource constraints are satisfied

Minimum Resource under latency constraints (MRLC)

- Additional constraint: Latency
  - Latency bound must be satisfied
- Resource usage is unknown in the constraints (a_k is unknown)
- Resource usage is the objective to minimize
  - Objective: Minimize $\cancel{t_n}$ a_k such that
    - Dependency constraints are satisfied
    - Latency constraint is satisfied

Minimum-latency unconstrained scheduling problem

- Given a set of ops *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:

$$t_i = \varphi(v_i),$$

$$t_i \geq t_j + d_j \qquad \forall\, i, j \text{ s.t. } (v_j, v_i) \in E$$

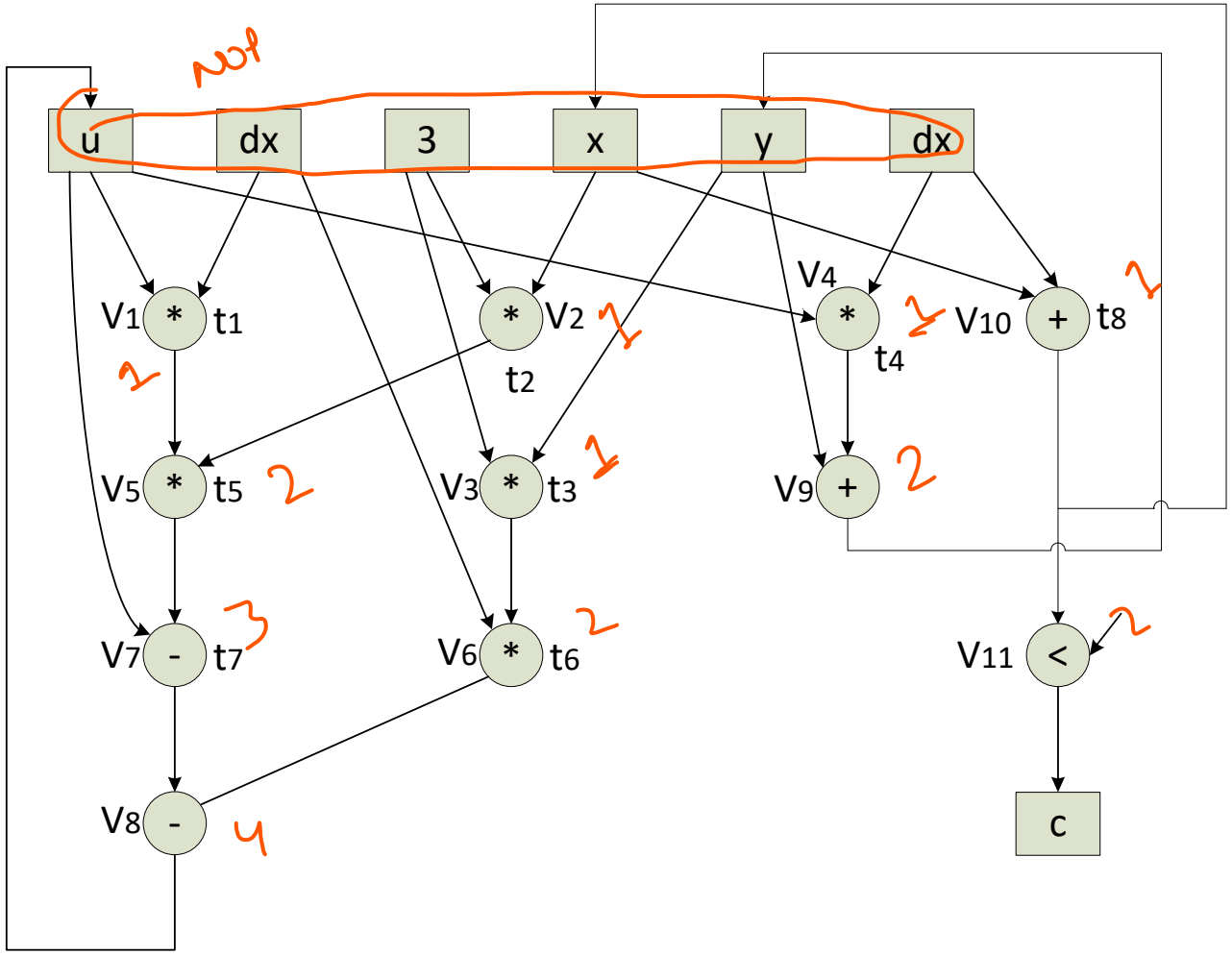
and t_n is minimum

ASAP scheduling algorithm

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

Here we are scheduling it as soon as its predecessor finished

ASAP Example

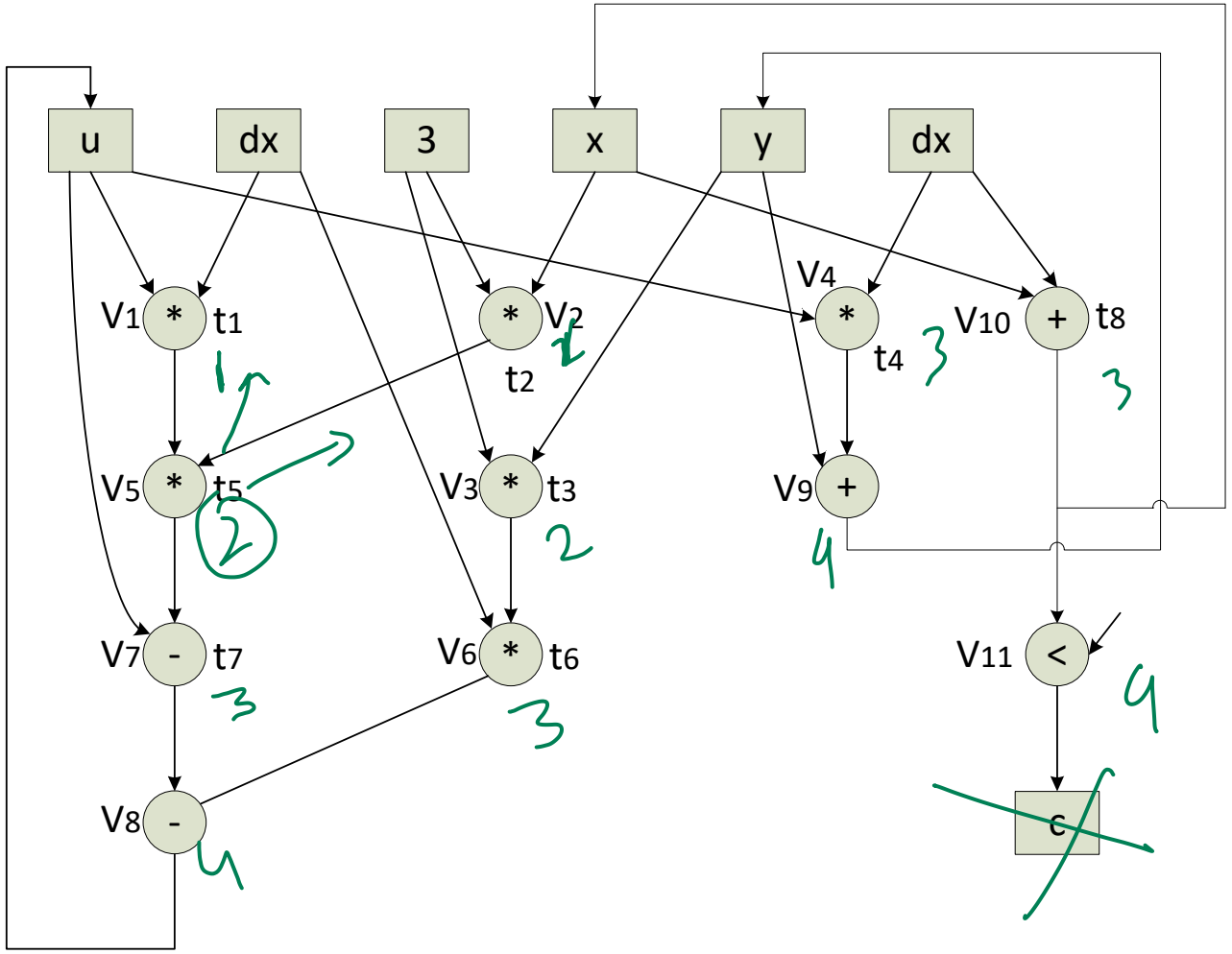


ALAP scheduling algorithm

```
ALAP ( G_s(V,E), λ̄ ) {
        Schedule v_n by setting t_n = λ̄ + 1;
        repeat {
                Select a vertex v_i whose successors are all scheduled;
                Schedule v_i by setting t_i = min_{j:(v_i,v_j) ϵ E} t_j - d_i;
        }
        until (v_0 is scheduled);
        return (t);
}
```

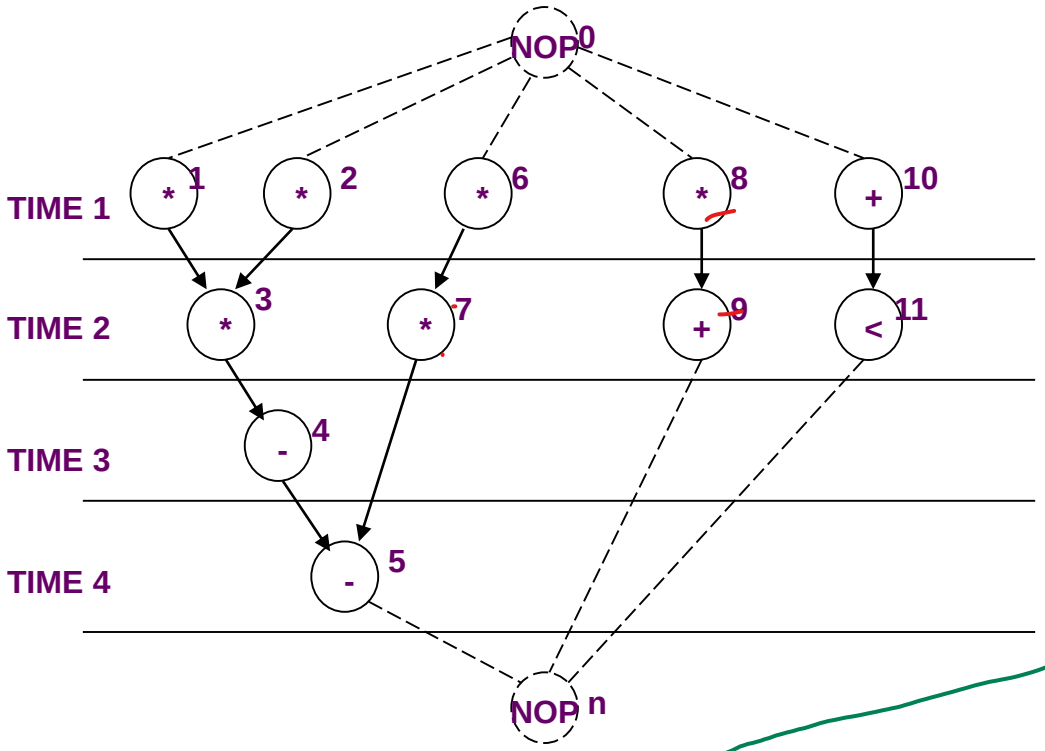
Latency constraints is given; Objective is to schedule the operation as late as possible without violating latency constrants

ALAP Example

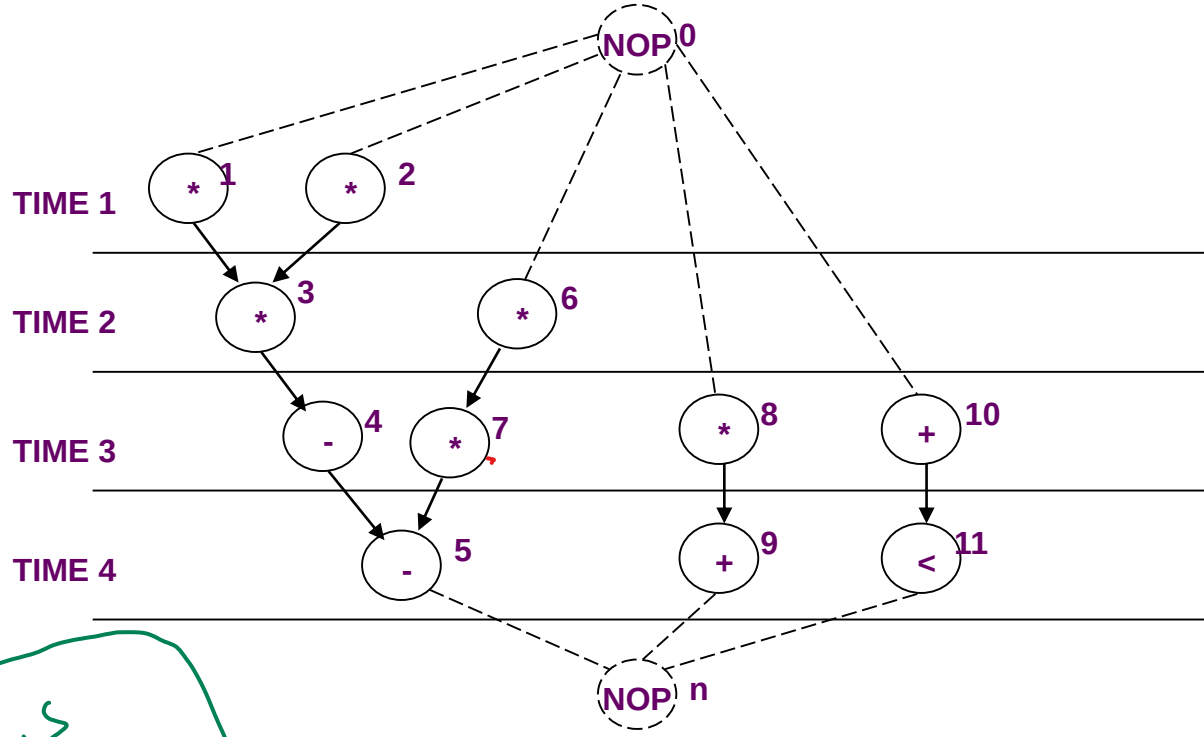


(c) ASAP → A

Mobility



ASAP



ALAP

$$Mob(V_i) = t_i^L - t_i^S$$

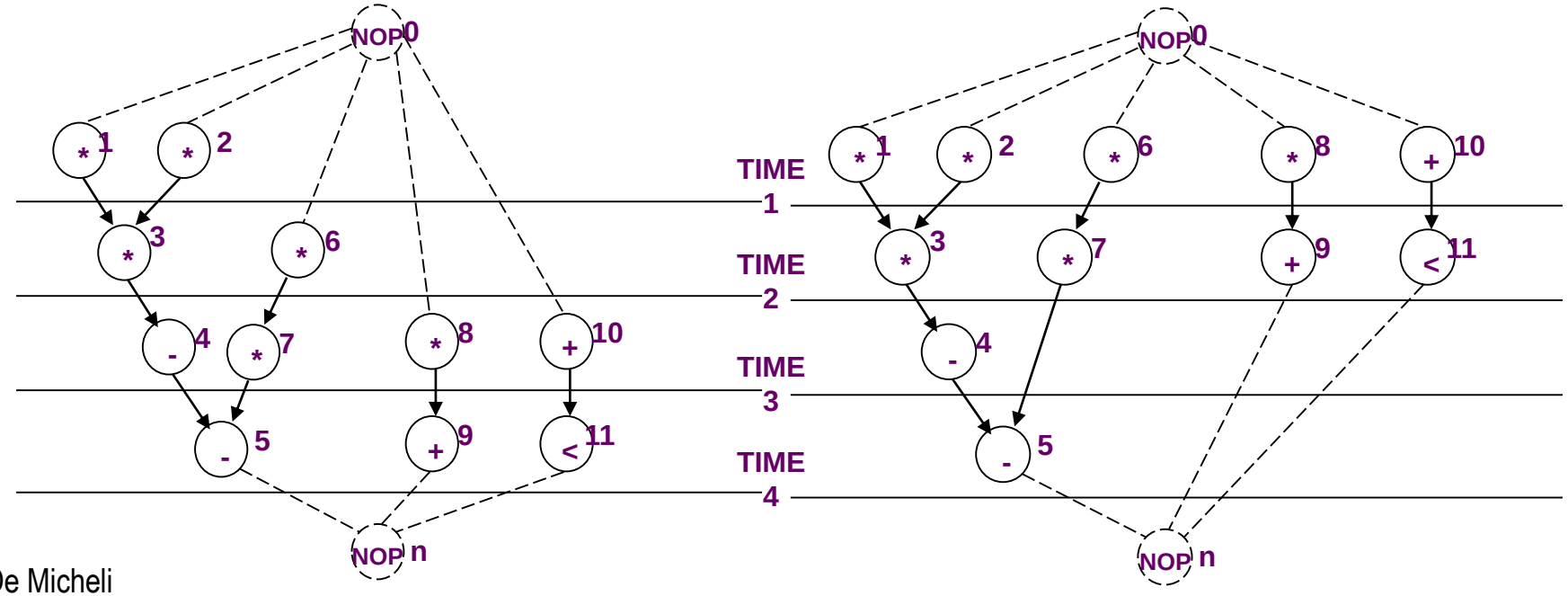
(μ) ALAP ASAP

Remarks

- ALAP solves a latency-constrained problem
- Latency bound can be set to latency computed by ASAP algorithm
- Mobility: → Imp
  - Defined for each operation
  - Difference between ALAP and ASAP schedule
- Slack on the start time

Example

- **Operations with zero mobility:**
  - $\{ v_1, v_2, v_3, v_4, v_5 \}$
  - **Critical path**
- **Operations with mobility one:**
  - $\{ v_6, v_7 \}$
- **Operations with mobility two:**
  - $\{ v_8, v_9, v_{10}, v_{11} \}$



Thank You