COMPUTER ORGANIZATION AND ARCHITECTURE (IT 2202)

Pipelining-1

Pipelining

What is Pipelining?

- It is an implementation technique, where multiple tasks are executed in an overlapped manner.

When can it be implemented?

- It can be implemented when a task can be divided into two or more subtasks, which can be performed independently. So this is the key idea for implementing pipelining.

Where are pipelining used in a Computer System?

- Pipeline is done in the hardware and it exploits different kinds of parallelism:- instruction level parallelism, Arithmetic parallelism etc.
- Instruction execution: numerous instructions execute in some sequence
- Arithmetic computation: Same operation (Floating/Multiplication) carries out on various data sets
- Memory Operation: Several Memory accesses to consecutive locations can be made.

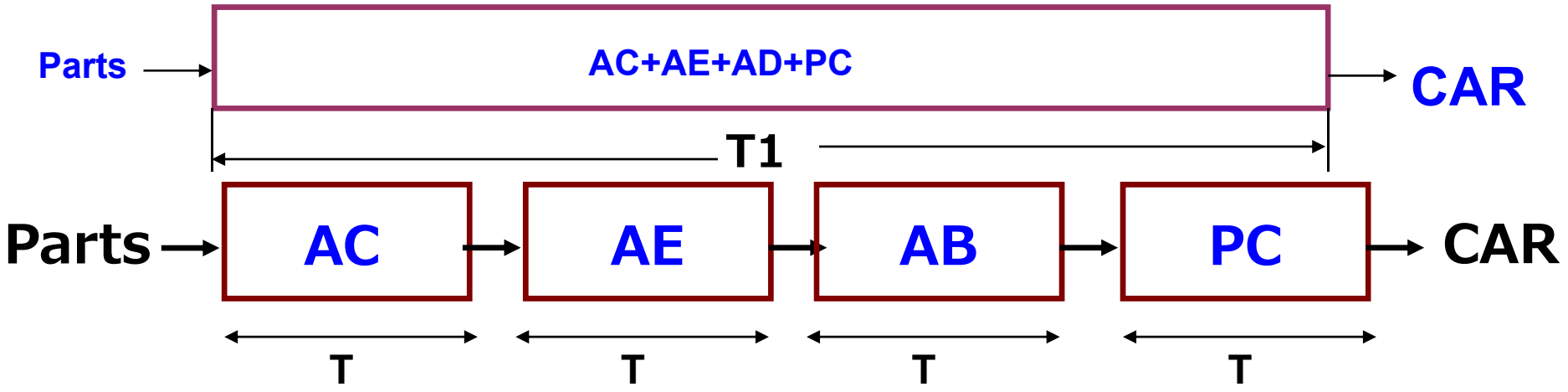
Pipelining

Benefits of Pipelining

- It is used to enhance the performance of a processor with very nominal increase in the cost of Implementation.
- Pipelining makes significant speed up in the processors.

Pipelining: Real Life Example

Suppose, we consider task of building a CAR. This task consists of four subtasks: Assembling Chassis (AC), Attaching Engine (AE), Attaching Body (AD), Painting and Cleaning (PC). All the four subtasks are performed in a single unit and Only one car is worked on at a time.



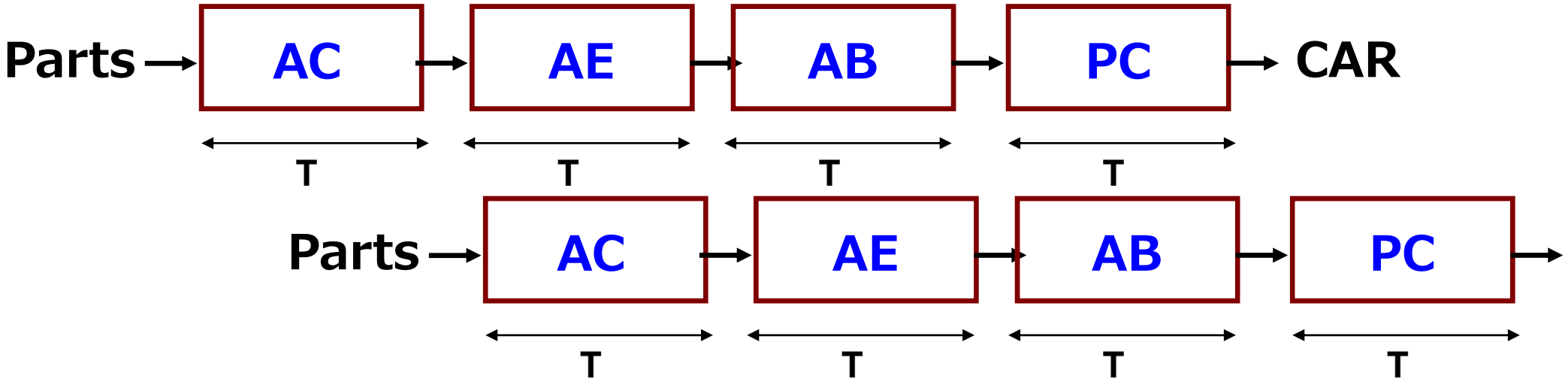
For N cars, Time T1=4T. N = 4N.T

- If each of these subtasks takes T minutes to complete, 4T minutes are needed to assemble each CAR.
- 4NT minutes are needed to assemble N CARs.

This implementation is known as Serial Implementation

Pipelining: Real Life Example

Now, we perform the four subtasks one by one. First Assembling Chassis (AC) will be made in stage 1, Attaching Engine (AE) will be made in stage 2, Attaching Body (AB) will be made in stage 3, and finally Painting and Cleaning (PC) will be done in stage 4.

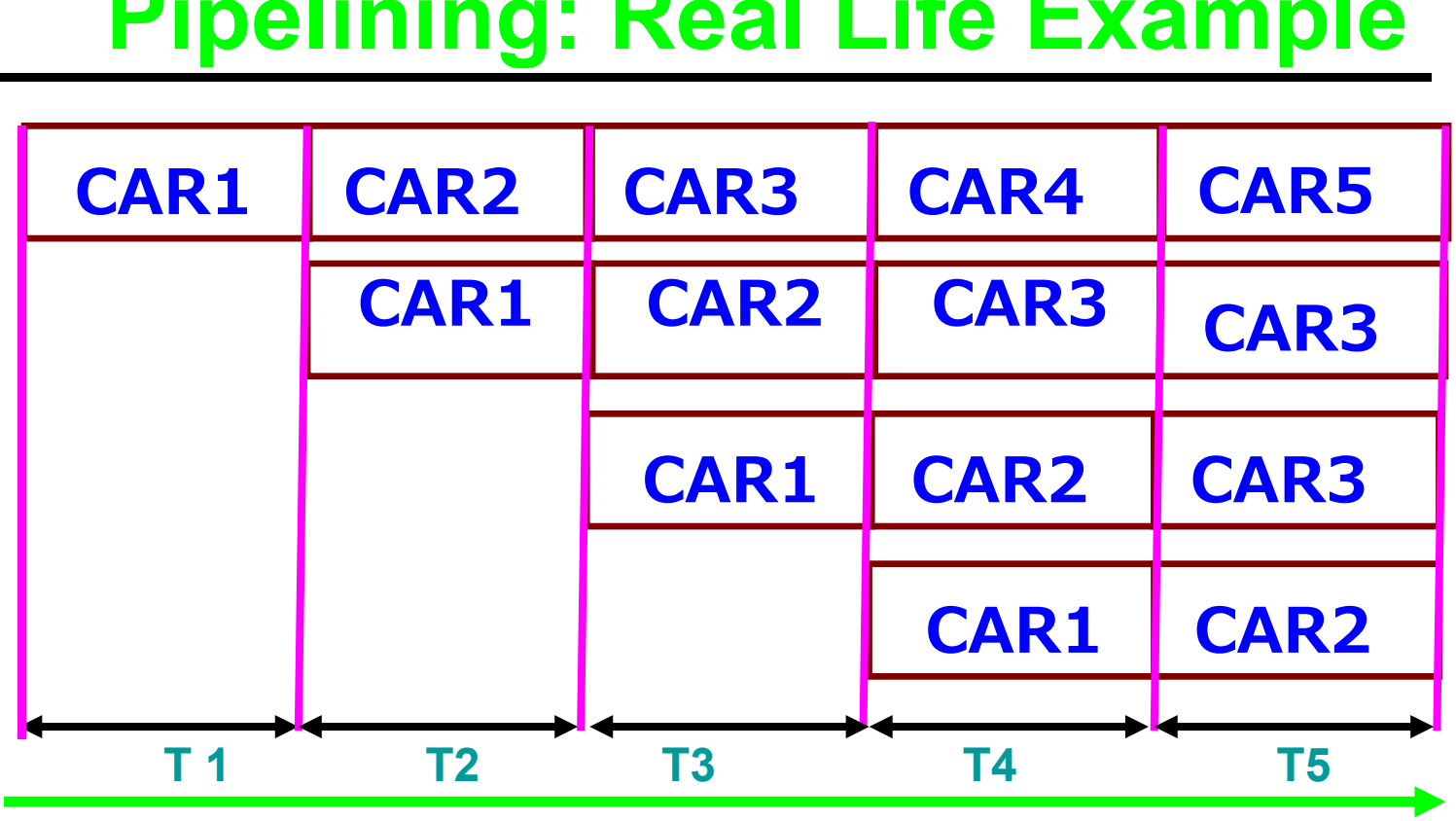


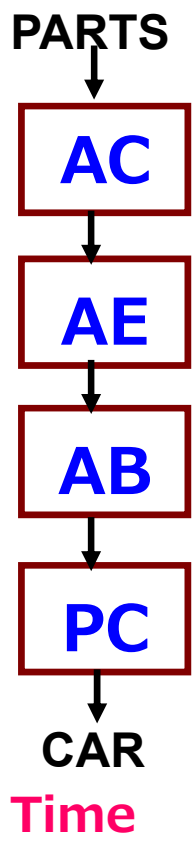
{4T+(N-1) T} time is required to assemble N automobile if each stages takes T unit time.

This implementation is known as pipeline Implementation

Different subtasks are done in the separate unit and each subunit will perform the specific task only. More than one tasks that means, multiple tasks will be performed in overlapped manner. This concept is known as Pipelining.

Pipelining: Real Life Example





- At T1, CAR1 enters into stage 1,
- At T2, CAR1 leaves stage 1 and moves to stage 2, CAR2 enters into stage 1.
- At T3, CAR1 leaves stage 2 and moves to stage 3, CAR2 leaves stage 1 and enters into stage 2.
- At T4, CAR1 leaves stage 3 and moves to stage 4, CAR2 leaves stage 2 and enters into stage 3, CAR3 enters into stage 2, CAR4 enters into stage 1. Finally at T4, CAR1 is built.
- At T5, CAR2 enters into stage 4. Finally, CAR2 is built.

Completion Time

- Time taken to complete Each stage = T
- After T4, CAR1 is manufactured. So CAR1 takes 4T unit time. At T5 Time CAR2 is manufactured, so CAR2 on ward, every car takes T unit time.

- Time required to build (N-1) car = (N-1).T
- Total Time required to assemble N cars

  = 4T+(N-1).T unit

Pipelining: Real Life Example

Computing Speed up

- Time required to assemble N cars in Serial Fashion = 4NT
- Total Time required to assemble N cars in overlapped fashion = 4T+(N-1).T
- Speed up achieved by pipeline over serial is

$$\frac{4NT}{4T+(N-1).T} = \frac{4N}{4+(N-1)} = \frac{4}{4/N+(1-1/N)}$$

For large N, i.e. N>> T, Speed up is

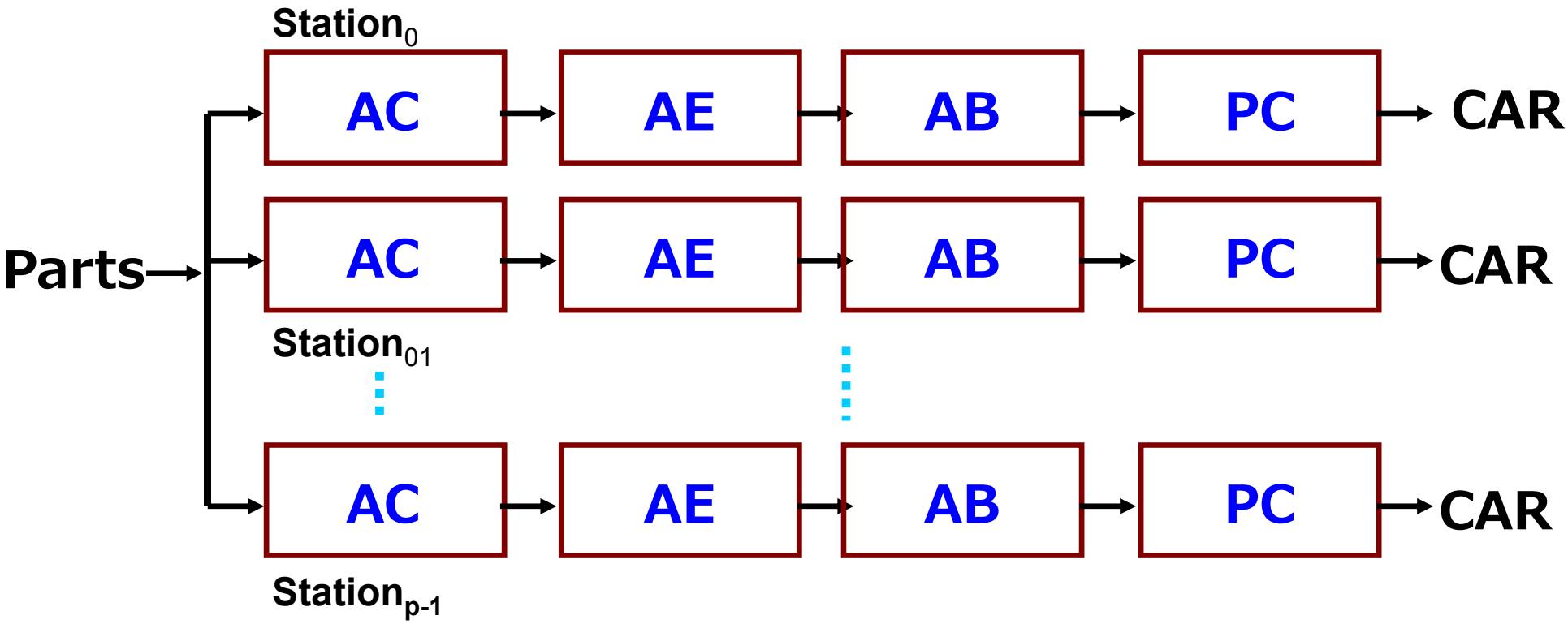
$$\frac{4}{4/N+(1-1/N)} = \frac{4}{4/N+(1-1/N)} \cong 4$$

For large N, Speed up is 4 times that of serial architecture.

An ideal pipeline with P stages provides a speed up of P compared to the serial architecture.

Parallel Implementation

Parallel Architecture: P stations, each capable of performing all four subtasks are used. All the P stations operate simultaneously, each working on a separate car.



It takes 4T minutes to complete one car, but P cars are produced in 4T minutes.

Parallel Implementation

- **To assemble N cars, it takes (4NT)/P.**
- **Speed up achieved by this architecture is P times that of serial architecture.**
- **Hardware complexity has increased by P-fold compared to serial paradigm to achieve P-fold speed up.**
- **Each of P stages could have been implemented as pipeline. It enhances throughput at cost of increased hardware complexity.**

Pipeline in Computer Processor

- **Pipeline and parallel implementation may be extended in processor hardware to achieve performance speed up in the hardware.**

- **In parallel implementation, speed up achieved will be more compared to Serial and pipeline Implementation, but hardware complexity will be significantly higher compared to Serial and pipeline Implementation.**

- **Parallel Implementation enhances throughput at cost of increased hardware complexity.**

- **In case of the pipeline, speed up will be achieved with nominal increase of the hardware overhead.**

Pipeline in Computer Processor

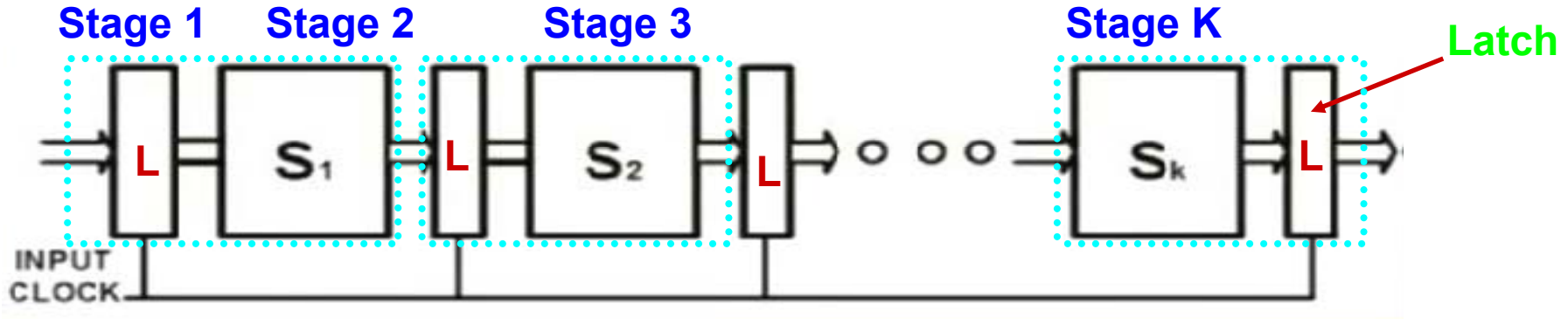
Use of Buffer stage in Pipeline

- In the example of building CAR, the movement of the stages is restricted. When the next stage is free, then only the car can be moved to the next stage from the current stage. So, some kind of intermediate place or stage is needed between the stages to keep the car temporarily before the next stage accepts it. This is called the buffering.
- In case of implement pipeline in hardware, we use a latch or a register between successive stages, because one stage finishes some calculation and prior to giving it to the next stage, may be the next stages still not finished. So, that value is temporarily kept in the latch, so that the next stage whenever it is free can take it from the latch.

Basic Concepts

How is Pipelining implemented?

- Different subtasks are performed by the different computational hardware blocks known as stages. Here, S_1, S_2,⋯, S_K are stages in pipeline processor of K-stages.
- Result produced by each stage is temporarily buffered in latches and passed on to the next stage.



- Latches (L) are made with master slave flip-flops and provide buffering between the stages.
- Stages are basically Combinational circuits.
- On arrival of the clock signal, all the latches transfer data to the next stage simultaneously.

Basic Concepts

Pipeline Processors

Pipelined processors can be categorized based on following parameters ...

- **Degree of overlap**
  - **Serial, overlapped or pipelined**
- **Depth of the pipelining**
  - **Shallow or Deep pipeline**
- **Structure of the pipeline**
  - **Linear or Non-linear**
- **Schedule of the operations in the Pipeline**
  - **Static or Dynamic Pipelining**

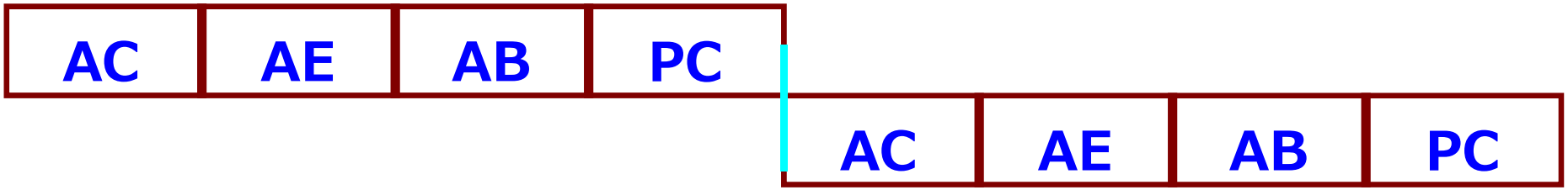
Basic Concepts

Degree of overlap

Degree of overlap is important parameters for pipeline processors.

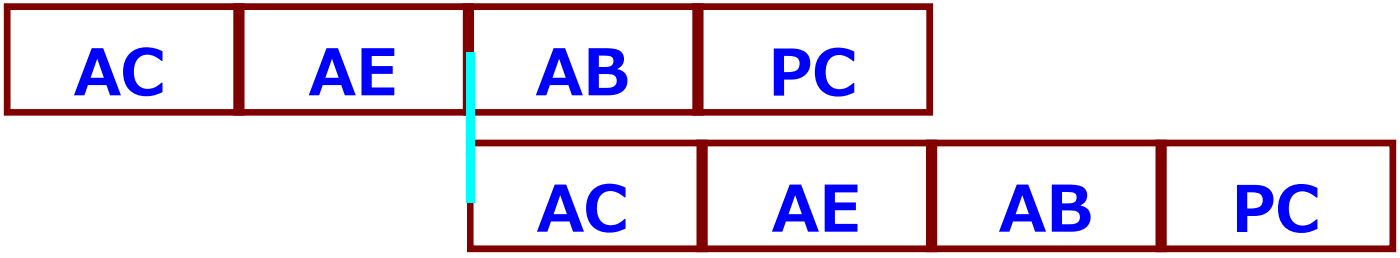
Serial

- **It is a strictly non-pipeline implementation. The next operation can start only after the previous operation finishes.**
- **The figure shows four stage operation. The operation takes AC, AE, AB, PC steps. So, only after the first operation is completed only then the second operation can start.**
- **It is strictly sequential, with no overlap.**



Overlapped

The diagram shows the partial overlap operation. Partial overlap results in a speedup because it is taking 2 time units less.

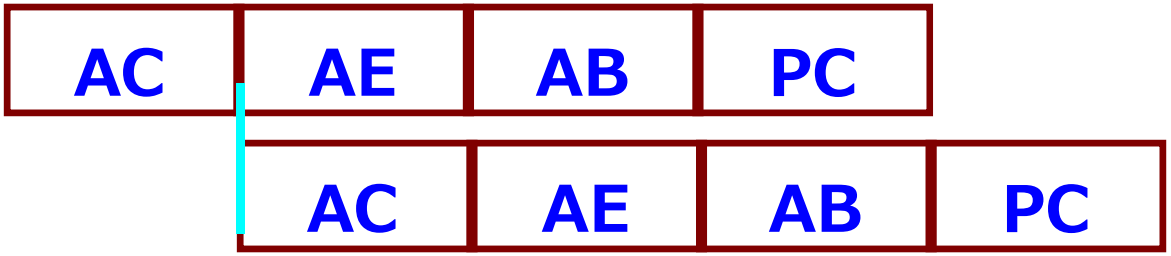


Basic Concepts Degree of overlap

Pipelined

Pipelining provides the complete overlap; when the first operation is finished with the first step (AC), it moves to the second step (AE), and the second operation can start with its first step (AC) as shown in the diagram. This is called fine-grained overlapped execution.

In this case, the time required is much less. So, depending on the degree of overlap, we can classify how deep or how efficient the pipeline implementation is.



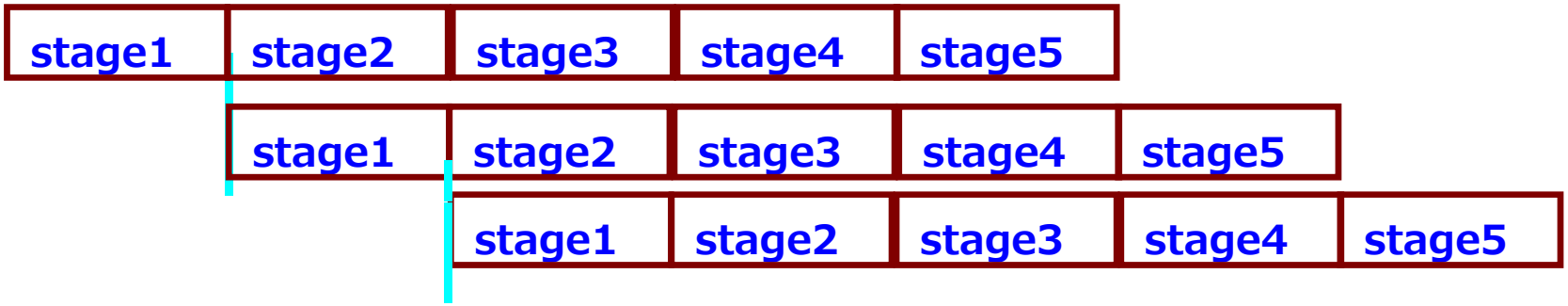
Depth of the pipelining

Pipeline is classified based on the depth i.e., number of stages in the pipeline. There are two parameters related to depth of pipeline : Shallow and Deep pipeline.

Degree of overlap

- If there are k number of stags, then the potential speedup can be approximately k.
- If we take a large value of k, we can get a very large speedup. But due to design issue, there are some limitations to increase the value of k.
- we cannot break up a computation into smaller pieces beyond a limit. Beyond a limit it does not make sense due to the delay of the latch which may be more and will become more than the delay of the computation. So, the depth of the pipeline is an issue.
- There are either a shallow pipeline with a few number of stages, or a deep pipeline with large number of stages. If a pipeline has 9 stages then potentially speedup is 9. Depth of pipeline is a design issue.

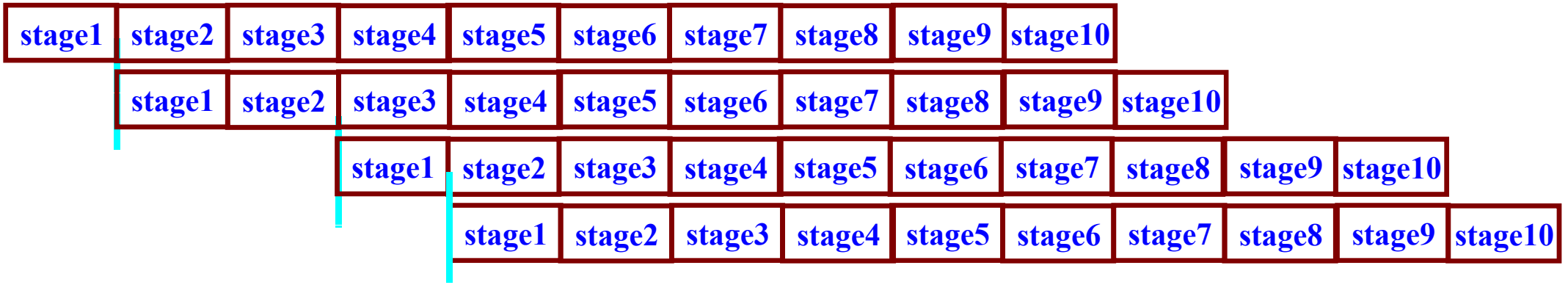
Shallow Pipeline: Shallow pipeline has fewer number of stages. In shallow pipeline, it is made with smaller number of stages smaller, individual stages are more complex. Therefore, the conflict in overlapped execution may occur. The conflicts in a pipeline are very important design issue.



Degree of overlap

Deep pipelining

Deep pipeline has larger number of stages. As it has a very large number of stages, each stage will become simpler. But there is a limit of increasing pipeline stages as due to design constraints, the increased speed up can not be achieved after a certain value. The diagram shows a deep pipeline scheme.



Degree of overlap

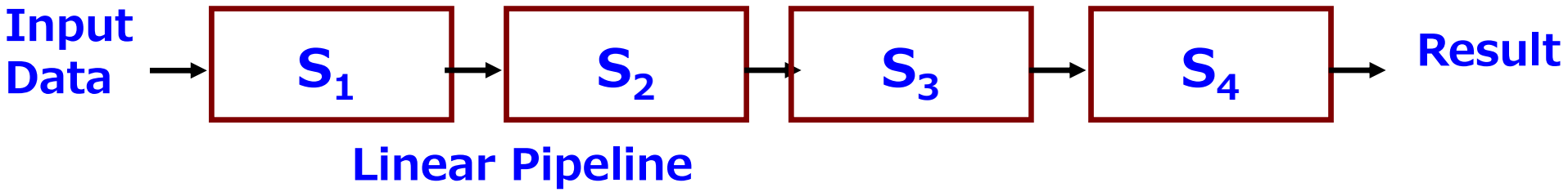
Structure of the pipeline

Pipeline processor is classified as Linear and Non-linear pipeline .

Linear Pipeline: The linear pipeline is conventional and simple. In Linear pipeline, the computation has been divided into a number of stages that are supposed to be executed linearly one after the other.

This operation follows a straight-line sequence. The stages that constitute the pipeline are executed one by one in sequence.

The diagram shows a four stage leaner pipeline in which based on the input data, S1 stage will be executed first, next based on the data output of S1, S2 stage will be executed, then based on the data output of S2, S3 stage will be executed and finally based on the data output of S3, S4 stage will be executed and produce the final result at the output. That means, the operation is done in a sequence. This is the order of execution for every input data.



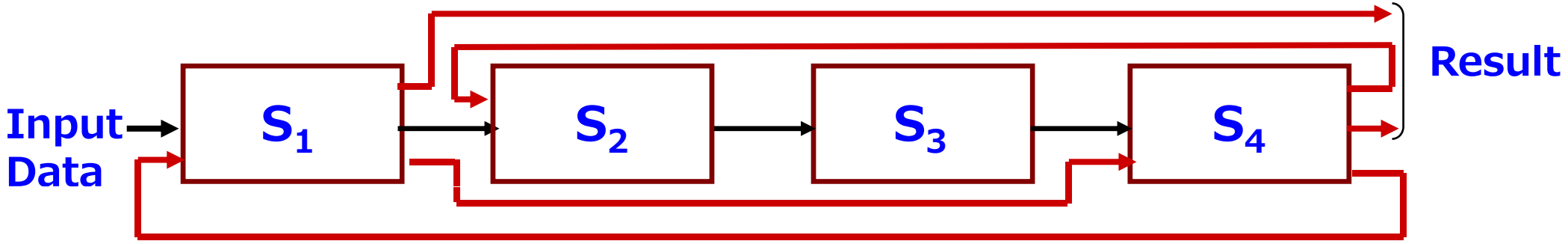
Linear Pipeline

Degree of overlap

Non-linear pipeline: The non-linear pipeline provides complex pipeline implementation. In non-linear pipeline, the stages may not execute in the straight-line or linear sequence. That means, a stage may execute more than once for a given data set.

This diagram shows non-linear pipeline with 4 stages. The latches are not added here. Based on the input data, S1 stage will be executed first, next based on the data output of S1, S2 stage will be executed, then based on the data output of S2, S3 stage will be executed and based on the data output of S3, S4 stage will be executed and output may be taken from S4. Next sequence, based on the data available at S4, S2 stage will be executed and based on the data output of S2, S3 stage will be executed and based on the data output of S3, S4 stage will be executed and output may be taken from S4.

based on the data available at S4, S1 stage will be executed and based on the data output of S1, S4 stage will be executed and based on the data output of S4, S1 stage will be executed and output may be taken from S1. In this example one possible sequence can be (S1, S2, S3, S4, S2,S3, S4,S1,S4,S1). This is so-called non-linear pipeline as stages did not execute in a linear sequence. This example shows that a particular stage executed more than once for a given data set.



Non-linear Pipeline

Basic Concepts

Schedule of the operations in the Pipeline: The pipeline can also be classified based on how the pipeline is scheduled; it can be either static or it can be dynamic.

Static Pipeline

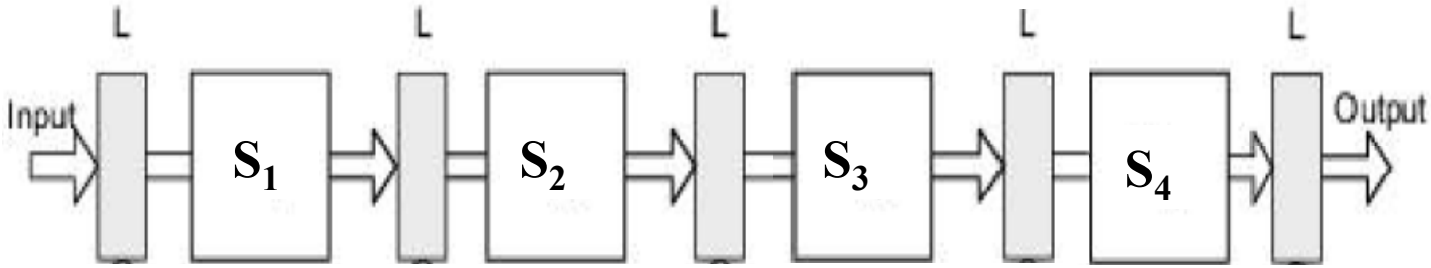
- In static scheduling, same sequence of pipeline stages are executed for all data sets or instructions.
- If one data or instruction stalls, the new data can not be fed. Stall means due to some ongoing computation, the pipeline must be stopped temporarily. If a particular data/instruction stalls , then all the subsequent data sets also get delayed.

Dynamic Pipeline

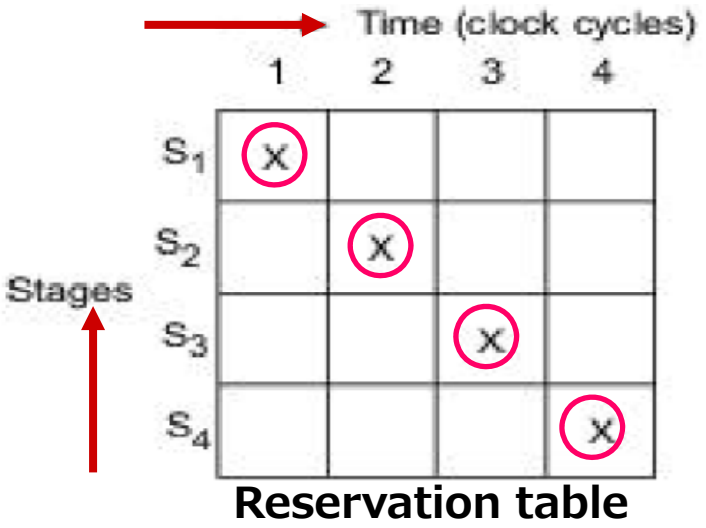
- In a dynamic pipeline, with time, the pipeline can be configured to perform variable functions and different sequence of pipeline stages can get executed.
- This allows feed forward and feedback connections between the stages as shown in non-linear pipeline.

Reservation Table

- Reservation table is a data structure that represents the utilization pattern of successive stages.
- Reservation table is basically a space-time diagram of the pipeline that shows precedence relationships among the pipeline stages.
- X-axis represents the time steps and
- Y-axis represents the stages (Space).
- Number of columns indicates the total time required by the pipeline to evaluate the result.



Four Stage Synchronous Pipeline

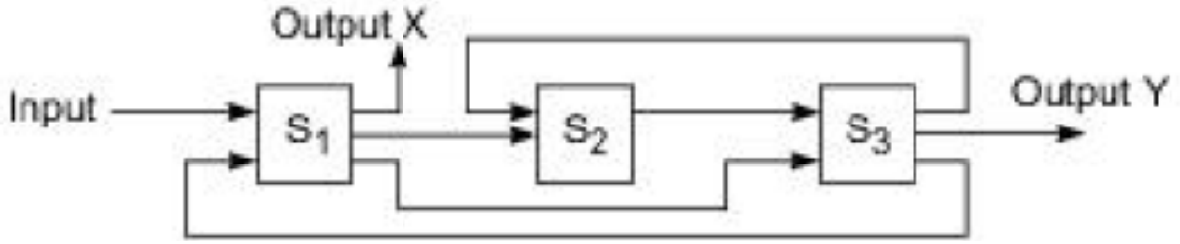


Reservation table

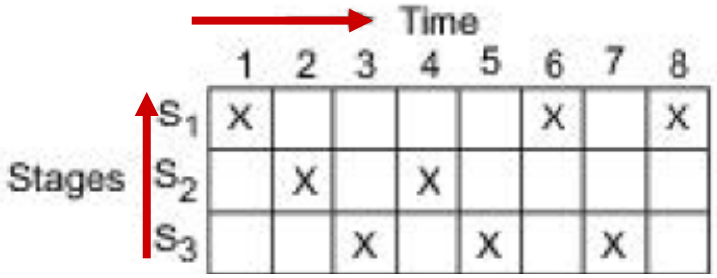
The reservation table for a 4-stage linear pipeline is shown. A linear pipeline with S1 S2 S3 S4 is considered. Input data goes to S1 in 1st cycle, in 2nd cycle it goes to S2, in 3rd cycle to S3, in 4th cycle, it goes to S4. Reservation table represents exactly this operation.

In 1st time step, S1 is used, in 2nd time step S2 is used, in 3rd time step, S3 is used, 4th time step, S4 is used. This is basically a space-time diagram that shows precedence relationship among the pipeline stages. Number of columns indicates the total time required by the pipeline to evaluate the result.

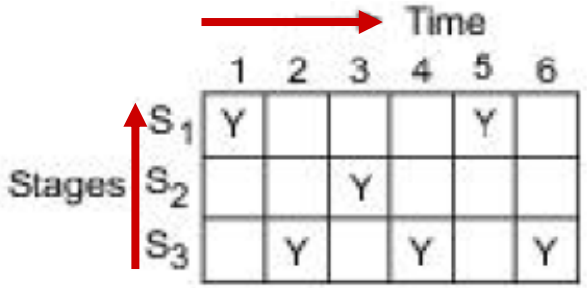
Reservation Table



Three Stage Synchronous Pipeline



Reservation table for function X



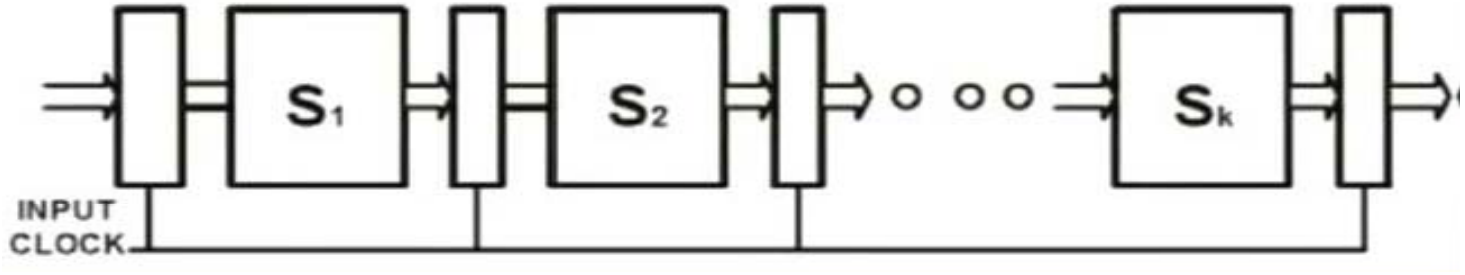
Reservation table for function X

- **Fig shows a Non-linear pipeline which is implementing two function X and Y. The computation of X needs 8 steps: 1st time step S1, 2nd time step S2, 3rd time step S3, 4th time step S2 (from S3 again, it goes to S2), 5th time step again S3, 6th time step S1 (from S3, it goes back to S1), 7th time step again S3. 8th time step, from S3 to S1 again and it is finished.**

- **The computation of Y needs 6 steps: 1st time step S1, 2nd step time S3, 3rd time step S2, 4th time step S3, 5th time step S1, 6th time step S3 and output come out from S3.**

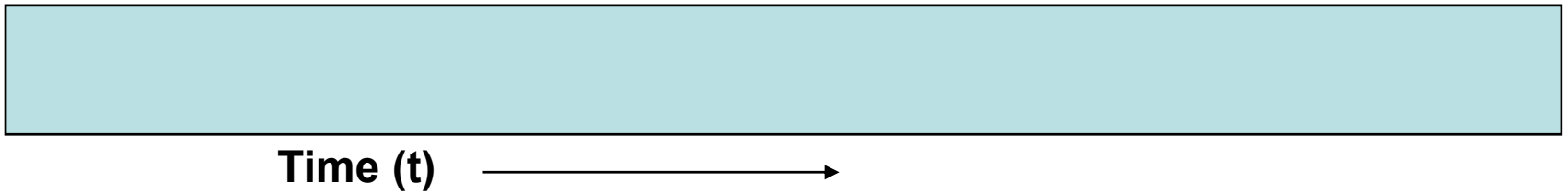
Linear Pipeline Processor

- **Linear Pipeline Processor is constructed with k processing stages, each stage performs a subtask as a task can be divided into a number of subtasks.**
- **External inputs (operands) are fed into the pipeline at the first stage S_1.**
- **Processed results are passed from stage S_i to stage S_{i+1} for all i= 1, 2,...k-1.**
- **Final result emerges from the pipeline at the last stage S_i.**

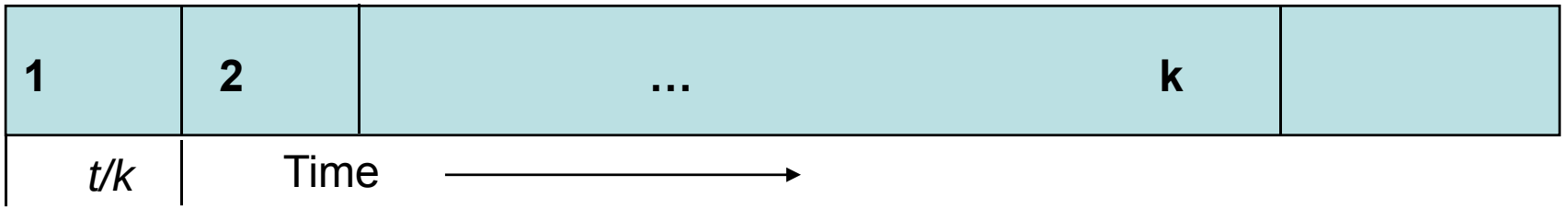


Linear Pipeline Processor

Example: (t) time is required to complete k stages.



- Each task will require (t/k) time as k tasks are performed in k stages.

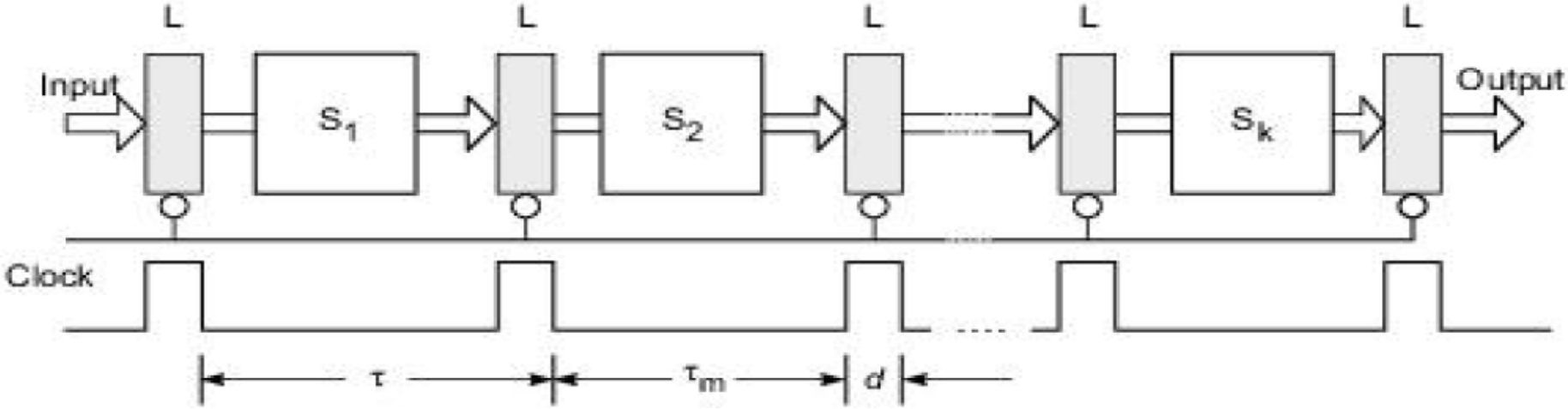


Depending on the control of data flow along the pipeline, the linear pipelines are classified in two categories i.e., Pipelining can be implemented in two ways.

a) Synchronous pipeline, b) Asynchronous pipeline

Linear Pipeline Processor

a)Synchronous Implementation



A Synchronous Pipeline Model

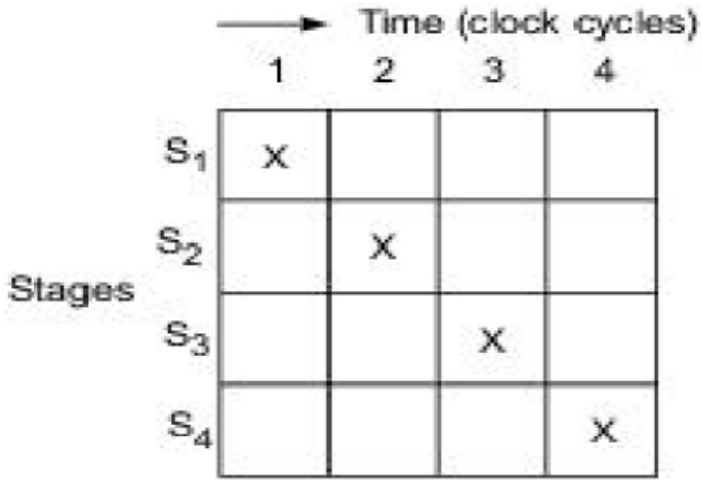
Si=Stage i

L=Latch

τ= Clock Period

τ_m= Maximum Stage Delay

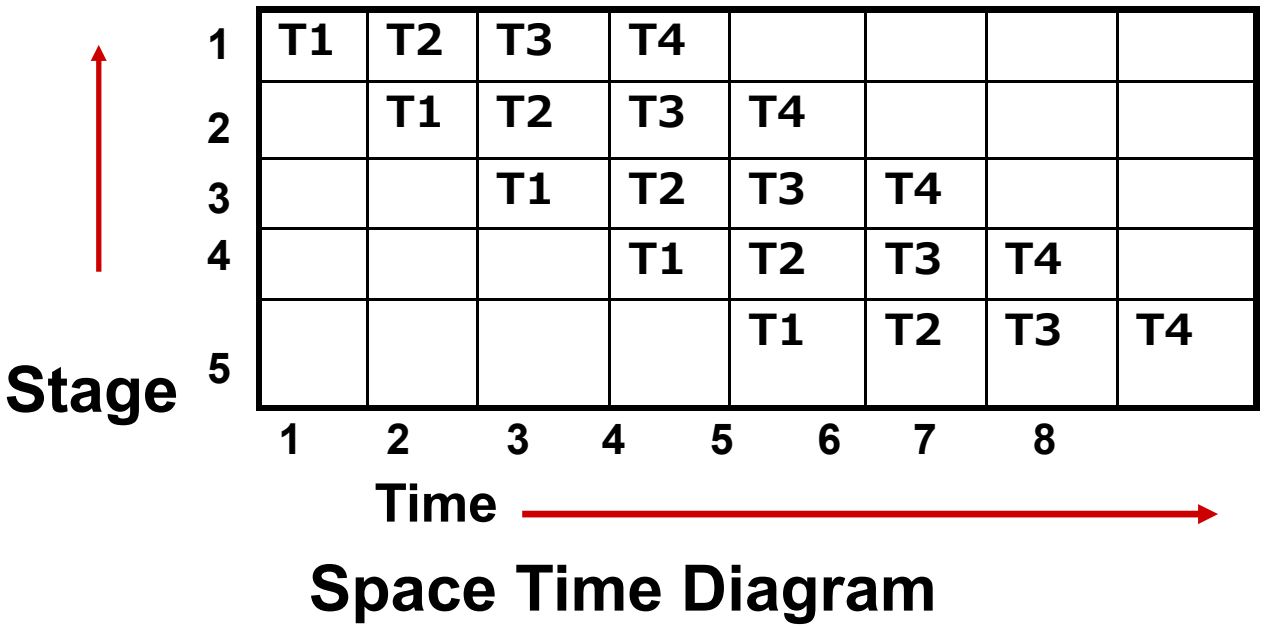
d= Latch delay



Reservation table of a four-stage linear pipeline

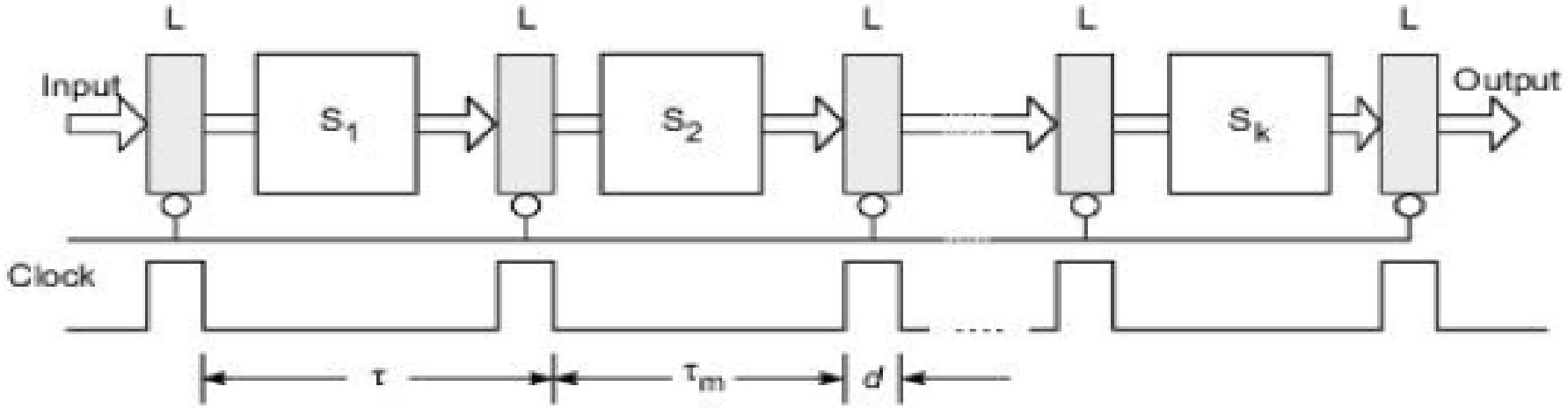
Linear Pipeline Processor

How the tasks are executed?



Space Time Diagram

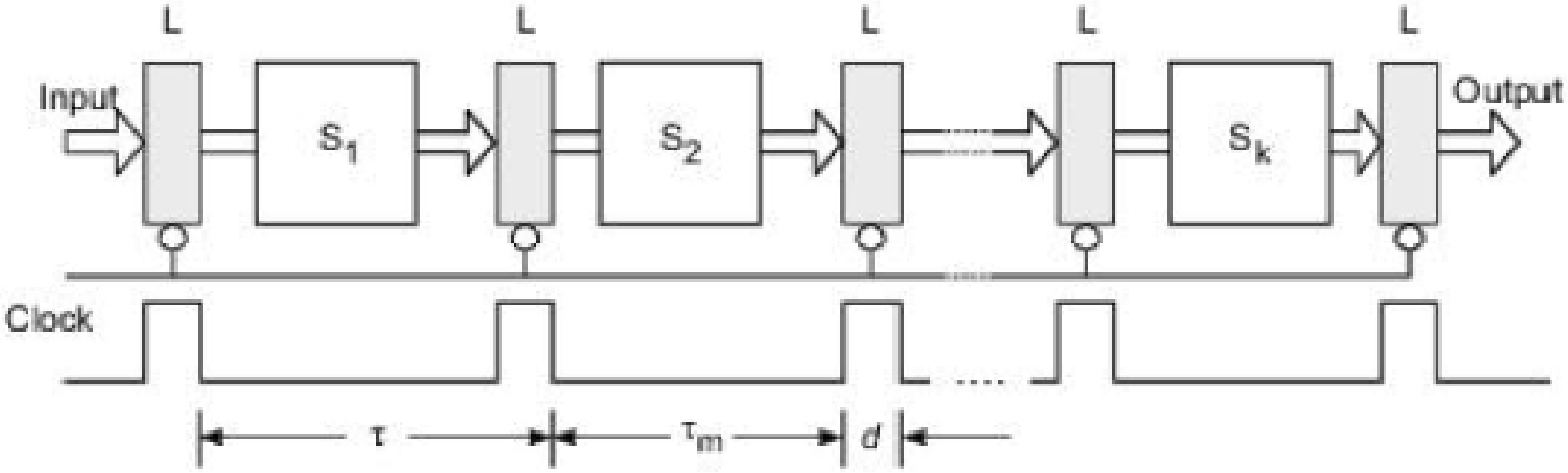
Synchronous Pipeline



- Pipeline processor has k stages and latch is placed in between two stages.
- A common clock is applied to all the stage latches and at the input latch.
- Different stages require different times to perform the operations of different stages.
- Suppose, S1 takes time T1, S2 takes time T2, S3 takes time T3, Sk takes time Tk.
- How do we find out the cycle time that is equal to tau (τ)?

  τ= maximum time slot required among the stages.

Synchronous Pipeline



- **Clock depends on the cycle time or time required to perform the operations.**

Clock cycle (τ) of a pipeline:

τ = maximum of delay among the stage delays + latch delay

= max $\{\tau_i\}_i^k{}_1$ + d = τ_m + d [τ_m = maximum of all the stage time steps]

- **In general, τ_m >> d by one to two orders of magnitude. This implies that the maximum stage delay τ_m dominates the clock period.**

Synchronous Pipeline

Pipeline frequency is defined as

$$f = 1/\tau$$

- **If one result is expected to come out of the pipeline per cycle, f presents the maximum throughput of the pipeline, where, throughput denotes the maximum stage delay as τ_m**

Speed up, Efficiency and Throughput

- A pipeline with k-stages having clock cycle time of tau (τ) executes n tasks.
- First task T1 requires a time equal to ($k.\tau$) to complete its operation.
- Remaining (n- 1) tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to [(n - 1).τ].
- To complete *n* tasks using a *k*-stage pipeline, {k + (n - 1)} clock cycles are required.
- Total time required = $T_k = \{[k+(n-1)].\ \tau\}$, where τ is the clock period.

  An equivalent-function non-pipeline processor which has a flow-through delay of k.τ.

- Time required to execute n tasks on this non-pipelined processor (T_1) = $nk\tau$

Speed up, Efficiency and Throughput

Speedup Factor:

- Throughput is the outputs produced per clock cycle and that throughput will be equal to 1 in ideal situation that means, when the pipeline is producing one output per clock cycle.
- In practical situation, the output cannot be produced in each cycle and because of problems known as hazards.

Speed up = (Time taken by non-pipeline implementation/Time taken by pipeline implementation)

In pipeline implementation, time taken to complete *n* tasks using a *k*-stage pipeline (T_K)= {k + (n - 1)} clock cycles = [k + (n - 1)]. τ

[τ →clock period].

Speed up, Efficiency and Throughput

Time required to execute *n* tasks on this non-pipelined processor (T1) = $nk\tau$

Speed up (S_K) = T_1/T_k
=(Time taken by non-pipeline implementation/Time taken by pipeline implementation)
= $(n.k.\tau)/[\{k + (n - 1).\ \tau\}]=(nk)/[(k-1)+n]$

For large number of tasks, n will be equal to infinity i.e., *k* is negligible compared to n.

So, the expression can be written as…

S_K = $(nk)/[(k-1)+n]$ [As n = ∞, take k-1=0]
= $(nk)/n=k$

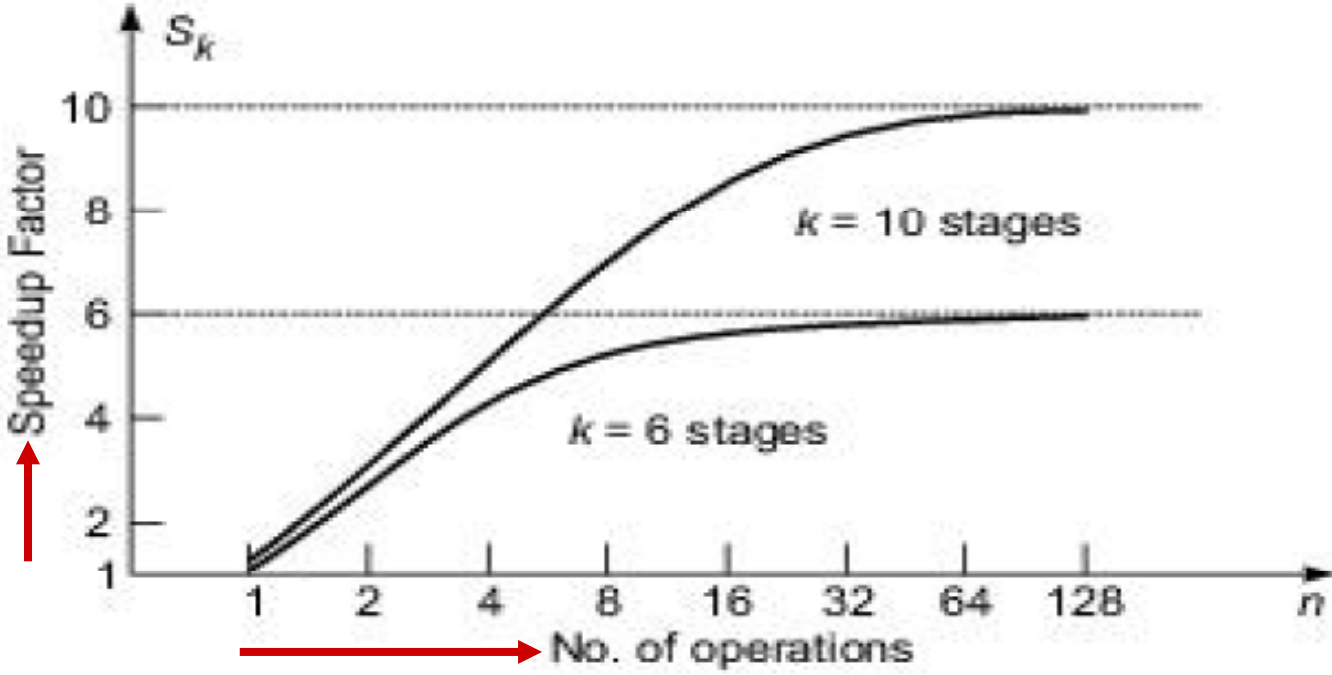
Speedup will be equal to k [k→ number of stages]. In ideal situation, the speedup is equal to number of stages.

Speed up, Efficiency and Throughput

Optimum Speedup [Speed up (Sk) =(nk)/[(k-1)+n]

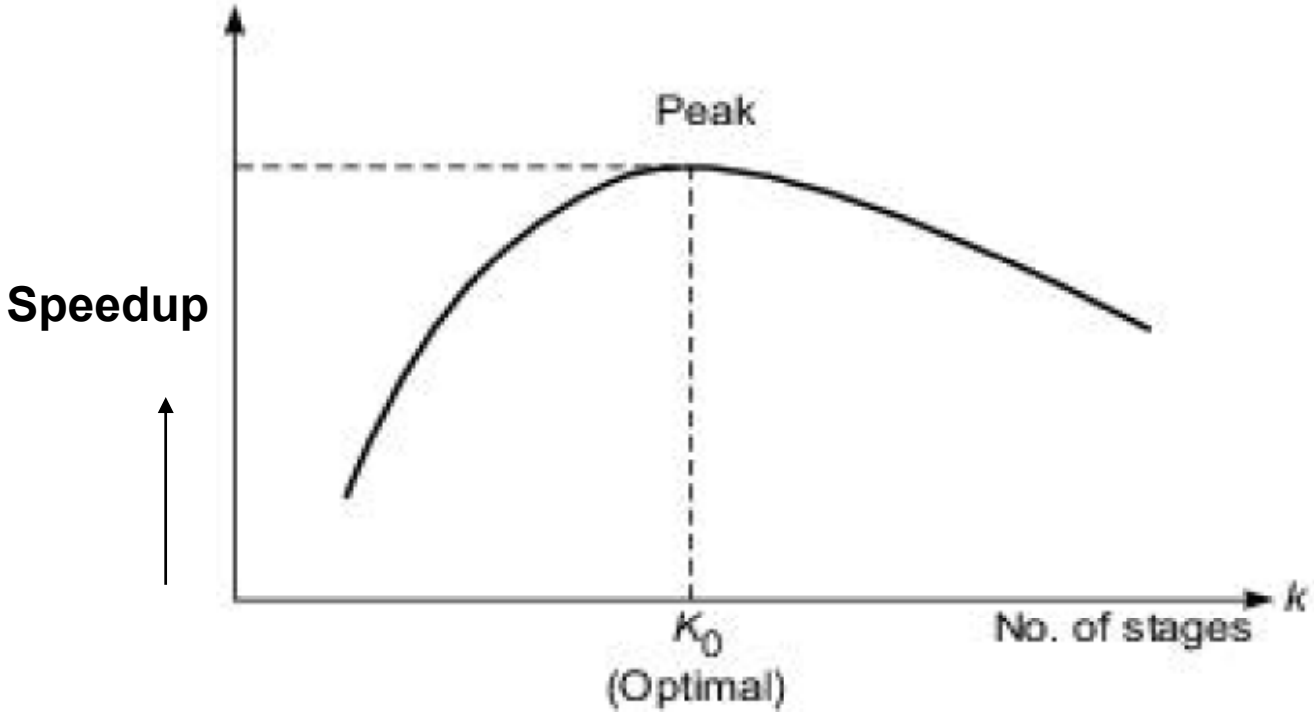
n --->		1	2	4	8	16	32	64	128	256	512	1024	2048
S_K	k=6	1	1.71	2.67	3.69	4.57	5.19	5.57	5.77	5.89	5.94	5.97	5.99
	k=8	1	1.78	2.91	4.27	5.57	6.56	7.21	7.59	7.79	7.89	7.95	7.97
	k=10	1	1.82	3.08	4.71	6.4	7.8	8.77	9.34	9.66	9.83	9.91	9.96

- **This graph and table show that as we vary the number of tasks "n", the speedup also increases.**
- **But there is a limit on the value of K beyond which it can not be increased. For n= 128 and K=10, the maximum speedup is achieved.**



Speed up, Efficiency and Throughput

Optimum Speedup



- **Most pipelining is staged at the functional level with $2 \leq k \leq 15$**

- **In real computers, it does not exceed 10.**

Speed up, Efficiency and Throughput

- **Efficiency:** **The efficiency (E_k) of a linear k-stage pipeline is defined as**

$$E_k = \frac{S_k}{k} = \frac{n}{k+(n-1)}$$

- **Efficiency approaches 1 when n→∝ and a lower bound on E_k is 1/k when n=1.**

- Pipeline Thoughput (H_k) is defined as number of tasks (operations) performed per unit time.

$$H_k = \frac{n}{[k+(n-1)].\ \tau} = \frac{nf}{k+(n-1)} \qquad \text{Where } \tau = 1/f$$

Time required to complete n tasks =[k + (n - 1)]. τ}

Maximum throughput *f* occurs when (Ek) =1 as **n→∝**

Assignments

1. Define Pipeline. When will it be implemented? What are the benefits of the pipelining? How will it be implemented?
2. Explain linear and Non-linear Pipeline?
3. What are static and dynamic Pipeline?
4. What is Reservation Table? Draw the Reservation table for linear and non-linear pipeline?
5. Consider a k-stage pipeline with a clock cycle time τ used to execute n tasks. Derive Speed factor, efficiency and throughput of this pipeline. Graphically show the optimum speedup.
6. Consider the execution of a program of 15,000 instructions by a linear processor with a clock rate of 25 Mhz. Assume that the instruction pipeline has five stages and that one instruction is issued per clock cycle. The penalties due to branch instructions and out-of-sequence executions are ignored.
   a) Calculate the speedup factor using this pipeline to execute the program as compared with the use of an equivalent non-pipelined processor with an equal amount of flow-through delay.
   b) What are the efficiency and throughput of this pipeline processor?

Solutions

- $n = 15{,}000$ instructions or tasks.
- $f = 25\ MHz$.
- $k = 5$ stages.
- $1-$issued processor.

The Speedup (S_k), Efficiency, (E_k), and Throughput (H_k) factors are :

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau}$$
$$= \frac{nk}{k + (n-1)}$$
$$= \frac{(15{,}000)(5)}{5 + (15{,}000-1)}$$
$$= \frac{75{,}000}{15{,}004}$$
$$= 4{,}999$$

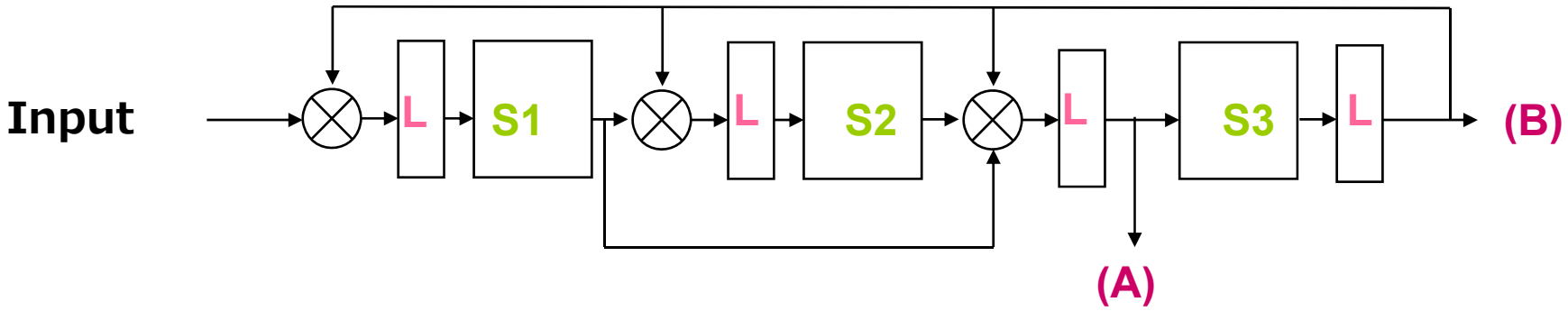
$$H_k = \frac{nf}{k + (n-1)}$$
$$= \frac{(15{,}000)(25)}{5 + (15{,}000-1)}$$
$$= \frac{375{,}000}{15{,}004}$$
$$= 24{,}99\ MIPS$$

$$E_k = \frac{S_k}{k}$$
$$= \frac{4{,}999}{5}$$
$$= 0{,}999$$

Pipeline -II

Nonlinear Pipeline [Dynamic]

- **can be reconfigured to perform variable functions at different times that means, multiple functions can be computed.**

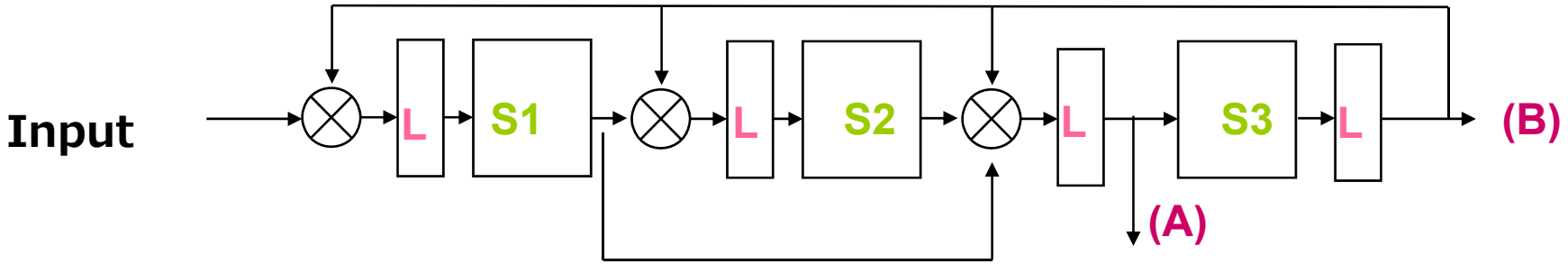


- **allows feed forward and feedback connections in addition to the streamline connections. For this reason, this structure is called nonlinear pipeline.**

- **This architecture computes two function A and B. Same pipeline can either compute a function A or compute a function B.**

Nonlinear Pipeline

- **Consider the nonlinear pipeline implementing two functions (A) and (B).**



A	t0	t1	t2	t3	t4	t5	t6	t7
S1	X			X			X	
S2		X						X
S3			X		X	X		

B	t0	t1	t2	t3	t4	t5	t6	t7
S1	X				X			
S2			X			X		
S3		X		X			X	

- Operation A represented by the reservation table A, requires 8 cycles to compute (time steps: t0 to t7) and this is the sequence of stage occupancy.
- First, input data enters into S1, then to S2, then to S3, then again to S1; then S1 to S3, then in next cycle it will be in S3, S3 to S1; S1 to S2 and from S2 it finishes or it goes out.

- Operation B represented by the reservation table B, requires 7 cycles to compute (time steps: t0 to t6).
- First, input data enters into S1, then to S3, then to S2, then to S3; then S3 to S1, then S1 to s2, then S2 to S3; and finally from S3 it finishes or it goes out, that means, s3 is the output point for function B.

Nonlinear Pipeline

- At a particular time step, only one stage usually works but more than one stage may be used during one time step.
- In the reservation table, for function A during time step t3, both the stages s1 and s3 are used.
- After time step t2, the output is available for s3. During t3, both s1 and s3 are used. By setting the multiplexer in such a way that the output of s3 goes to both s1 as well as s3 itself.
- During time step t4, s3 is used taking data from s1. During t5 step, s3 is used taking input from s3.
- During t_6, s1 is used taking input from s3.
- Finally during t7, s2 is used. We get the output from the computation stage s2.

A	t0	t1	t2	t3	t4	t5	t6	t7
S1	X			X			X	
S2		X						X
S3			X	X	X	X		

Nonlinear Pipeline

- **Pipeline may be used for computation of more than one functions simultaneously.**
- **Analysis of the reservation table helps in designing the pipeline for multi-functions.**

A	t0	t1	t2	t3	t4	t5	t6	t7
S1	X			X			X	
S2		X						X
S3			X	X	X	X		

- **Reservation table determines time step at which, a new task is fed to the pipeline.**
- **We cannot feed in a new task to the pipeline at every time step because in that case there can be data clash.**

Nonlinear Pipeline [Pipeline Scheduling]

- If two tasks use same pipeline stage at same time step, then there will be data clash or collision that means any attempt by two or more initiations to use the same pipeline stage at the same time will cause a collision.
- Task needs to be scheduled to avoid clash. This is called pipeline scheduling. We have to decide, when we can schedule a new task to the pipeline.

- Actually, the latency analysis will be performed to avoid collision.
- Latency is actually the delay between two successive data inputs or initiations that are fed to the pipeline.
- The main objective is to calculate the latency value or the sequence of latency values that will not result in collision in any of the stages.
- Latency is basically defined as the number of time units or clock cycles between two successive initiations that means, the minimum number of time steps is needed to apply two inputs to the pipeline. If latency analysis is not done careful enough, then two or more initiations may try to use the same stage at the same time that results in a collision.

Nonlinear Pipeline

- **Latency is the time step difference between initiation of two tasks.**
- **In S_1 Stage, the latency between subsequent two time steps (t_1 and t_9) will be 8**
- **Latency sequence: A latency sequence is the sequence of latencies of successive initiation of tasks.**
- **Computation stages (5): S_1, S_2, S_3, S_4,S_5**
- **9 time steps = $t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_8\ t_9$**

A	t1	t2	t3	t4	t5	t6	t7	t8	t9
S1	X								X
S2		X	X					X	
S3			X						
S4				X	X				
S5						X	X		

Stage	Latency between
S1	t1-t9=8
S2	t2-t3=1 t2-t8=6 t3-t8=5
S3	
S4	t4-t5=1
S5	t6-t7=1

Nonlinear Pipeline

- **Some of the latencies will definitely result in a collision.**
- **Latencies which result collision are referred to as forbidden latencies and these latencies can not be used.**
- **Forbidden latencies can simply be estimated by calculating the distance between two marks (X) in the same row of the reservation table.**

- **A collision implies resource conflicts between two initiations in the pipeline.**
- **All collisions must be avoided in scheduling by a sequence of pipeline initiations.**

Stage	Latency between
S1	t1-t9=8
S2	t2-t3=1 t2-t8=6 t3-t8=5
S3	
S4	t4-t5=1
S5	t6-t7=1

Set of forbidden latencies

F = ((9-1), (2-1), (8-2), (8-3), (6-5),(8-7))
= (8, 1, 6, 5, 1,1)
F = [1, 5, 6, 8]

Nonlinear Pipeline

A	t1	t2	t3	t4	t5	t6	t7	t8	t9
S1	X								X
S2		X	X					X	
S3			X						
S4				X	X				
S5						X	X		

- **Forbidden latencies for this reservation table**

  F = [1, 5, 6, 8]

- **Latencies that do not cause collisions are called permissible Latencies.**
- **Permissible Latencies = [2,3,4,7]**

- **Latency sequence: defined as a sequence of permissible non-forbidden latencies that do not result in a collision.**
- **In this example, sequence (2,3,4,7) is permissible sequence. Suppose, we are using a latency sequence of 2 3 4, then a gap of 2 is given between the first and second input, a gap of 3 is given between second and third input, a gap of 4 is given between third and fourth input. They will not create any collision.**
- **Latency cycle is nothing but a sequence where the same subsequence is repeated.**
- **Example: suppose, we use a latency sequence (say): 2,3,2,3,2,3,… and this repeats indefinitely. The latency cycle is (2, 3). Similarly, if we use a latency sequence of (3, 3, 3, 3, 3), then we can say, a latency cycle contains a single latency (3).**

Nonlinear Pipeline (Scheduling without Collision)

Now, we will discuss technique to obtain the shortest latency between initiations causing no collision.

Collision Vector:collision vector is a bit vector or a binary vector. Each of the components in this vector can be either be 0 or 1.

- **In a reservation table having m columns, the maximum forbidden latency is n ≤ (m-1).**
- **Permissible latencies p will satisfy the condition: 1 ≤ p ≤ (n-1).**
- **Collision vector is an n-bit binary vector.**

A	t1	t2	t3	t4	t5	t6	t7	t8	t9
S1	X								X
S2		X	X					X	
S3			X						
S4				X	X				
S5						X	X		

- $C = (C_n, C_{n-1}, \ldots, C_2, C_1)$, **where n is the maximum forbidden latency and** $C_i = 1$ **if latency I causes collision i.e., (i ∈ F (F=Forbidden latency)), else** $C_i = 0$.
- C_n **is always 1.**

Forbidden latencies for this function (A): $F = [1, 5, 6, 8]$,
Collision Vector for this function: $C = 1\ 0\ 1\ 1\ 0\ 0\ 0\ 1$

Nonlinear Pipeline

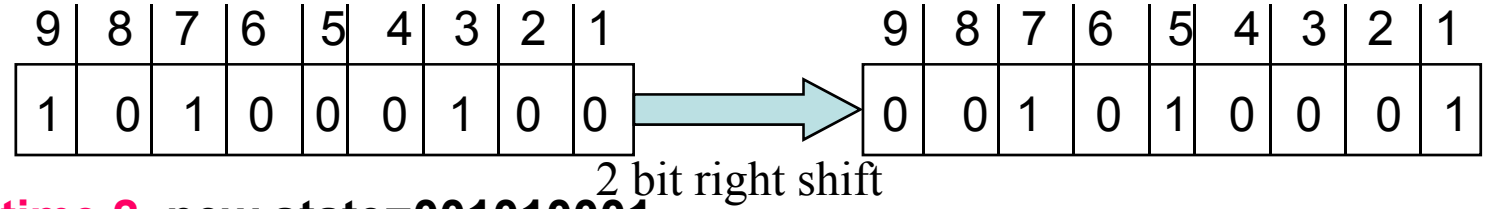
In n-bit collision vector (n, where m= maximum forbidden latency), a particular bit *(i)* $(C_1, C_2, \ldots C_i, \ldots, C_n)$ is 1 if latency *i* causes a collision, and that bit is 0 if the latency *i* does not cause a collision.

Next, from the collision vector, we will construct a state diagram, where we define the states as the status of collisions at a particular point of time. This state diagram tells which time gaps or latencies can result in a collision at that point of time. Basically, the state transition diagram can contain multiple states and we can move from one state to the other state.

In the state diagram, the initial state is the collision vector. From the collision vector, we can move from any state to some other state by advancing the time by a number say, p. Suppose, we are currently in time t, If we advance the time by p. then the present state will be shifted right by p positions.

Nonlinear Pipeline

Example: suppose, Present state (101000100) where time = (1, 2, 3, 4, 5, 6, 7, 8,9) and forbidden times (3,7,9) where we cannot feed the next input --- these are forbidden latencies. Now, say, if we advance the time by 2 steps, then 3 position after two steps shift will become 1, 7 position will become 5, 9 position will become 7. So, actually we are shifting this right by 2 positions.

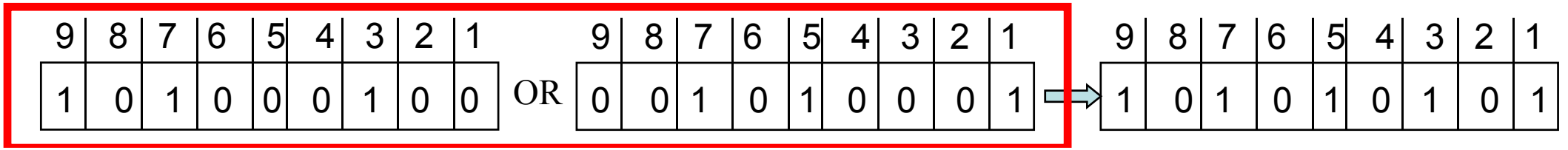


New status after time 2, new state=001010001.

The next state is obtained by shifting it right. The initial collision vector lists the forbidden latencies. The latencies that are forbidden will always remain forbidden.

After right shifting, the bit by bit logical OR operation will be performed on right shifted result with the initial collision vector so that whenever there is a 1 in the initial collision vector, that will appears in the final result, because those will always be forbidden and we cannot use that latency at any point in time.

Now the collisions will occur at time 1, 3, 5, 7, and 9.



Nonlinear Pipeline

Now we are considering our previous example.

A	t1	t2	t3	t4	t5	t6	t7	t8	t9
S1	X								X
S2		X	X					X	
S3			X						
S4				X	X				
S5						X	X		

Forbidden latencies for this function (A): F = [1, 5, 6, 8],

Permissible Latencies = [2,3,4,7]

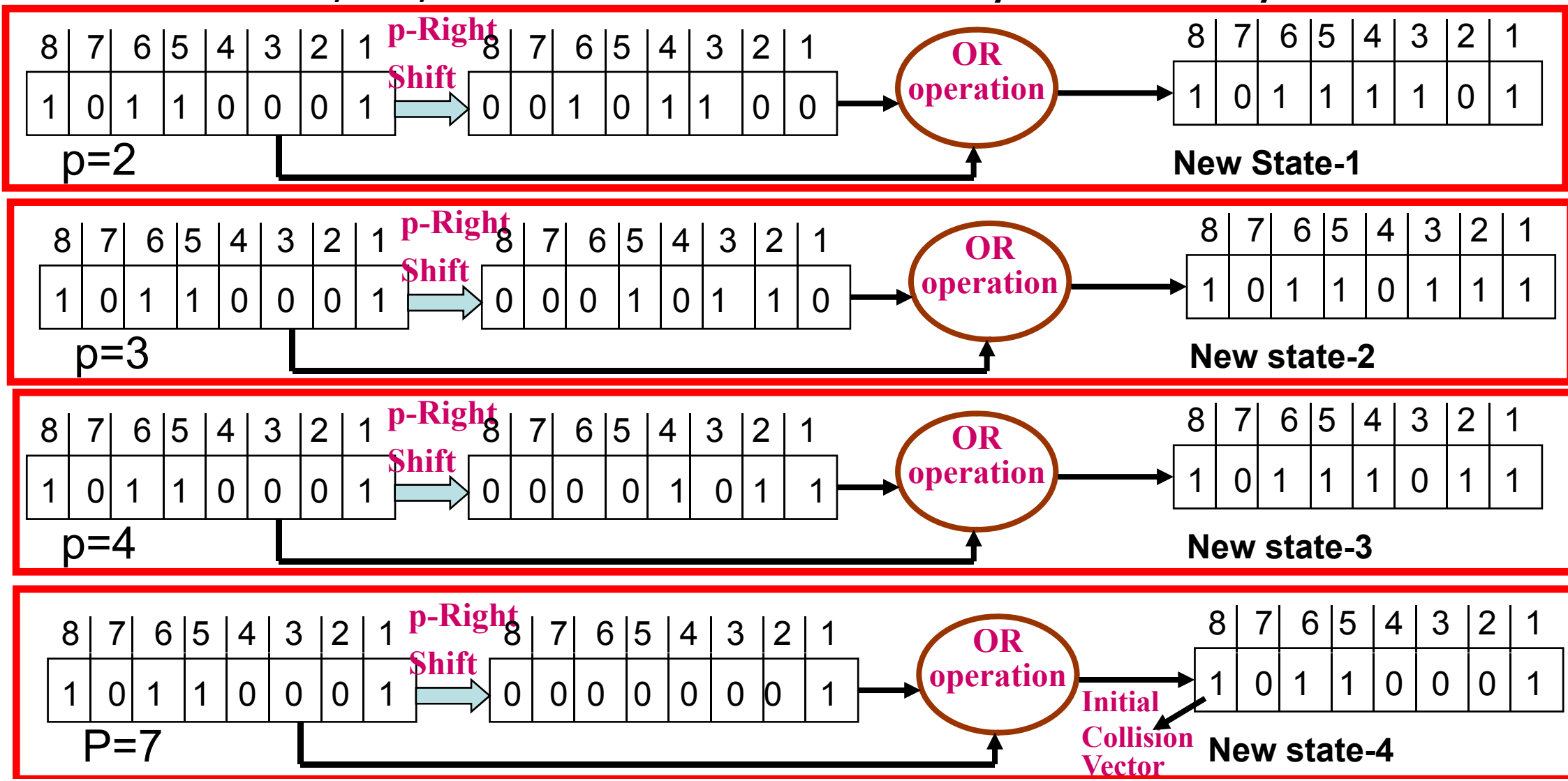
Collision Vector for this function: *C*=1 0 1 1 0 0 0 1

- **From the collision vector C, we can construct a state diagram specifying the permissible state transitions among the successive initiations.**
- **Next state at time ($t+p$) is obtained by shifting the present state *p*-bits to the right and OR-ing with the initial collision vector *C*.**
- **In this example, the forbidden latencies are 1,5,6,8. So, (1,5,6,8) will be the initial state. Now from the initial state, we can advance the time only by that amount where we have 0's that are not forbidden, which means time 2, 3, 6 or 7, or more. If I advance it by 1 you see my collision vector was 1 0 1 1 0 1 0.**

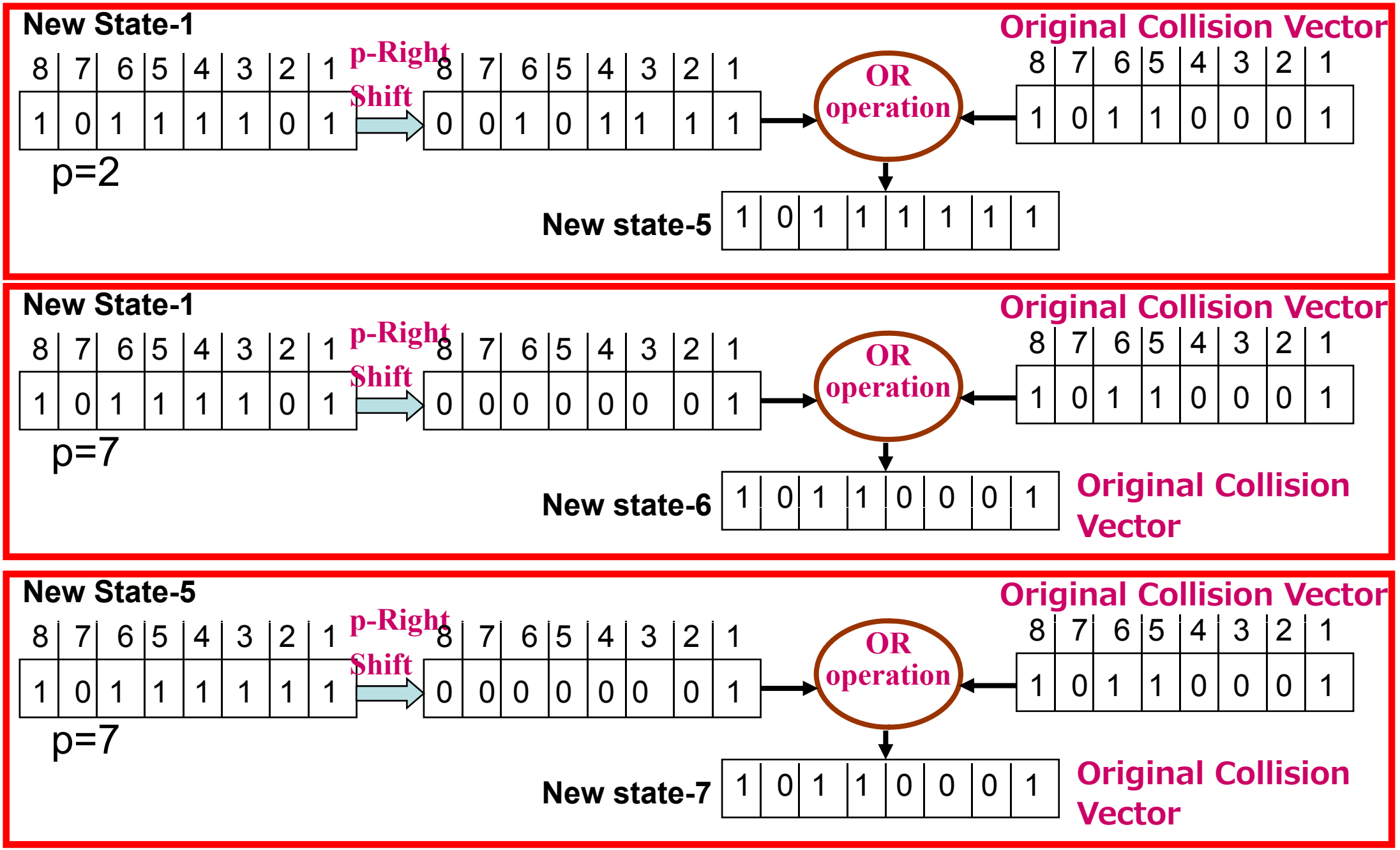
Nonlinear Pipeline

Forbidden latencies (F) = [1, 5, 6, 8], Permissible Latencies = [2,3,4,7], Collision Vector (*C*)=1 0 1 1 0 0 0 1

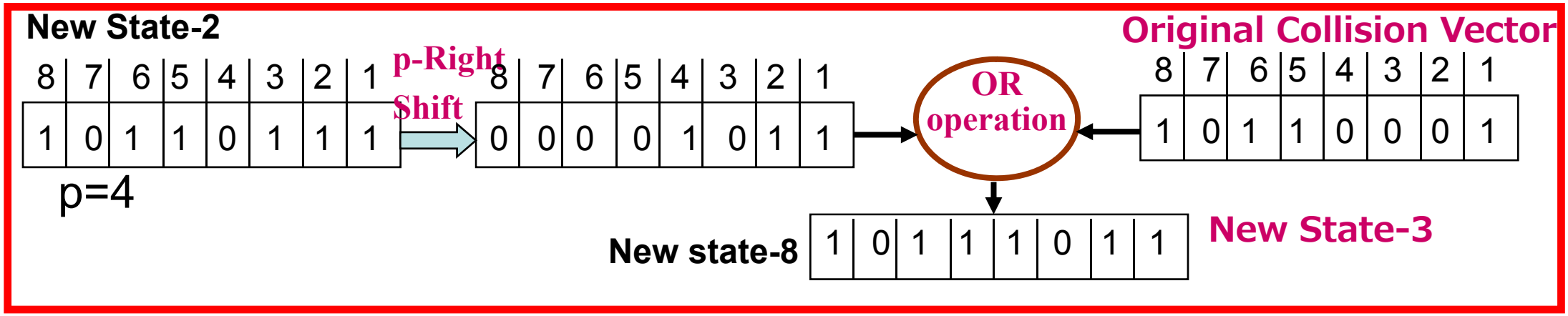
- **We find '0' in 2nd, 3rd, 4th, and 7th latency of the collision vector. We can start new task at 2nd, 3rd, 4th and 7th or after 7th latency i.e. 7+ latency**

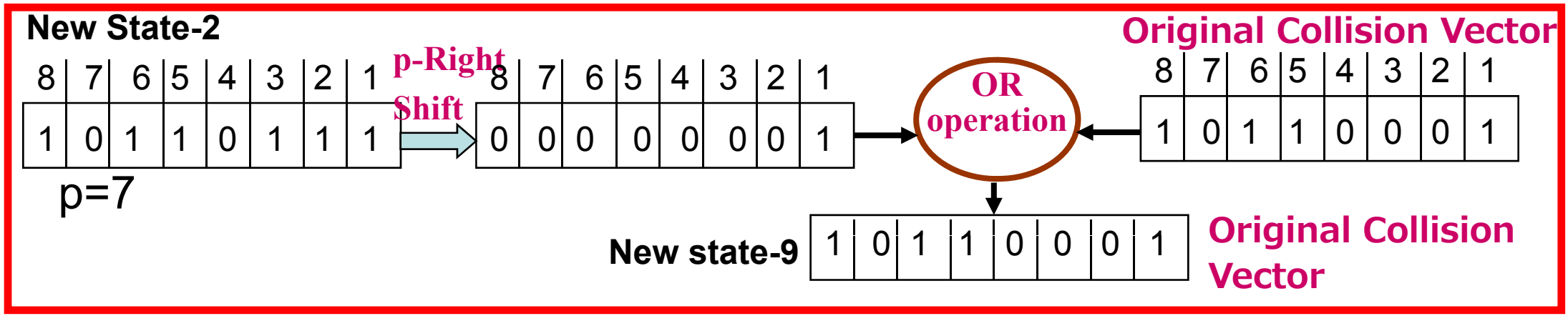


Nonlinear Pipeline

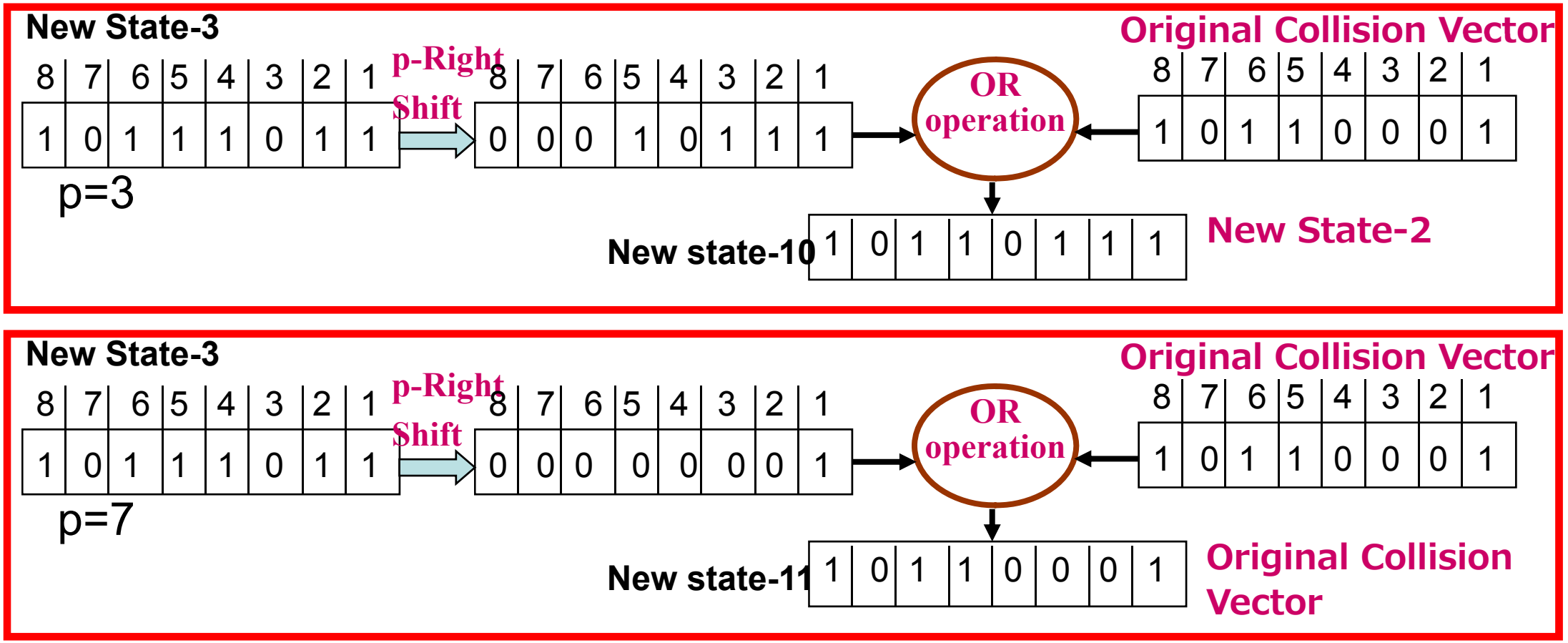


Nonlinear Pipeline

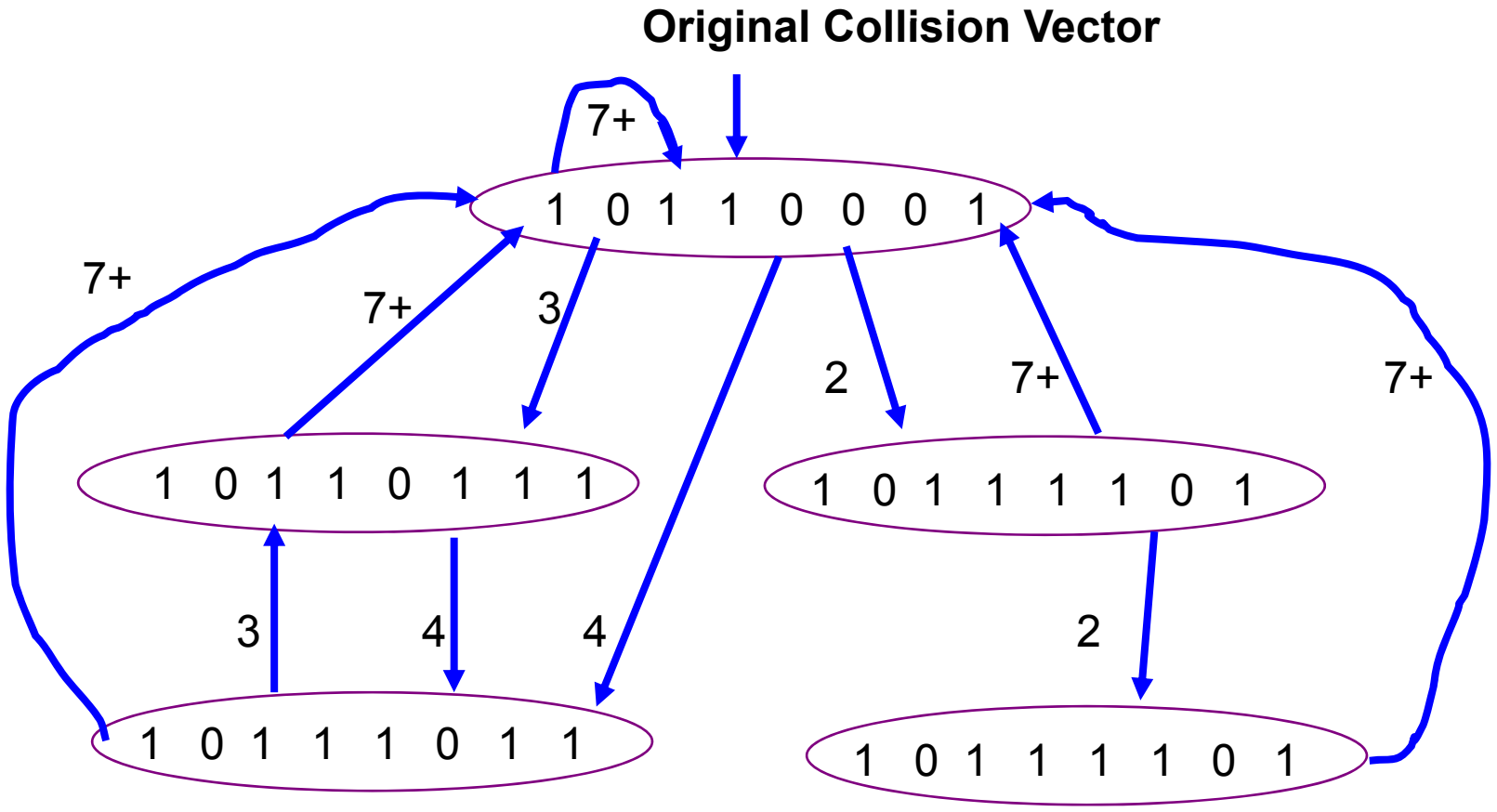




Nonlinear Pipeline



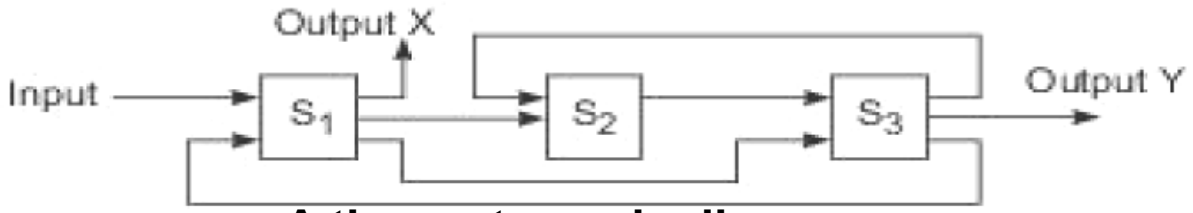
Nonlinear Pipeline



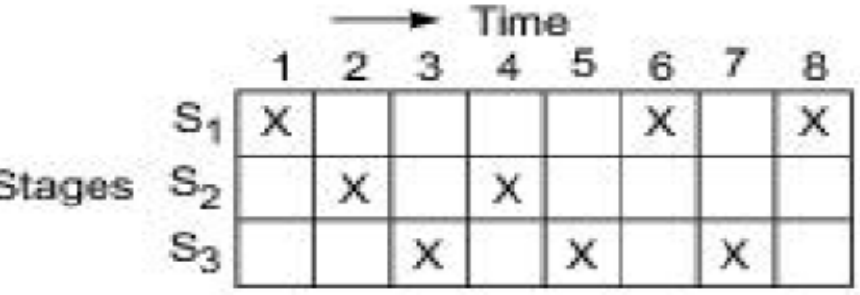
State Diagram for Collision Vector (1, 5,6,8)

Nonlinear Pipeline

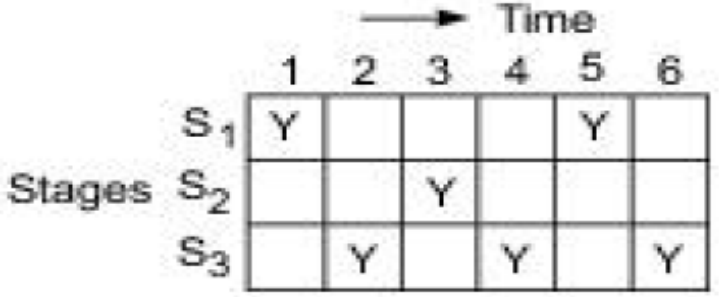
Example: Find the reservation tables for both the functions X and Y and draw the state transition table for both X and Y.



A three-stage pipeline



Reservation table for function X



Reservation table for function Y

Forbidden Latency=(6-1),(8-1),(8-6),(4-2),(5-3),(7-3),(7-5) =5,7,2,2,2,4,2=2,4,5,7

Permission latencies = 1,3,6

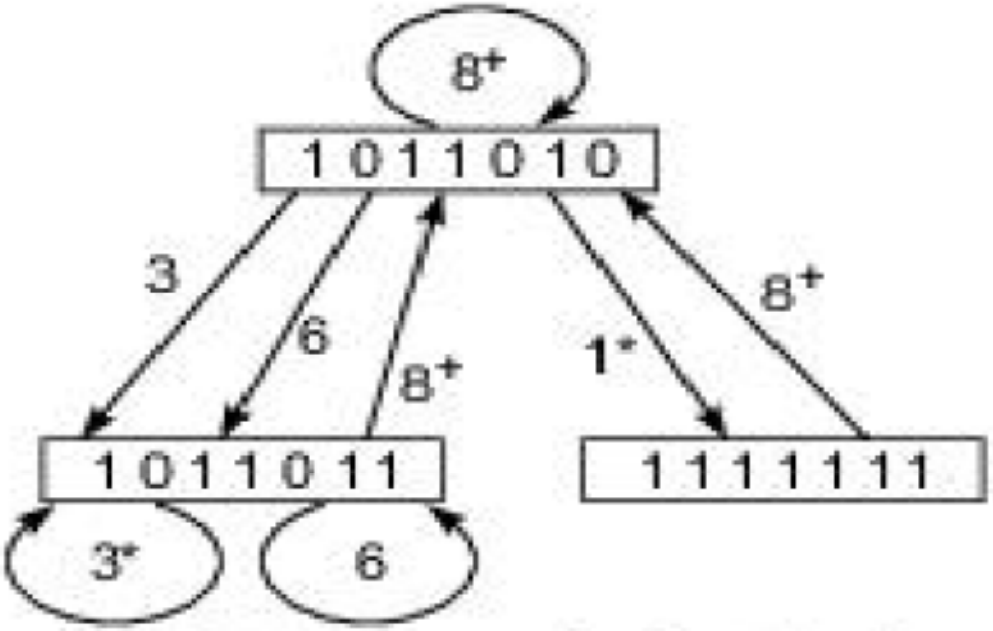
Collision Vector={1011010}

Forbidden Latency=((5-1), (4-2),(6-2),(6-4) =(4,2,4,2)=(2,4)

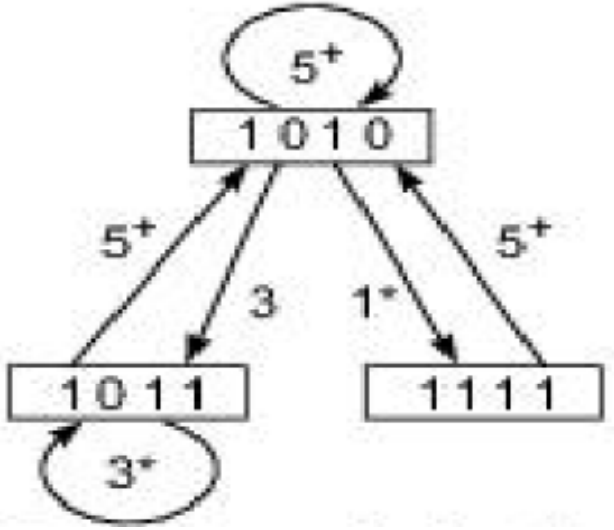
Permissible Latencies=(1,3)

Collision Vector=1010

Nonlinear Pipeline



State diagram for function X

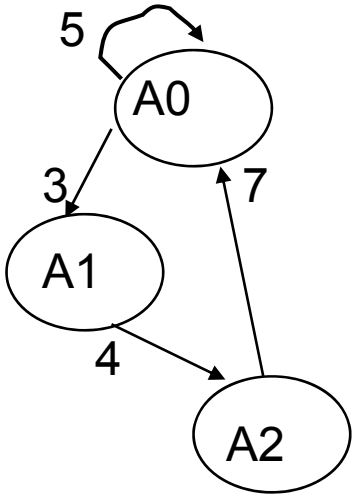


State diagram for function Y

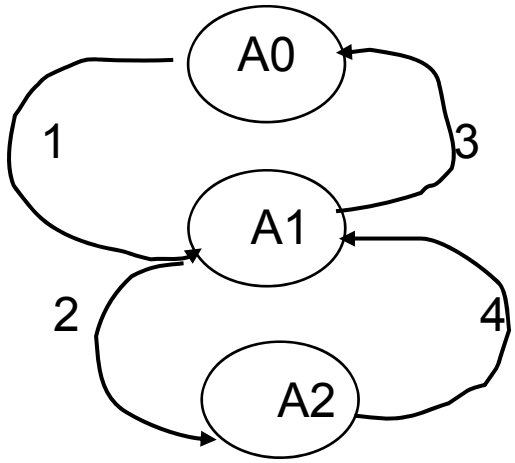
Nonlinear Pipeline

- **Average latency: {sum of all the latency} /{ # of latencies in the cycle}. Example: latency cycle (1,5) has an average latency of {(1+5)/2} or 3.**
- **Constant latency cycle: If in the latency cycle, only one latency value is contained, it is called constant latency cycle.**
- **Suppose, a task starts at cycle 1, and a new second task starts at cycle 4, the cycle difference = 4-1=3, now a 3rd new task starts at cycle 7, then cycle difference = (7-4)=3, and so on.**
- **Here, constant cycle latency =3=3,3,3,…**
- **Average latency is a dominant parameter for scheduling events in a pipeline.**
- **Success of pipeline architecture implementation depends greatly on scheduling of events in a pipeline without causing collision.**
- **This information is mainly obtained by deriving the minimal average latency (MAL) between initiations.**

Nonlinear Pipeline



(a) State Diagram: Simple Cycle



(b) State diagram: not simple

- **Simple Cycle: In the state diagram, among the cycles, there are certain cycles in which each state appears only once. Such cycles are called simple cycle. A simple cycle is a latency in which each state appears once.**

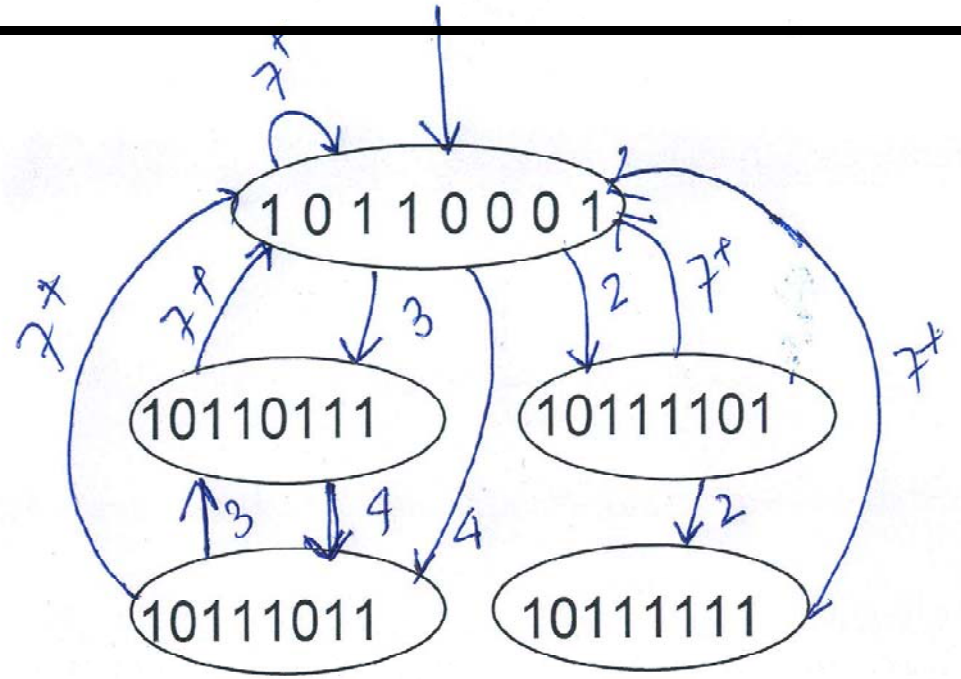
- **In Fig. (a), cycles (5), (3,4,7) are simple cycles. (5) is simple cycle because it is not repeating state more than once.**

- **In fig. (b), cycle (1,2,4,3) is not simple cycle because it repeats state (A1) twice.**

Nonlinear Pipeline

A	t1	t2	t3	t4	t5	t6	t7	t8	t9
S1	X								X
S2		X	X					X	
S3			X						
S4				X	X				
S5						X	X		

Reservation Table for function A



State diagram

List of all the Simple Cycle:

Cycles	Average latency
(7)	7
(3,7)	5
(4,3)	3.5
(4,3,7)	4.6
(4,7)	5.5
(2,7)	4.5
(2,2,7)	3.6
(3,4,7)	4.6

Cycle (3,4,3,7) is not simple cycle because it repeats state (10110111)

Nonlinear Pipeline

List of all the Simple Cycle:

Cycles	Average latency	
(7)	7	
(3,7)	5	
(4,3)	3.5	Greedy Cycle
(4,3,7)	4.6	
(4,7)	5.5	
(2,7)	4.5	
(2,2,7)	3.6	Greedy Cycle
(3,4,7)	4.6	

- **Greedy Cycle: In the state diagram, the simple cycles with edges of minimum latencies are called greedy cycles.**

- **Greedy cycles are those cycles in which the average latency is less than equal to (=<) no of "1" bits in the initial collision vector. In collision vector (1 0 1 1 0 0 0 1), # of "1"= 4.**

- **So, Greedy cycles will be those cycles whose average latency is less than equal to 4.**

- **Cycles (4,3) and (2,2,7) are greedy cycles.**

Nonlinear Pipeline

List of all the Simple Cycle:

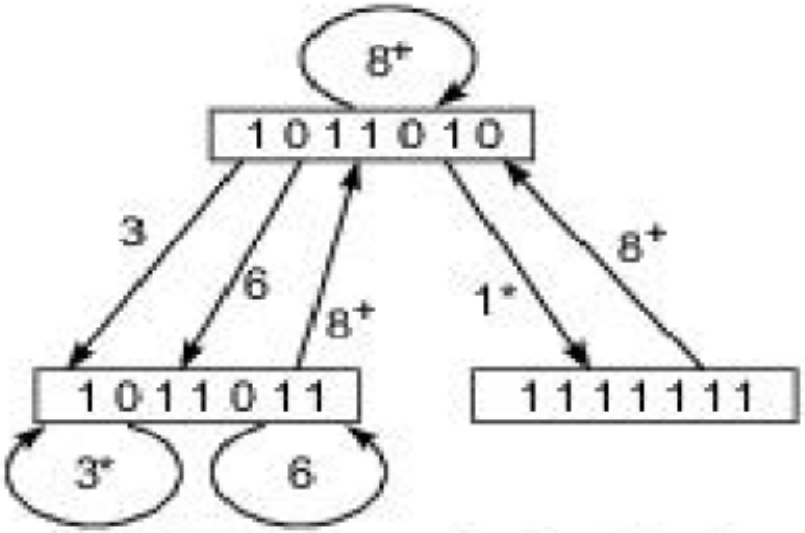
Cycles	Average latency	
(7)	7	
(3,7)	5	
(4,3)	3.5	Greedy Cycle
(4,3,7)	4.6	
(4,7)	5.5	
(2,7)	4.5	
(2,2,7)	3.6	Greedy Cycle
(3,4,7)	4.6	

- **MAL: At least, one of the greedy cycles will ultimately give Minimum Average Latency (MAL) that could be used in pipeline scheduling to make the entire pipeline collision free.**

- **Here, greedy cycle (4,3) has an average latency (3.5) which is MAL for executing the function A with no-collision.**

- **Successive initiations of tasks in the pipeline could then be carried out using this MAL to maximize the throughput of the pipeline.**

Nonlinear Pipeline



State diagram for function

A simple cycle is a latency cycle in which each state appears only once. In this state diagram, only (3), (6), (8), (1, 8), (3, 8), and (6, 8) are simple cycles. The cycle [1, 8, 6, 8) is not simple because it travels through the state (1011010) twice. Similarly, the cycle (3, 6, 3, 8, 15) is not simple because it repeats the state (1011011) three times.

Some of the simple cycles are greedy cycles. A greedy cycle is one whose edges are all made with minimum latencies from their respective starting states.

Cycles (1, 8) and (3) are greedy cycles. In Fig.6.6c Greedy cycles are (1, 5) and (3). Such cycles must first be simple, and their average latencies must be lower than those of other simple cycles. The greedy cycle [1, 8) has an average latency of [(l + 8)/2= 4.5], which is lower than that of the simple cycle (6, 8) [(6 + 8)/2=7].

The greedy cycle (3) has a constant latency which equals to the MAL for evaluating function X without causing a collision.

Nonlinear Pipeline

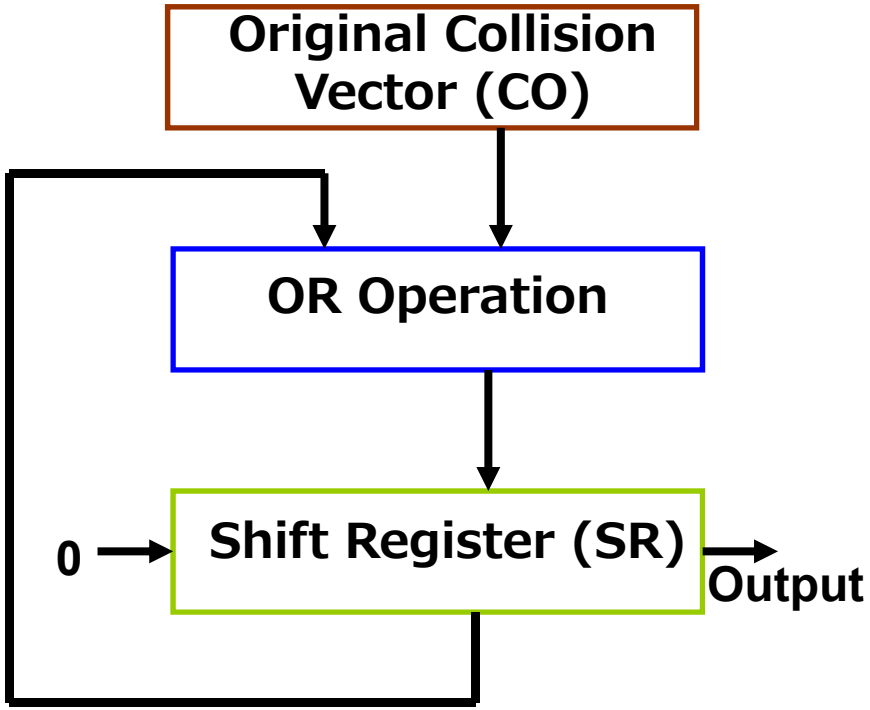
- **Collision vector determines the time steps at which, the new task would be initiated.**
- **At the start of the first task, this collision vector is loaded in to a shift register.**
- **At the end of every time step, shift register is given a right shift.**
- **If a bit 1 comes out of the shift register, then we cannot initiate a new task.**
- **If a 0 comes out of the shift register then we can initiate a new task.**
- **Whenever we initiate a new task, the collision vector is going to be different because the reservation table of the new task is going to be superimposed with the remaining part of the reservation table of the previous tasks. So we have to re-compute this collision vector to decide when next a new task can be initialized.**
- **whenever we initiate a new task, we simply perform bit wise 'or' with the collision vector of the new task with remaining in the shift register and that becomes the new collision vector.**

Nonlinear Pipeline

Scheduling Algorithm

Step 1: Load Initial Collision Vector in a SR

```
If (LSB of SR is 1) then
    begin
        Do not initiate an operation;
        Shift SR right by one position
        with 0 insertion at Left end
    end
else
    begin
        Initiate an operation;
        Shift SR right by one position
        with 0 insertion at the left end
        SR=SR (OR) CO
end
```



If Output =0, Operation will be initiated (No Collision)

If Output =1, Operation will not be initiated Collision)

Explanation is given in the next slide

Nonlinear Pipeline

- **First, initial collision vector is loaded in a shift register. At every point in time, a right shift is done, where a "0" is fed from left side of Shift register and right output of the shift Register is checked whether a "0" is coming out or a 1 is coming out. If a 1 is coming out, it indicates that next time instant is forbidden, the data can not be fed at that time.**

- **If the last bit that is coming out is 1 then we do not initiate an operation simply and shift the register by 1 bit with 0 insertion at the left end, and in the next iteration again check whether new input is allowed to be fed or not.**

- **If it is a 0, we are moving to the next state. Next state is achieved by shifting the right and ORing with the collision vector. If the bit is 0, we initiate an operation and then shift it right and the shifted value is OR-ed with the collision vector and is loaded back into the shift register, and this process repeats.**

Nonlinear Pipeline

- **At every step we are trying to shift it by a number of bits that is allowed. We go on shifting till the next 0 comes, then we OR it with the collision vector. And we always move when you get the first 0; that means we are looking only for greedy cycles. From every state we are moving out following the smallest or the least value of the cycle.**

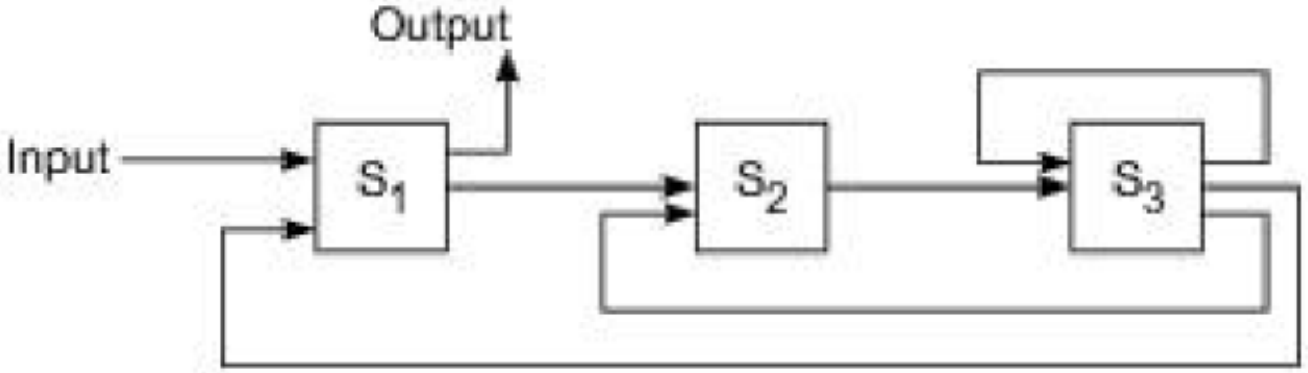
- **The initial collision vector C is initially loaded into the register. The register is then shifted to the right. Each 1-bit shift corresponds to an increase in the latency by 1. When a '0' bit emerges from the right end after p shifts, it means p is a permissible latency. Likewise, a 1 bit being shifted out means a collision, and thus the corresponding latency should be forbidden.**

- **Logical 0 enters from the left end of the shift register. The next state after p shifts is thus obtained by bitwise-ORing the initial collision vector with the shifted register contents.**

Nonlinear Pipeline

Pipeline Schedule Optimization

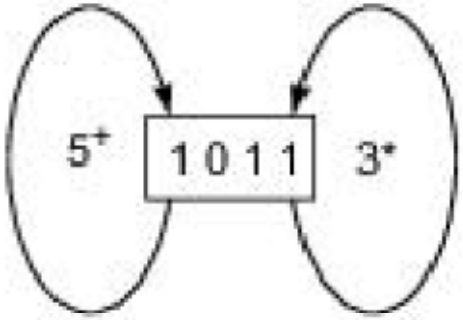
- There are ways to optimize a pipeline schedule. Consider a non-linear pipeline. Reservation table shows that at time step 4, two of the stages are simultaneously used.
- Forbidden latencies= (1 ,2,4), Collision Vector= 1 0 1 1;MAL=3



Three Stage Pipeline

	1	2	3	4	5
S1	X				X
S2		X		X	
S3			X	X	

Reservation Table



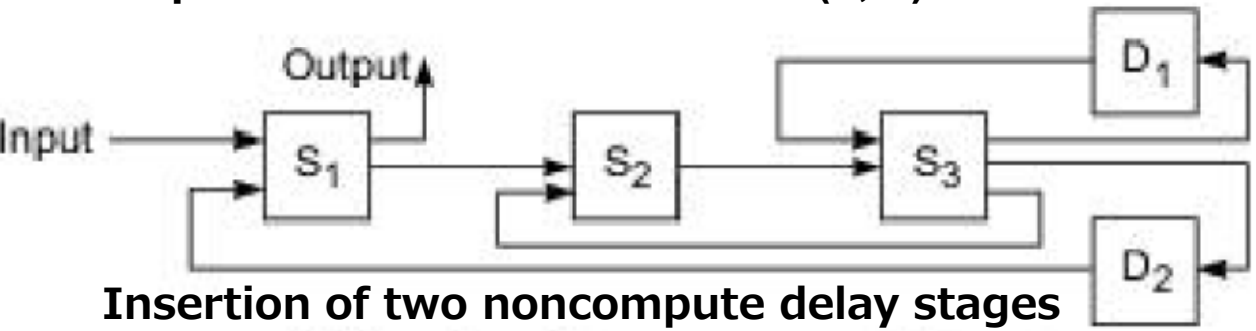
State Transition Diagram

Now, we will insert some delay elements (some dummy stages) in the pipeline, which do not compute anything just it take up some clock cycles. If we insert dummy stages that consume some cycles, then operation time will increase. The time may be increasing for a single computation, but when many data are fed one after the other in the existing pipeline, there will be lot of collisions, but if we insert these delay elements, the number of collisions may become less.

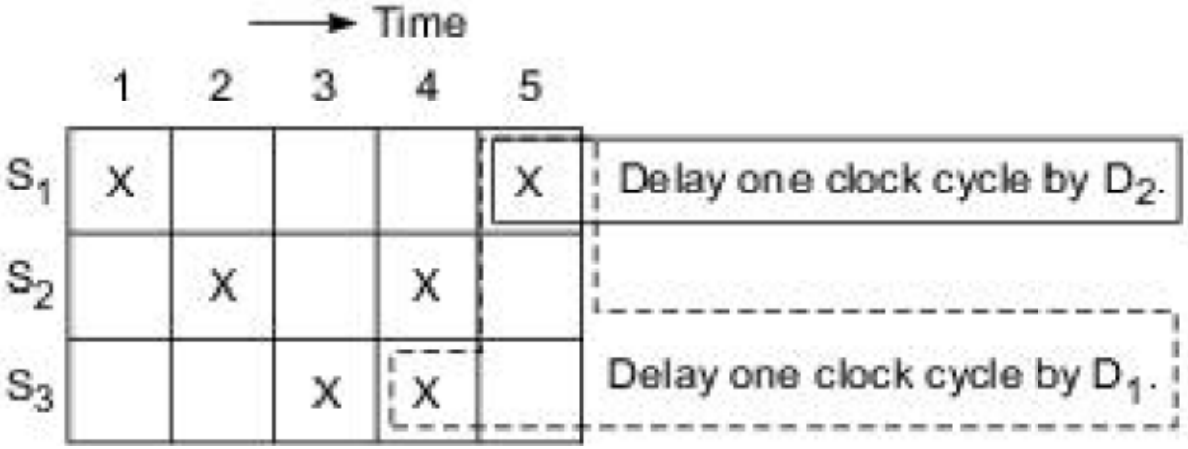
Nonlinear Pipeline

Pipeline Schedule Optimization

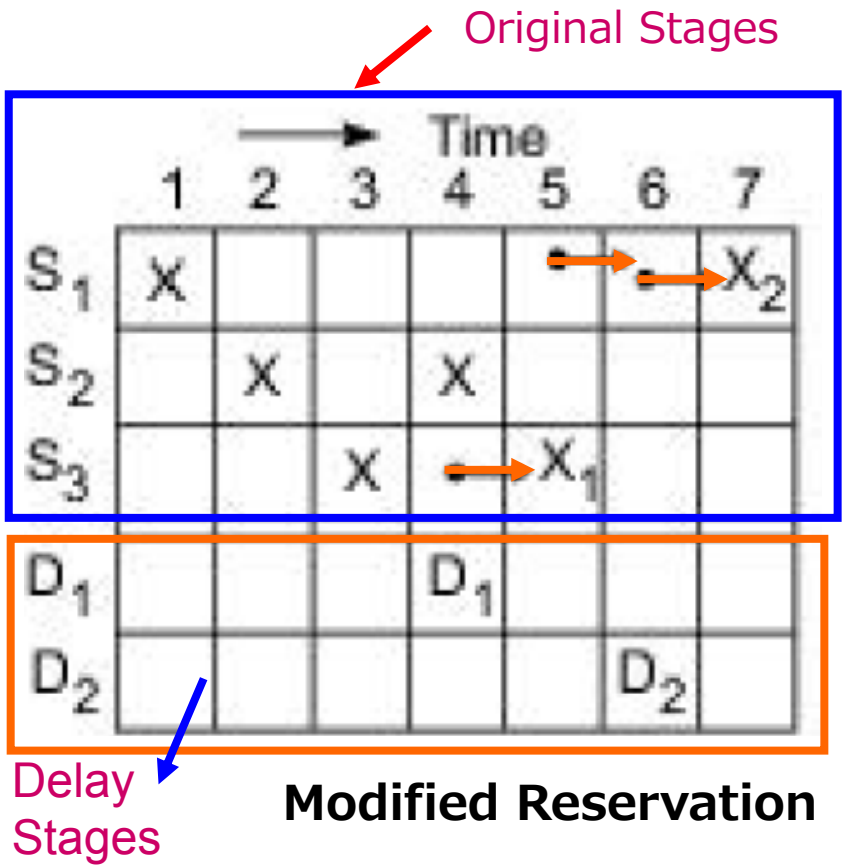
We insert delays (D1 & D2) in two of the branches. While going from S3 to S1, D2 is inserted, and from S3 to S3, D1 is inserted. In the original reservation table, two additional dummy stages are introduced. Because of the delay some (X) marks will get shifted. After S3, first we go to D1, then we are going back to S3 (Earlier it goes directly to S3). Similarly, from S3 are going to D2, from D2 we are going to S1. Now reservation is a slightly complex. Forbidden latencies = (2, 6).



Insertion of two noncompute delay stages



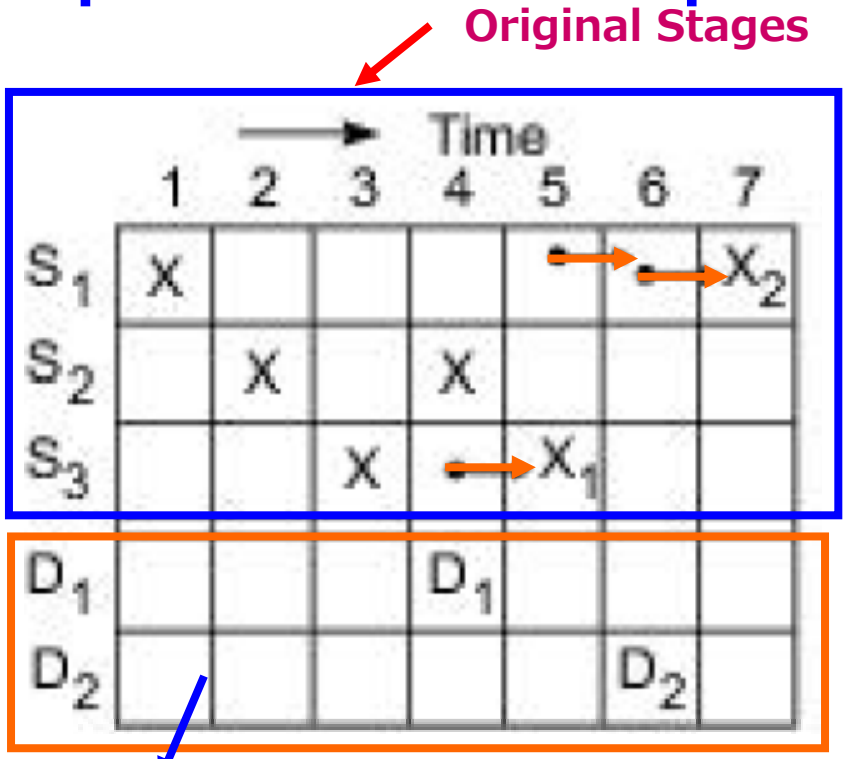
Reservation table and operations being delayed



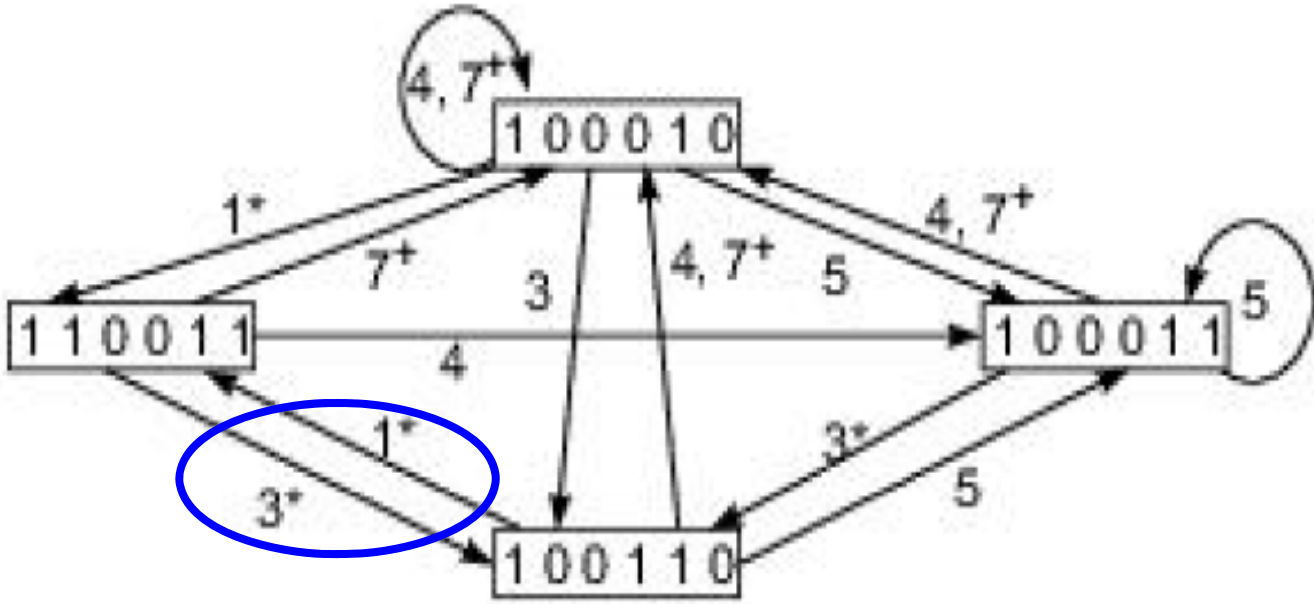
Modified Reservation

Nonlinear Pipeline

Pipeline Schedule Optimization



Modified Reservation



Modified State Diagram

Reduced MAL=(1+3)/2=2

The greedy cycles is the best cycle i.e. (1, 3); The average Value=(1+3)/2=2. Earlier MAL was 3, now by inserting the delay elements, we have achieved MAL of 2.

So, by inserting the dummy delay stages, the performance of a non-linear pipeline can be improved.

Assignments

Problem 1: For the following reservation tables,

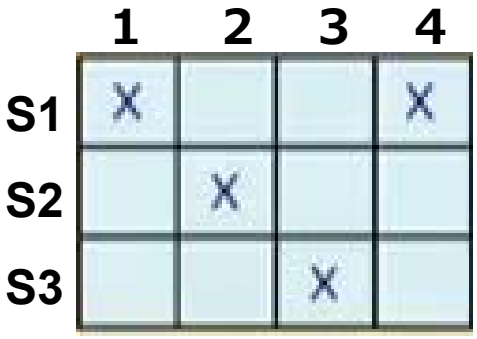
a) What are the forbidden latencies?

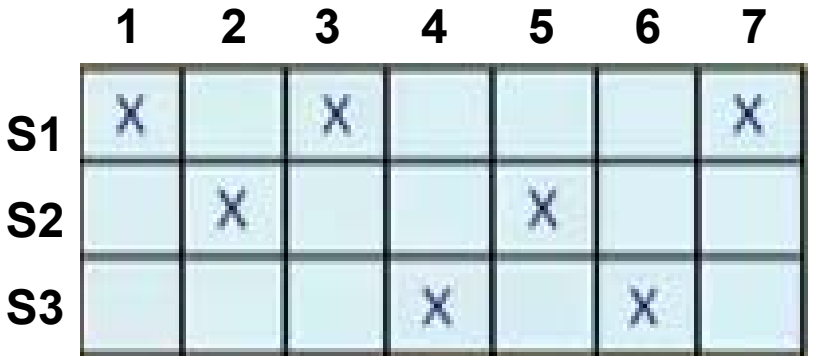
b) Show the state transition diagram

c) List sll the simple cycles and greedy cycles.

d) Determine the optimal constant latency cycles, and the MAL.

e) Determine the pipeline throughput, for τ =2ons.





Assignments

Problem 2: consider the following reservation table for a four-stage pipeline with a clock cycle τ = 2 ns.

	1	2	3	4	5	6
S1	X					X
S2		X		X		
S3			X			
S4				X	X	

a) What are the forbidden latencies and the initial collision vector?

b) Draw the state transition diagram for scheduling the pipeline.

c) Determine the MAL associated with the shortest greedy cycle.

d) Determine the pipeline throughput corresponding to the MAL and given (τ).

e) Determine the lower bound on the MAL for this pipeline. Have you obtained the optimal latency from the above state diagram?

Assignments

Problem 3: You are allowed to insert one non-compute delay stage into the pipeline as shown in the reservation table to make a latency of i permissible in the shortest greedy cycle. The purpose is to yield a new reservation table leading to an optimal latency equal to die lower bound.

	1	2	3	4	5	6
S1	X					X
S2		X		X		
S3			X			
S4				X	X	

a)Show the modified reservation table with five rows and seven columns.

b) Draw the new state transition diagram for obtaining the optimal cycle.

c) List all the simple cycles and greedy cycles from the state diagram.

d) Prove that the new MAL equals the lower bound.

e) What is the optimal throughput of this pipeline? Indicate the percentage of throughput improvement compared with that obtained in the original Architecture.

Assignments

Problem 4: Consider the following pipeline reservation table. Let the pipeline clock period be τ = 2 ns.

	1	2	3	4
S1	X			X
S2		X		
S3			X	

a) What are the forbidden latencies?

b) Draw the state transition diagram.

c) List all the simple cycles and greedy cycles.

d) Determine the *o*ptima] constant latency cycle and the minimal average latency.

e) Determine the throughput of this pipeline.

Assignments

Problem 5: Consider the five-stage pipelined processor specified by the following reservation table.

	1	2	3	4	5	6
S1	X					X
S2		X			X	
S3			X			
S4				X		
S5		X				X

a) List the set of forbidden latencies and the collision vector.

b) Draw a state transition diagram showing all possible initial sequences (cycles) without causing a collision in the pipeline.

c) List all the simple cycles from the state diagram.

d) Identify the greedy cycles among the simple cycles.

e) What is the minimum average latency (MAL) of this pipeline?

f) What is the minimum allowed constant cycle in using this pipeline?

g) What will be the maximum throughput of this pipeline?

h) What will be the throughput if the minimum constant cycle is used?

Assignments

Problem 6: A non-pipelined processor P has a clock frequency of 250 MHz and an average CPI of 4. Processor M, an improved version of P is designed with a 5-stage linear instruction pipeline. However, due to latch delay and clock skew, the clock rate of M is only 200MHz.

a) If a program consisting of 5000 instructions are executed on both processors, what will be the speed up of processor M as compared to processor P.

b) Calculate the MIPS rate of each processor during the exceution of this particular program.