



# Computer Architecture

**Lecturer: Mihai Negru**

2<sup>nd</sup> Year, Computer Science

## Lecture 2: High Level Synthesis

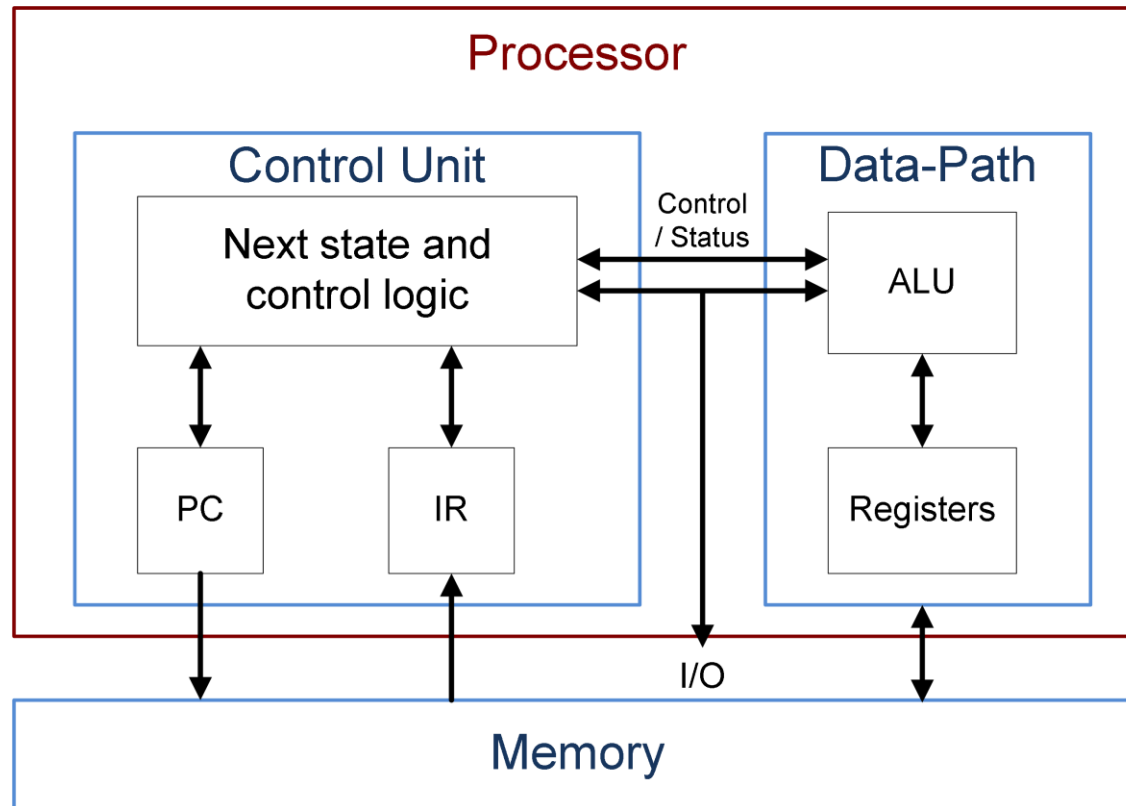
<http://users.utcluj.ro/~negrum/>



# Register Transfer Level (RTL)



- Digital System
  - Data-Path – registers, busses, processing logic
  - Control Unit – determines the sequence of data-processing operations of the Data-Path





# Register Transfer Level (RTL)



- A system is described at RTL level by the transfer of information between the memory elements of the system

RT operation:  $R_{dest} \leftarrow f(R_{src_1}, R_{src_2}, \dots, R_{src_n})$

- **Abstract RTL** – a behavioral specification
  - Does not take into account the structure of the digital system
  - Not related to timing or resources
- **Physical or Concrete RTL** – provides an implementation of the behavioral specification based on a selected structure at clock period granularity
  - Related to resources and timing constraints
- **RTL Design**
  - Converts a behavioral specification (**Abstract RTL**) into a structural description (**Concrete RTL**)



- A digital system specified at Concrete RTL level includes the following 3 components:
  - The set of registers/memories in the system.
  - The functional units, which perform the required operations.
  - The control that supervises the sequence of operations in the system.
- **Register Transfer Level (RTL) / Register Transfer Notation (RTN)**
  - RTL represents an algebraic notation used to define machine-level operations
  - Not executed by a computer – used to explain/describe how the computer works.
  - **Micro-operation:** single register transfer operation
$$X \leftarrow Y + Z$$
  - **RTL statement:** (condition)  $\rightarrow$  {micro-operation,..., micro-operation};
$$(c > 0) \rightarrow X \leftarrow Y + Z$$



# Register Transfer Notation (RTN)



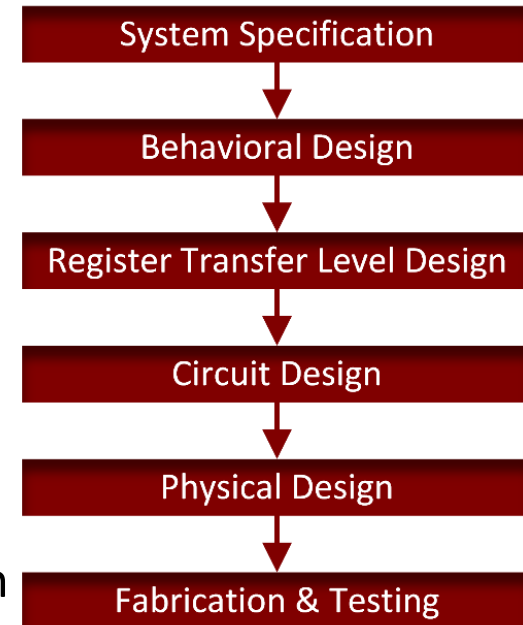
$\leftarrow$	Assignment
$=, \neq$	Tests for equality and inequality
$  $	Bit string concatenation
$X \leftarrow Y$	Data transfer of contents of regY to regX
$X \leftarrow 0$	Clears regX
$X \leftarrow Y + Z$	Adds contents of regY with regZ, load into regX
$X \leftarrow Y \vee Z$	ORs contents of regY with regZ, load into regX
$DR \leftarrow M[AR]$	Load into DR the contents of memory pointed to by AR
$R1 \leftarrow \gg R1$	Register R1 one bit right shift, with 0 into left-most bit
$R2 \leftarrow \ll R1$	Register R1 one bit left shift, with 0 into right-most bit
$X \leftarrow Y, A \leftarrow B$	Parallel transfers
$(\text{cond}) \rightarrow A \leftarrow B$	If cond = 1 then transfer contents of regB into regA
$S0 \rightarrow A \leftarrow B$	When in state S0, load regA with contents of regB
$P \cdot (a \cdot b) \rightarrow R2 \leftarrow R3$	When in state P, if a AND b is true then load R2 with contents of R3; a, b are signals



# High Level Synthesis (HLS)



- High Level Synthesis (Structural Synthesis)
  - Starts from an abstract behavioral description
  - Generates a Concrete RTL structural description
    - Functional units (+, -, \*), Memories, Interconnections
  - Obeys constraints (timing/size/performance)
- VHDL Synthesis (comparison)
  - Starts from an RTL description
  - Uses logic synthesis techniques to optimize the design
  - Generates a **standard-cell net list** → FPGA
- HLS basic operations
  - **Resource allocation** – the number and types of hardware components
  - **Scheduling** – assigning operations to time slots (clock cycles)
  - **Module Binding** – assigning operations to allocated hardware components
  - **Controller Synthesis** – design on control style and clocking scheme

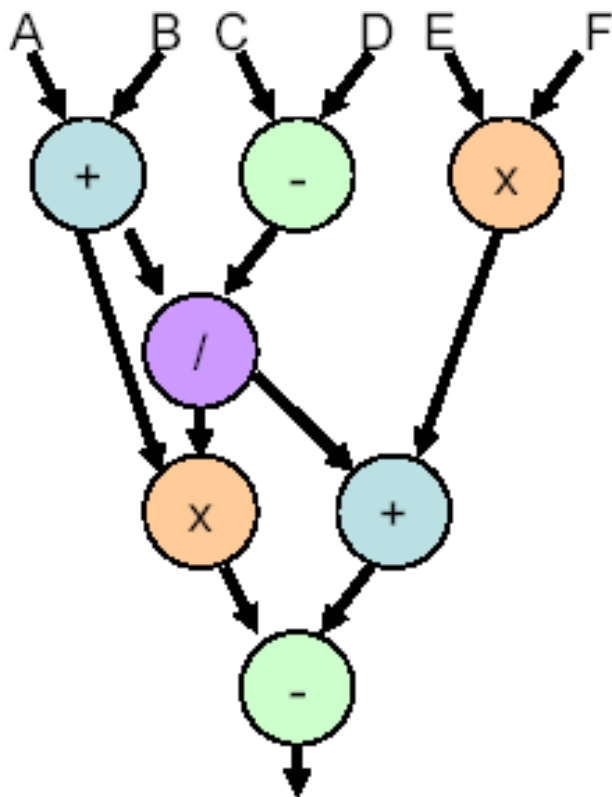




# DATA-FLOW Model of Computation



- Data-Flow graphs (DFGs) or Data-Dependency Graphs (DDGs)
  - Represent parallelism in computation and precedence of operations
  - Nodes: represent computations
  - Edges: represent precedence relations



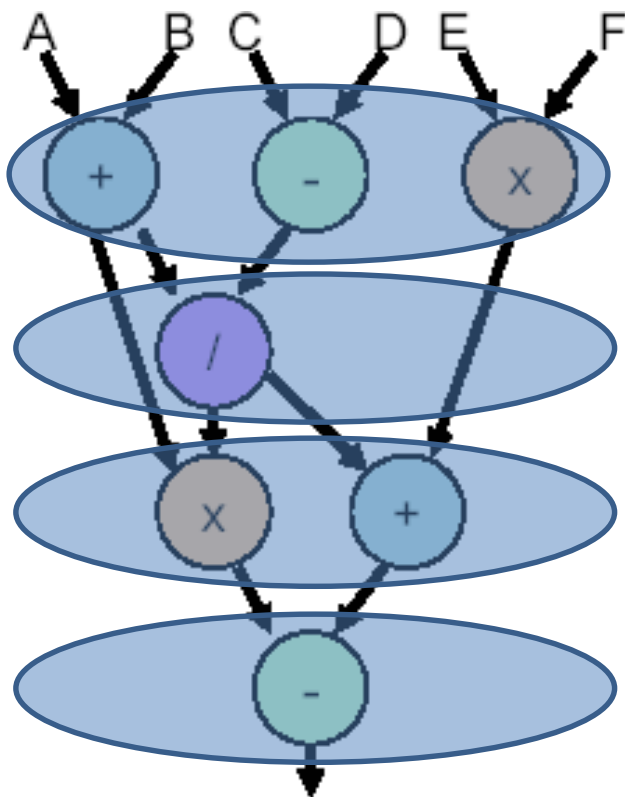
Data-Flow Graph – Example
Sequential Computation
tmp_1 = A+B
tmp_2 = C-D
tmp_3 = E*F
tmp_4=tmp_1/tmp_2
tmp_5=tmp_1*tmp_4
tmp_6=tmp_4+tmp_3
Result=tmp_5-tmp_6



# DATA-FLOW Model of Computation



- Data-Flow graphs (DFGs) or Data-Dependency Graphs (DDGs)
  - Represent parallelism in computation and precedence of operations
  - Nodes: represent computations
  - Edges: represent precedence relations













Data-Flow Graph – Example
Concurrent Computation
$\text{tmp\_1} = A + B, \text{tmp\_2} = C - D, \text{tmp\_3} = E * F$
$\text{tmp\_4} = \text{tmp\_1} / \text{tmp\_2}$
$\text{tmp\_5} = \text{tmp\_1} * \text{tmp\_4}, \text{tmp\_6} = \text{tmp\_4} + \text{tmp\_3}$
$\text{Result} = \text{tmp\_5} - \text{tmp\_6}$





- Behavioral optimization
  - no knowledge about the circuit implementation is required

Tree Height Reduction	
$x = a + b + c + d;$	can be split
$x = a + b; \quad x = x + c; \quad x = x + d;$	→ three additions in series.
$p = a + b; \quad q = c + d; \quad x = p + q;$	→ the first two additions can be done in parallel
Constant Propagation	Variable Propagation
$a = 0; b = a + 1; c = 2 \cdot b$ 	$a = x; b = a + 1; c = 2 \cdot a$ 
$a = 0; b = 1; c = 2$ 	$a = x; b = x + 1; c = 2 \cdot x;$ 
Operator strength reduction	Dead code elimination
$b = 3 \cdot x;$ 	<del><math>a = x;</math></del> $b = x + 1; c = 2 \cdot x;$
$t = x \ll 1; b = x + t;$ 	Removed if not referenced in subsequent code
Code motion	Common Sub-expression elimination
$\text{for } (i = 1; i \leq a \cdot b) \{ \}$ 	$a = x + y; b = a + 1; c = x + y;$ 
$t = a \cdot b; \text{for } (i = 1; i \leq t) \{ \}$ 	$a = x + y; b = a + 1; c = a$ 



# HLS – Example 1



- Consider the following program [1]:

repeat

$x_l = x + dx;$

$u_l = u - (3 * x * u * dx) - (3 * y * dx);$

$y_l = y + u * dx;$

$c = x_l < a;$

$x = x_l; u = u_l; y = y_l;$

until (c);

$x_l = x + dx$

$u_l = u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx)$

$y_l = y + u \cdot dx$

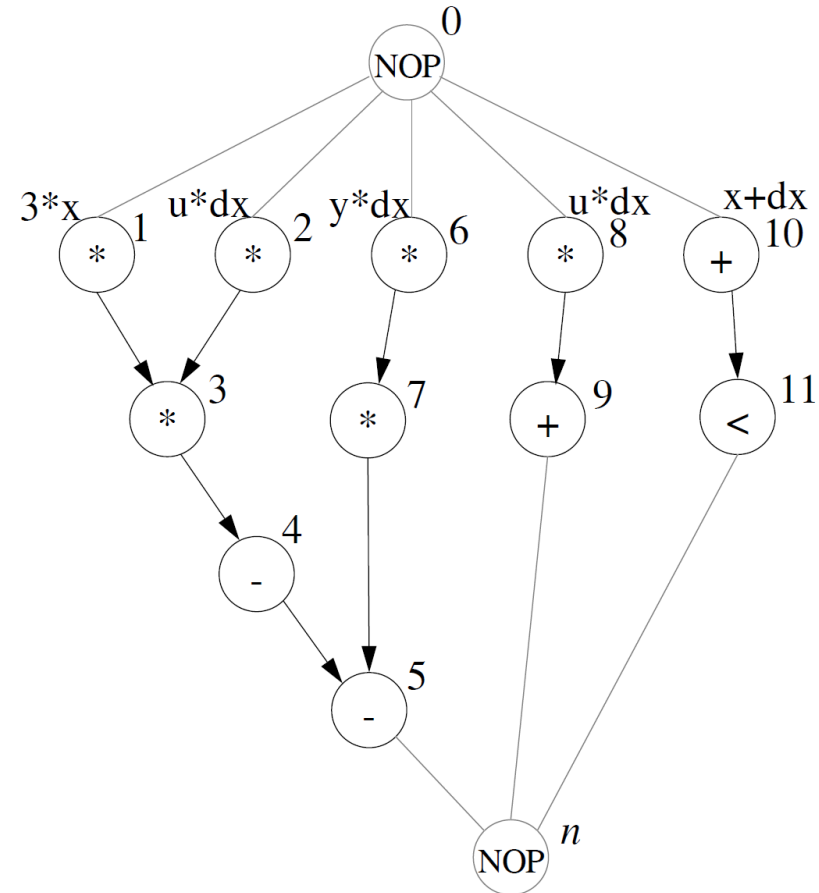
$c = x < a$

v10

v1-v7

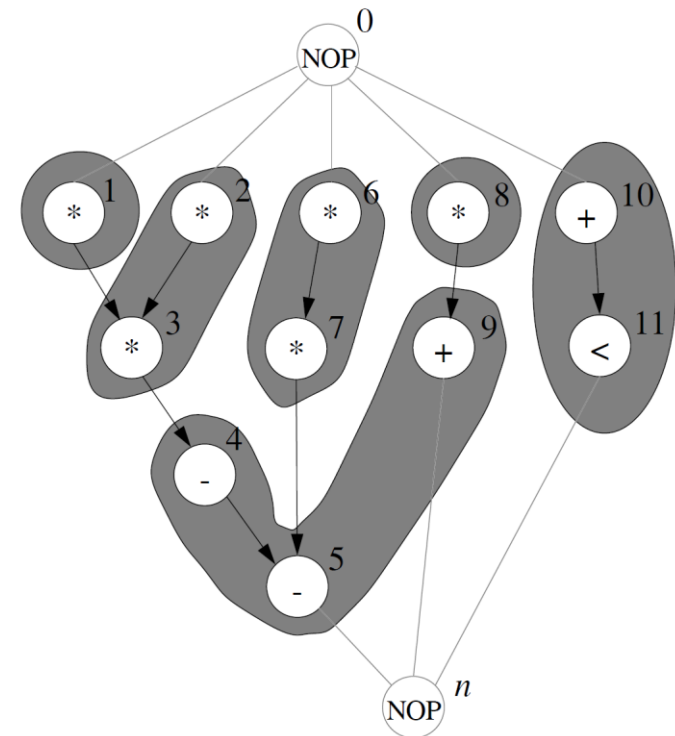
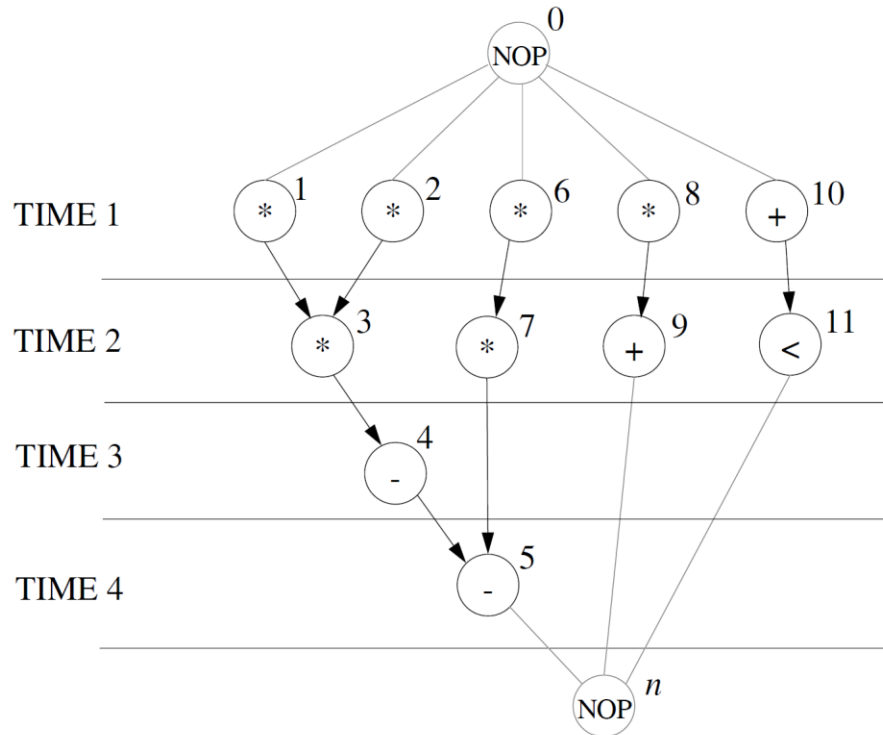
v8, v9

v11





# Scheduling and Binding



Sequencing Graph – 6 Multipliers, 5 ALUs

Allocation and Binding – 4 Multipliers, 2 ALUs

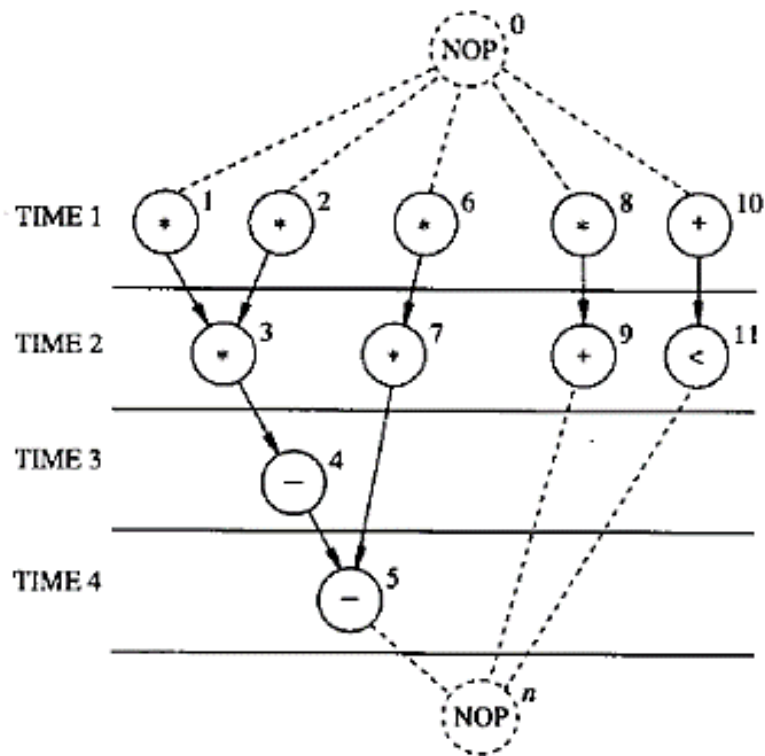
Scheduling	Associate a start time to each operation
Resources Sharing	A component may be used for several operations → sequential scheduling
Binding	Relates operations to available resources Specifies which resource implements an operation



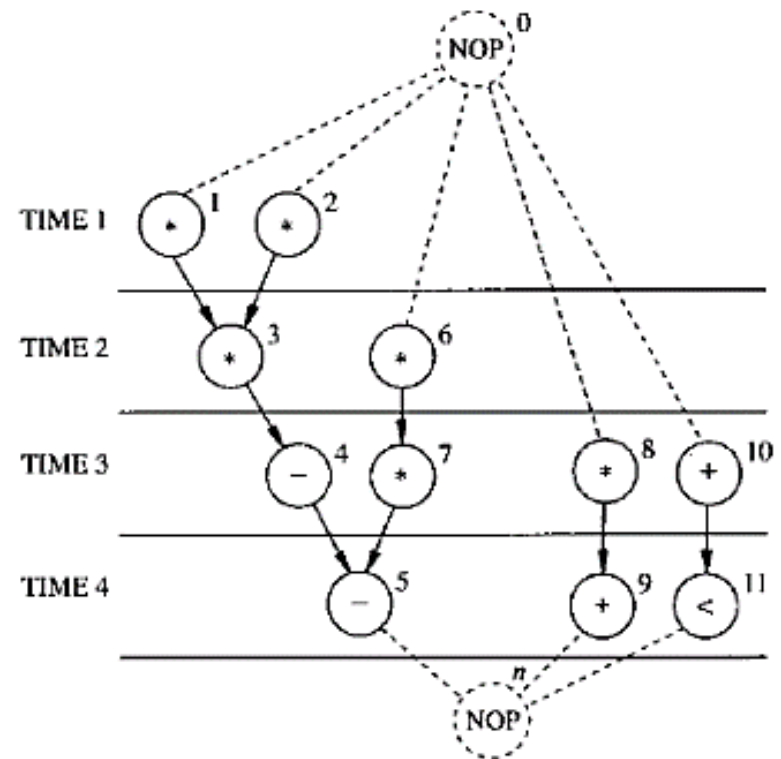
# Scheduling Without Resource Constraints



- **ASAP** – as soon as possible: the start time for each operation is the minimum allowed by all dependencies; yields the minimum start-time values
- **ALAP** – as late as possible, provides the maximum corresponding values



ASAP Scheduling



ALAP Scheduling



- **Mobility** defines the start time span of an operation
  - Use ASAP and ALAP scheduling
  - The difference in scheduling determines the mobility of the operations
  - Mobility can be exploited for a more efficient scheduling
- **Zero Mobility**
  - implies that an operation can be started only at the given time step in order to meet the overall latency constraints. When the mobility is larger than zero, then it measures the span of the time interval in which it may be started
- **Mobility example:**
  - Operations 1 – 5: mobility = 0
  - Operations 6 – 7: mobility = 1
  - Operations 8 – 11: mobility = 2



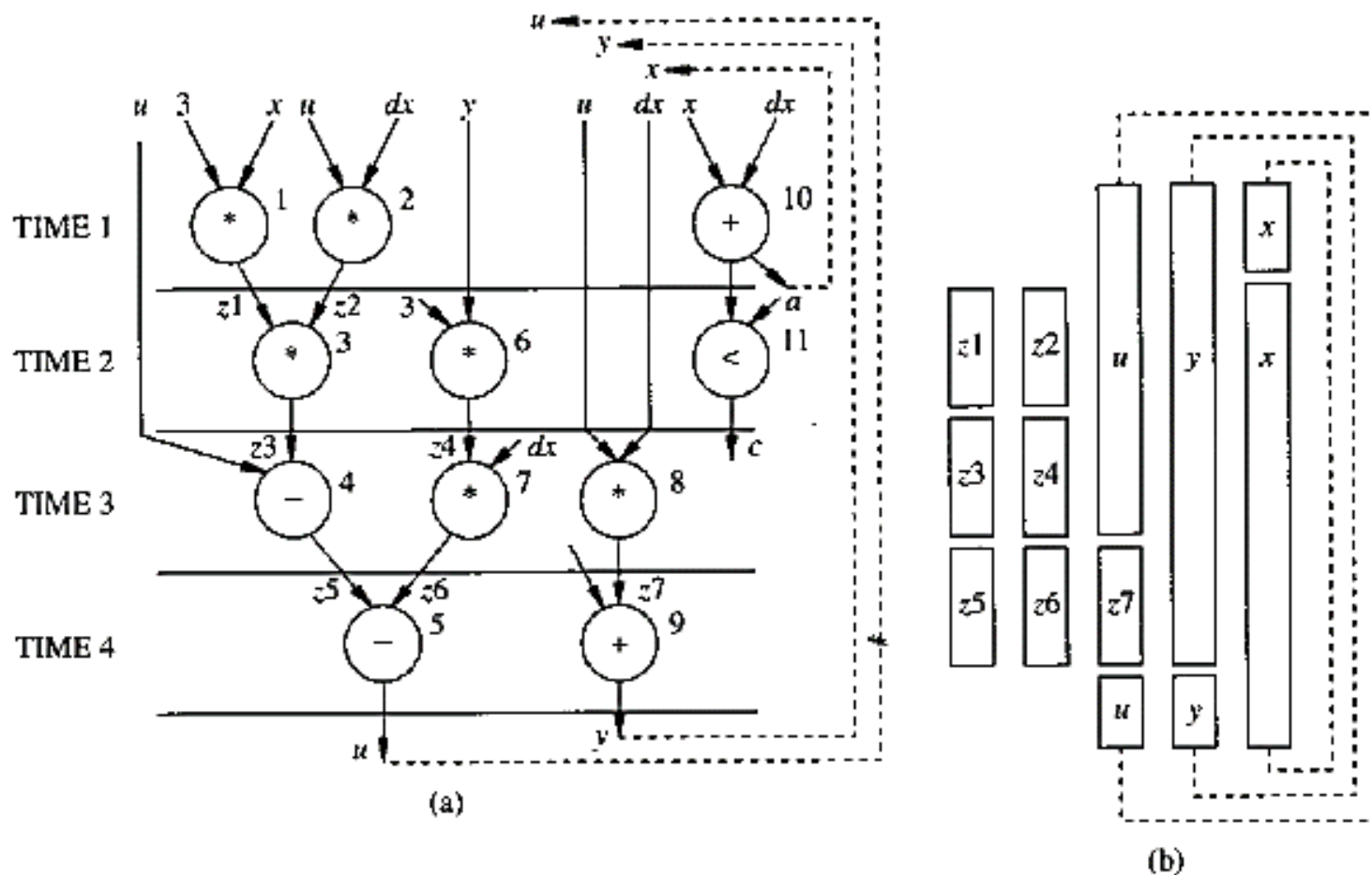
- Hardware models:
  - Functional Units
  - Registers
  - Register-File
  - Multiplexers
  - Tri-state buffers
  - Buses
- Parameters defining the hardware model:
  - Clocking strategy: single or multiple phase clocks
  - Interconnect: MUX and/or BUS based
  - Clocking of functional units
    - Single-cycle
    - Multi-cycle
    - Chaining
    - Pipeline



- **Data-Path synthesis**
  - Generate structural data-path realization from scheduled DFG
  - Two steps: allocation and binding
- **Allocation** – chooses FUs and registers
  - select the one that best matches the design constraints
  - Example: If area is more important than speed, adder is implemented using ripple-carry, if speed is more important than area, adder is implemented using look-ahead
- **Binding** – assigns operations to FUs, variables to registers
  - the aim is to connect FUs such that the cost of interconnection (number of mux, BUS) is minimized
- **Register-Binding Algorithm**
  - Lifetime of a variable – number of cycle times in which that variable is alive
  - Analyses the lifetimes of all variables
  - Establish the required number of registers



# Register Allocation and Binding



**Variable lifetime  $\rightarrow$  register allocation**

a. Sequencing graph, b. Variable lifetimes  $\rightarrow$  5 registers required

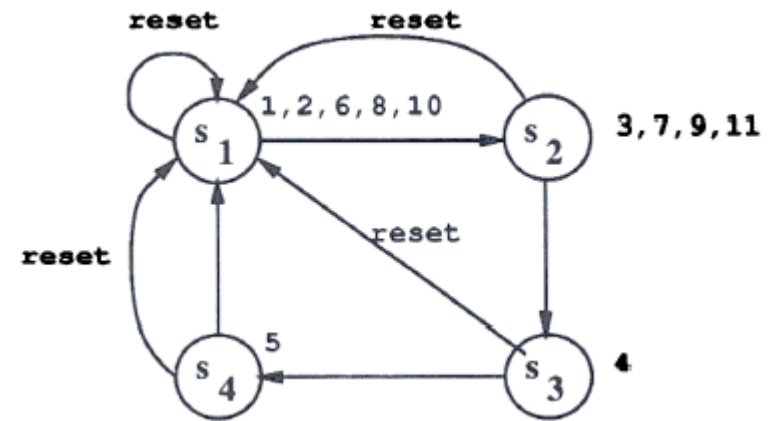
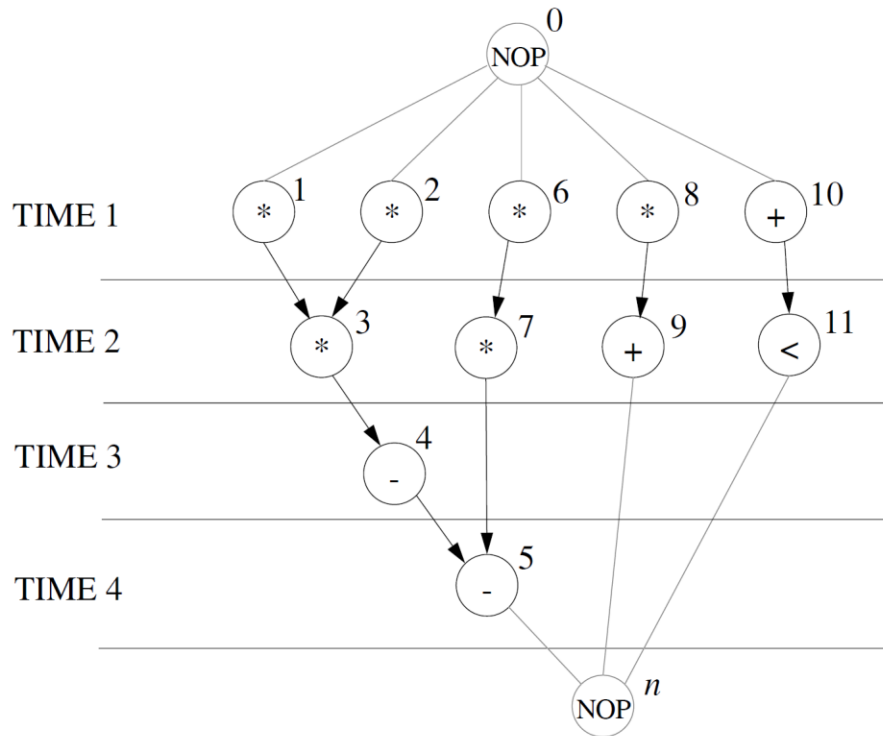




# Control Unit Synthesis



- The control flow described by the schedule is implemented using a state machine, one state for each control step
- The control signals generated from each state activates the Functional Units, Registers, MUX, BUS selects, etc.

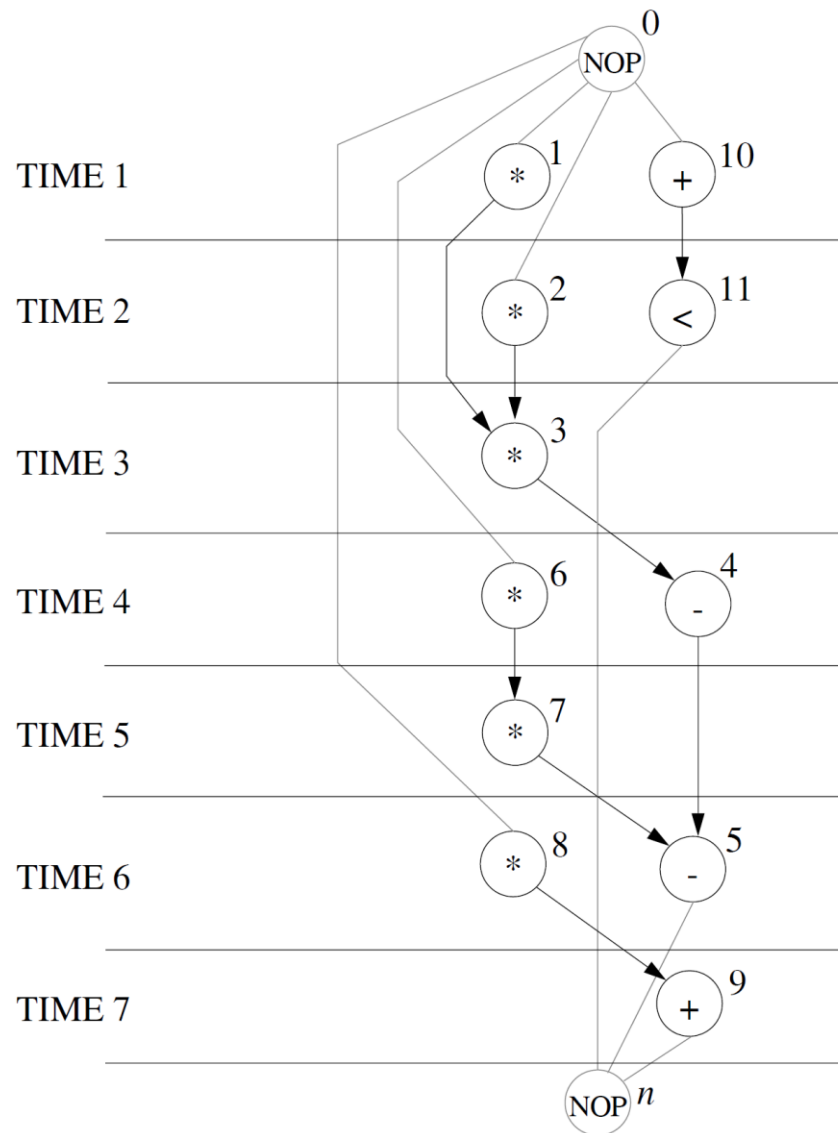


ASAP Scheduling FSM state transition diagram

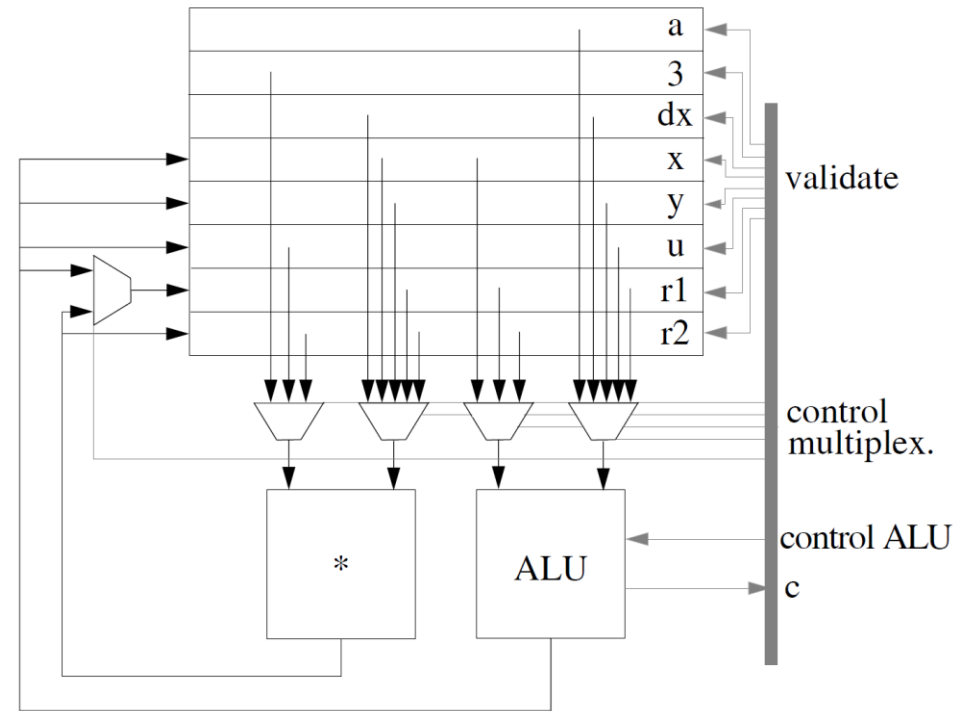
The numbers by the vertices of the diagram are the reference to the activation signals



# Scheduling With Resource Constraints



## 1 Multiplier and 1 ALU



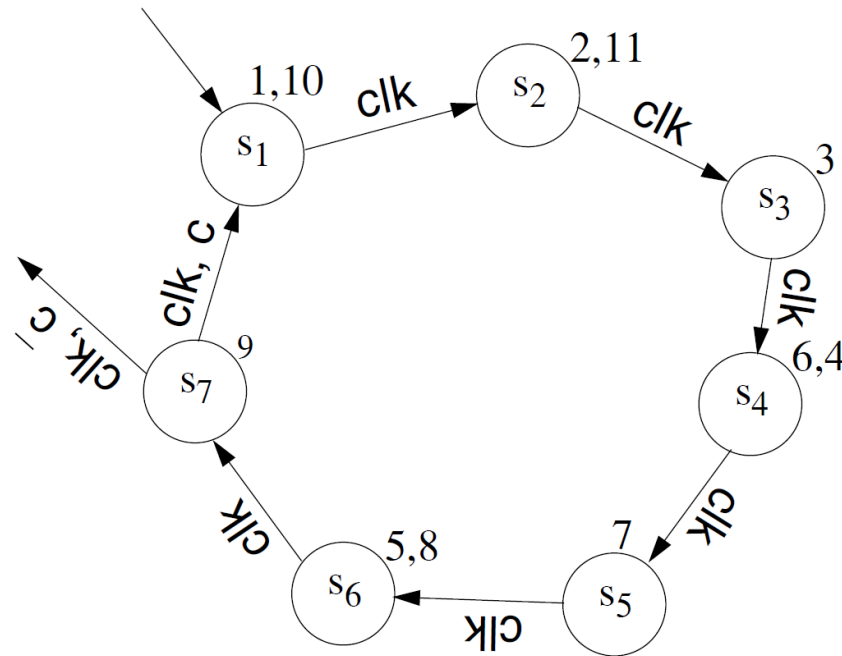
MUX-based data-path



# Scheduling With Resource Constraints



- 1 Multiplier and 1 ALU
  - One state for each clock cycle.
  - In each state the generated signals activate the needed operations



FSM state transition diagram



# Synthesis of Pipeline Circuits



- **Pipelining** – a common technique for enhancing the performance of a digital circuit or processor
- The circuit is partitioned in a linear array of stages each concurrently executing a task on a different set of data and feeding its results to the following stage
- Pipelining increases **throughput**.
- No resource constraints
- Operations have unit execution delays



# Design Constraints and Optimizations



- **Time** → speed; latency, throughput, clock frequency
- **Space** → cost; multilevel parallel combinational logic, sequential component sharing, pipelining
- **Power** → battery life; sleep modes
- **Testability** → crash; BIST (Built In Self Test)
- **Design styles:**
  - **Latency** (response delay time) optimization: single-cycle, multilevel parallel combinational logic data-path, no resource constraints, and slow clock
  - **Throughput** (frequency of result generation) optimization: pipelined data path, fast clock.
  - **Space** (resource) optimization: sharing, reuse of components, multi-cycle data-path.
  - **Space** (communication) optimization: MUX-based or 1, 2, 3 BUS multi-cycle data-path.



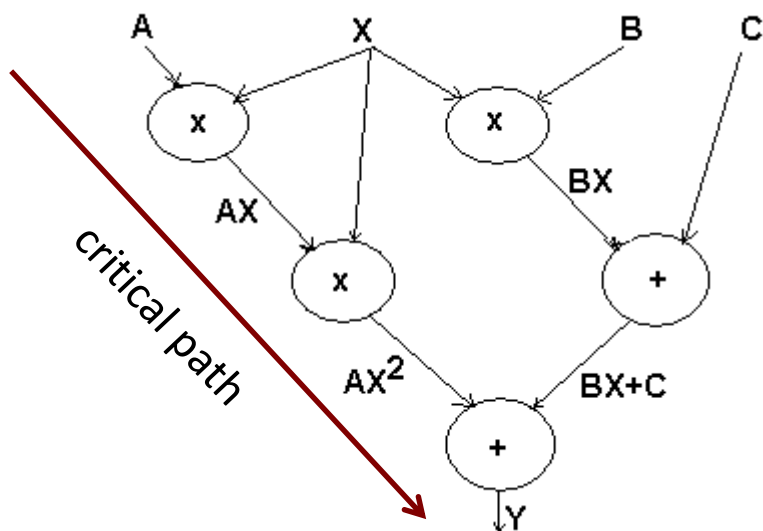
- **Single-cycle design**
  - Design for: optimal latency.
  - Resources: combinational functional units (FU), no resource constraints.
  - Communication: direct, point-to-point connections between FUs.
  - Clock frequency: defined by the critical path, the longest composed delay of the serial connected FUs
- **Pipeline design**
  - Design for: optimal throughput.
  - Resources: combinational FUs, inter stage registers, possible resource constraints.
  - Communication: through inter stage pipeline registers. The combinational FUs are isolated through inter stage pipeline registers and can work in parallel on different execution phases.
  - Clock frequency: defined by the slowest functional component and register overhead.
- **Multi-cycle design**
  - Design for: high clock frequency with spatial constraints. Resource sharing – reuse.
  - Resources: limited number of combinational FUs, temporary registers.
  - Communication: through temporary registers, MUX-es and BUS-es. The intermediate results of combinational FUs are registered for further use.
  - Clock frequency: defined by the slowest functional component and register overhead.



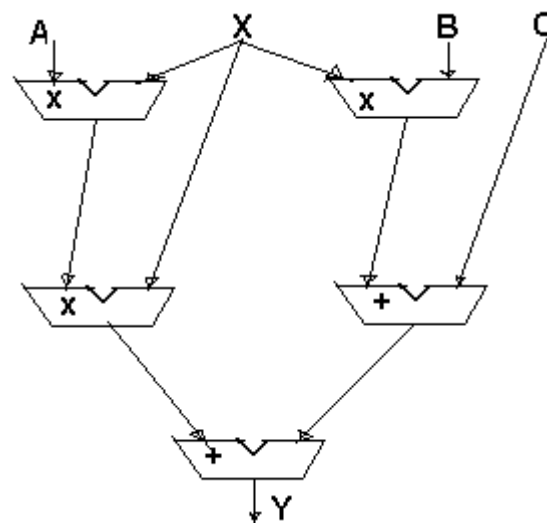
## Problem: $Y = A \cdot X^2 + B \cdot X + C$



- Design of application specific FUs, based on standard components
- **Solution 1 – Single-cycle**
- We have to rewrite the equation in terms of standard, 2 inputs, 1 - output operations:  $A \cdot X$ ,  $B \cdot X$ ,  $(A \cdot X) \cdot X$ ,  $(B \cdot X) + C$ ,  $((A \cdot X) \cdot X + (B \cdot X) + C)$
- The DFG shows the data dependencies and precedence of operations



DFG:  $Y = A \cdot X^2 + B \cdot X + C$



Data-Path:  $Y = A \cdot X^2 + B \cdot X + C$

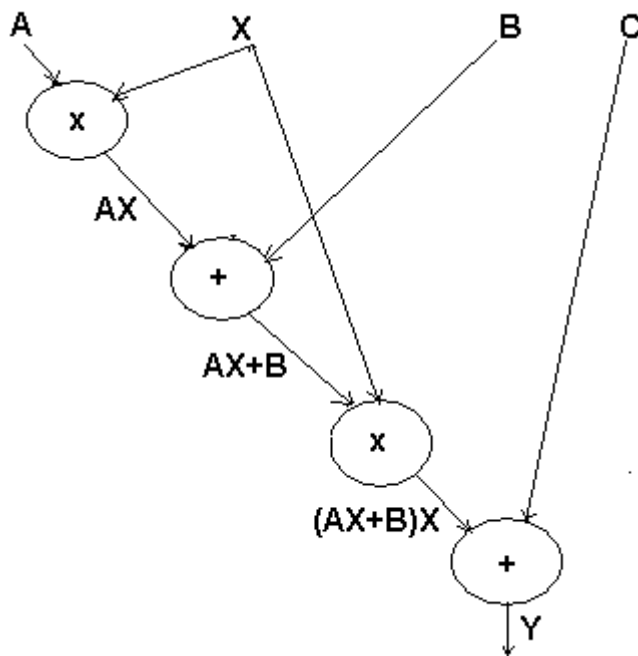
- Resources: 3 Multipliers, 2 ALUs. Critical Path defines the response time ( $2 \cdot x + 1 \cdot +$ )
- Data Flow driven execution: NO conditions, NO control signals



## Problem: $Y = A \cdot X^2 + B \cdot X + C$



- Solution 2 – Single-cycle
- Rewrite the equation:  $Y = (A \cdot X + B) \cdot X + C$



DFG:  $Y = (A \cdot X + B) \cdot X + C$

- Resources: 2 Multipliers, 2 ALUs. Critical Path defines the response time ( $2 \cdot x + 2 \cdot +$ )
- Data Flow driven execution: NO conditions, NO control signals
- 2 solutions with different response times and resource utilization

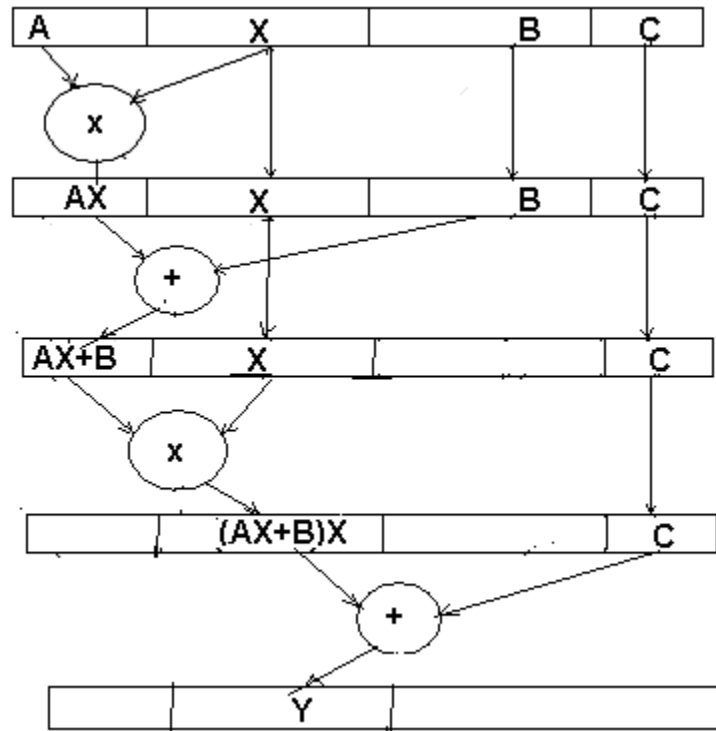




## Problem: $Y = A \cdot X^2 + B \cdot X + C$



### • Solution 3 – Pipeline (based on the second single-cycle solution)



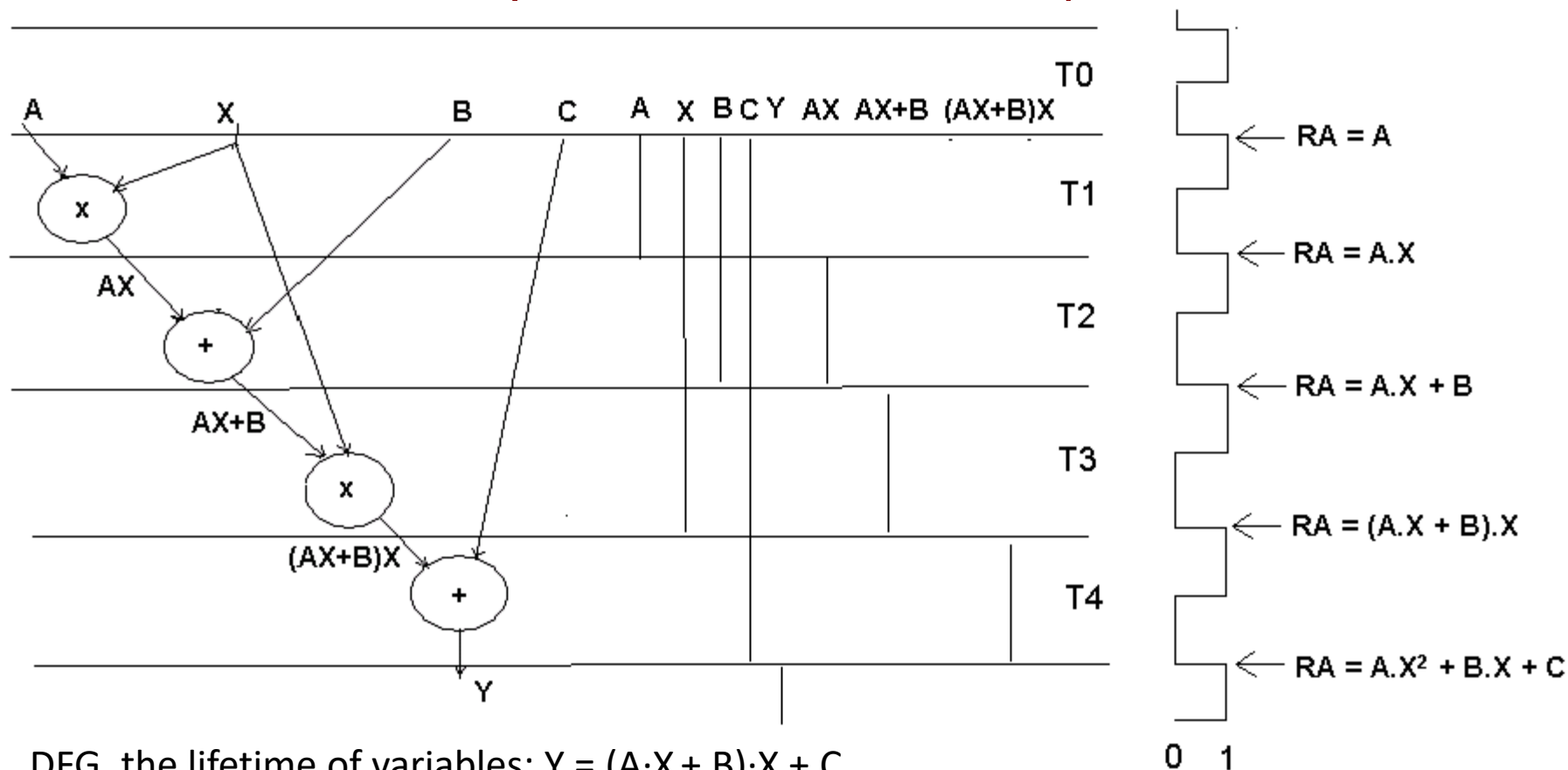
- Suppose the Adder and Multiplier delays are equal
- Introduce 4 pipeline registers after every combinational circuit  
→ 4 balanced pipeline stages
- At every clock cycle a new set of variables A, X, B, C can enter the pipeline
- **The pipeline stages are working simultaneously**
- The latency of the circuit (the propagation delay through the complete pipeline) is composed from the delay of the combinational and register components of the stages.
  - Latency > Single-cycle design.
- The pipelined design needs more resources than single-cycle
- After a pipeline filling period, in every clock cycle we obtain a new value of Y.
- **The throughput is 1 result /clock period.**



## Problem: $Y = A \cdot X^2 + B \cdot X + C$



- Solution 4 – Multi-cycle MUX-Based, 1 Multiplier and 1 ALU



- Use 4 temporary registers for the initial A, X, B, C values: RA, RB, RC, RX
- RA can be used as for A, A.X, A.X+B, (A.X+B).X and Y values – 5 different values





## Problem: $Y = A \cdot X^2 + B \cdot X + C$



- Concrete RTL description for  $Y = (A \cdot X + B) \cdot X + C$  and control signals

T0: The initial values of the variables have been loaded

T1:  $RA \leftarrow RA \times RX;$      $S2 \leftarrow X;$   $S1 \leftarrow 2;$     //  $RA = A \cdot X$

T2:  $RA \leftarrow RA + RB;$      $S2 \leftarrow 0;$   $S1 \leftarrow 1;$     //  $RA = A \cdot X + B$

T3:  $RA \leftarrow RA \times RX$      $S2 \leftarrow X;$   $S1 \leftarrow 2;$     //  $RA = (A \cdot X + B) \cdot X$

T4:  $RA \leftarrow RA + RC;$      $S2 \leftarrow 1;$   $S1 \leftarrow 1;$     //  $RA = A \cdot X^2 + B \cdot X + C$

RTL description

MUX control

Tracing

The control unit can be implemented as a 4-state FSM (5 by including T0)

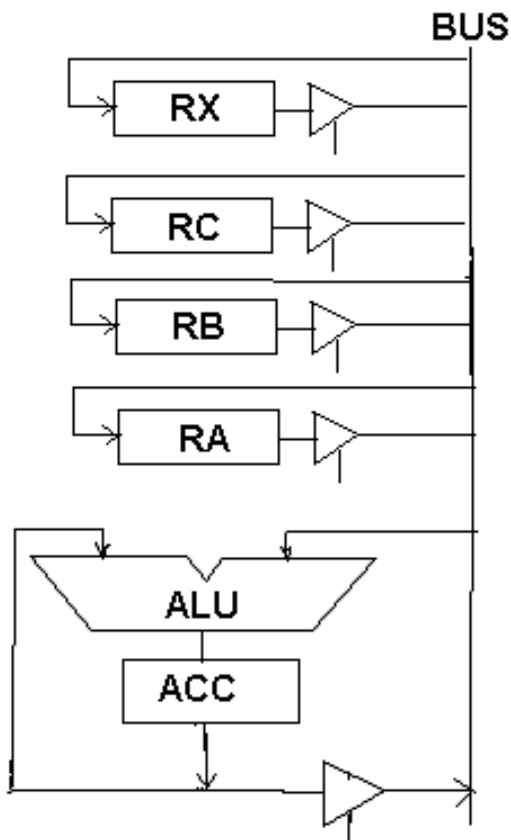
Pipeline vs. Multi-cycle throughput difference:

- Pipeline:** at every clock cycle a new set of variables  $A, X, B, C$  can enter the pipeline and a new result is produced, after the filling time.
- Multi-cycle:** at every 5<sup>th</sup> clock cycle a new set of variables can enter and a result is produced.
- The clock cycles can have the same values.



## Problem: $Y = A \cdot X^2 + B \cdot X + C$

- Solution 5 – Multi-cycle 1-BUS, Temporary ACC register, 1 ALU



Multi-Cycle Data-Path  
1-BUS

T0: Suppose, that the initial values of the variables have been loaded

T1:  $ACC \leftarrow \text{ALUtransfer RA};$     rdRA, wrACC,    // ACC = A  
ALUOp=transfer;

T2:  $ACC \leftarrow ACC \times RX;$     rdRX, wrACC,    // ACC = A · X  
ALUOp= \*;

T3:  $ACC \leftarrow ACC + RB;$     rdRB, wrACC,    // ACC = A · X + B  
ALUOp= +;

T4:  $ACC \leftarrow ACC \times RX;$     rdRX, wrACC,    // ACC = (A · X + B) · X  
ALUOp= \*;

T5:  $ACC \leftarrow ACC + RC;$     rdRC, wrACC,    // ACC = A · X<sup>2</sup> + B · X + C  
ALUOp= +;

RTL description

Control Definition

Tracing

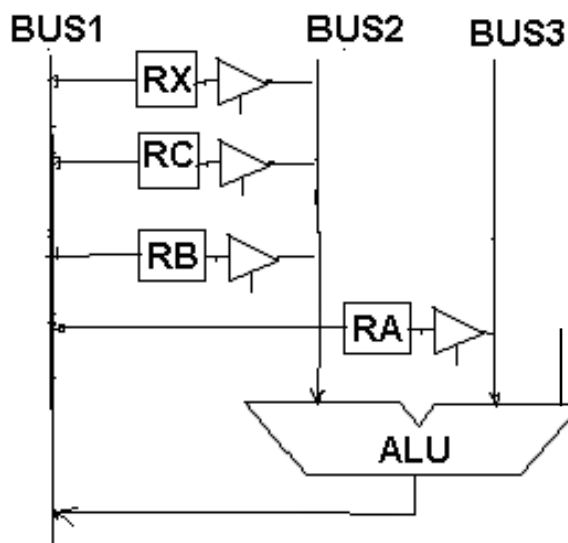
Tri-state buffers for all writes to the bus.

The Control Unit can be implemented as a 5 state FSM.



## Problem: $Y = A \cdot X^2 + B \cdot X + C$

- Solution 5 – Multi-cycle 3-BUS, 1 ALU



T0: Suppose, that the initial values of the variables have been loaded

T1:  $RA \leftarrow RA \times RX;$  | rdRA, rdRX, ALUop= \*, wrRA; | //  $RA = A \cdot X$

T2:  $RA \leftarrow RA + RB;$  | rdRA, rdRB, ALUop= +, wrRA; | //  $RA = A \cdot X + B$

T3:  $RA \leftarrow RA \times RX;$  | rdRA, rdRX, ALUop= \*, wrRA; | //  $RA = (A \cdot X + B) \cdot X$

T4:  $RA \leftarrow RA + RC;$  | rdRA, rdRC, ALUop= +, wrRA | //  $RA = A \cdot X^2 + B \cdot X + C$

RTL description

Control Definition

Tracing

Multi-Cycle Data-Path  
3-BUS

BUS3 can be used for initial loading of the registers, through the ALU  
The Control Unit can be implemented as a 4 state FSM.



# Problems – Homework



- For the equation:  $a \cdot x^2 + b \cdot x + c$ , draw the dataflow graph for:
  - a. ASAP, ALAP scheduled execution
  - b. Pipelined execution
    - Compare the latencies, throughputs and costs of the solutions.
  - a. Implement the multi-cycle 2-bus based solution
- For the equation:  $w = a \cdot (b \cdot x + z) - c \cdot (a \cdot y + x)$  assume:
  - a. Implementation without constraints.
  - b. Implementation with 1 Add/Subtract and 1 Multiplier unit; Latencies = 1 clk.
    - For the implementations a and b show the scheduled DFGs, the lifetime of the variables, the minimal number of necessary registers and the corresponding data paths for a multi-cycle design.



1. P. Eles, Z. Peng, “System Synthesis of Digital Systems”, Lecture slides, March 2000,  
<http://www.ida.liu.se/~petel/SysSyn/lect1.frm.pdf>
2. P. Coussy, D.Gajski, M.Meredith, and A.Takach, „An introduction to high-level synthesis”, *IEEE Design Test of Computers*, 26(4):8 – 17, jul. 2009.
3. Egon Boerger and Robert Staerk, “A Method for High-Level System Design and Analysis”, (Abstract State Machines chapter), Springer-Verlag 2003.