Unit-V-II

Pipelining

- Pipelining is a technique of decomposing a sequential process into suboperations with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments
- A pipeline can be visualized as a collection of processing segments through which binary information flows
- Each segment performs partial processing dictated by the way the task is partitioned
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline
- The final result is obtained after data have passed through all segments

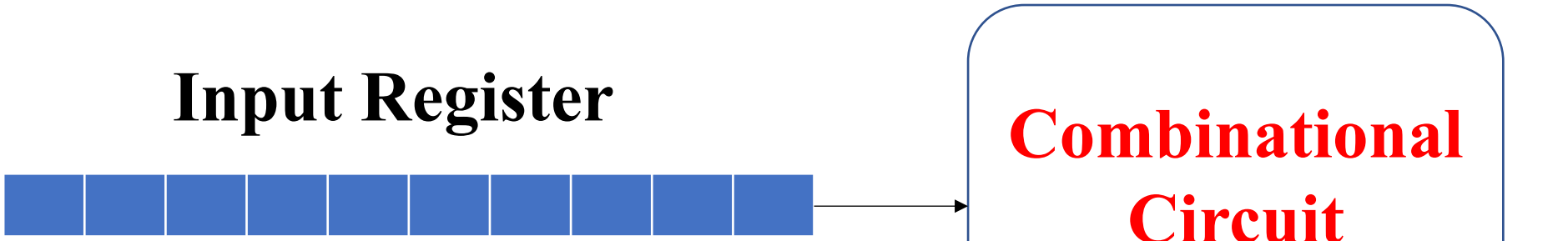
Pipeline ------- Flow of Information

Analogous to an industry assembly line

Pipelining

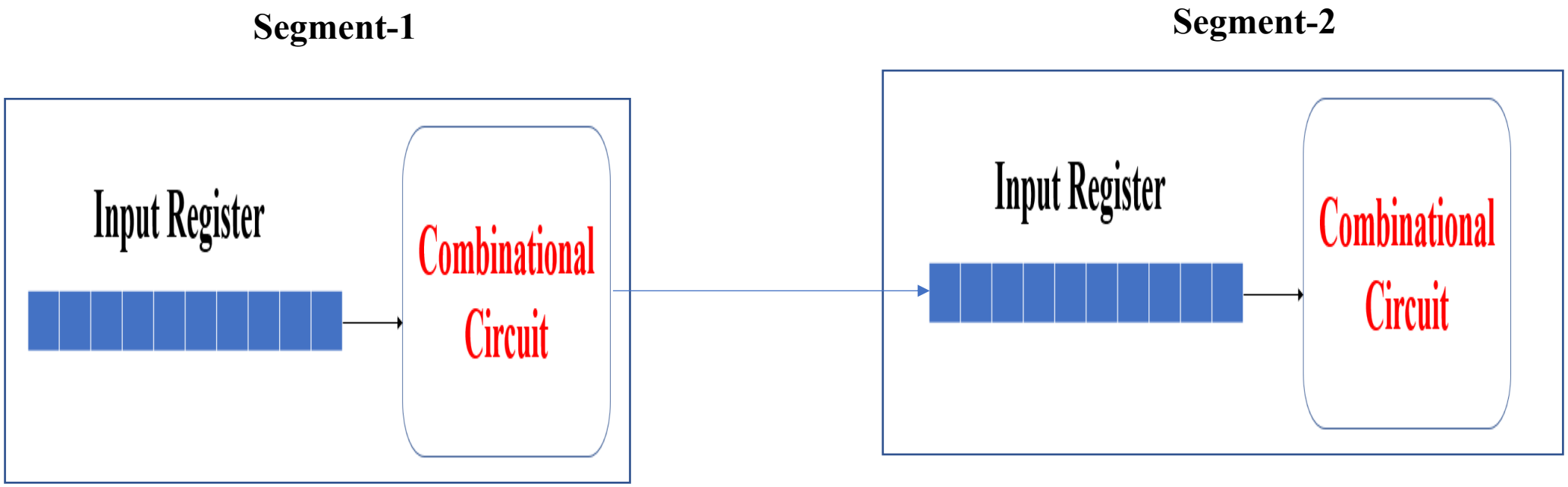
- It is the characteristic of pipelines that several computations can be in progress in distinct segments at the same time
- The overlapping of computation is made possible by associating a register with each segment in the pipeline
- The registers provide isolation between each segment so that each can operate on distinct data simultaneously

Each Segment in a Pipeline



Register holds the data

Combinational circuit performs suboperation in the particular segment



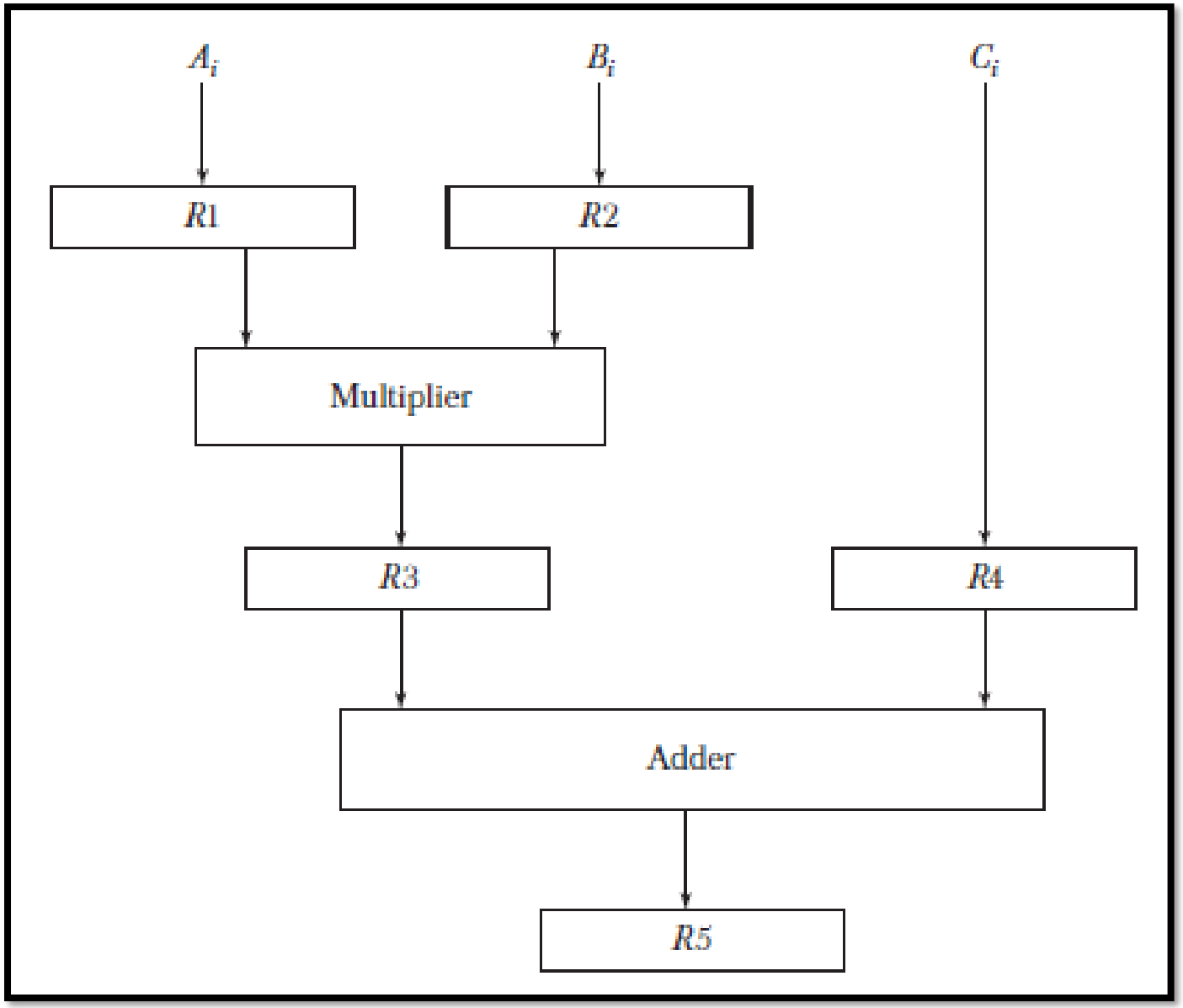
- The output of the combinational circuit in a given segment is applied to the input register of the next segment
- A clock is applied to all registers after **enough time** has elapsed to perform all segment activity
- In this way the information flows through the pipeline one step at a time

$$A_i * B_i + C_i \quad \text{for } i = 1, 2, 3, \ldots, 7$$

$$R1 \leftarrow A_i, \quad R2 \leftarrow B_i$$

$$R3 \leftarrow R1 * R2, \quad R4 \leftarrow C_i$$

$$R5 \leftarrow R3 + R4$$



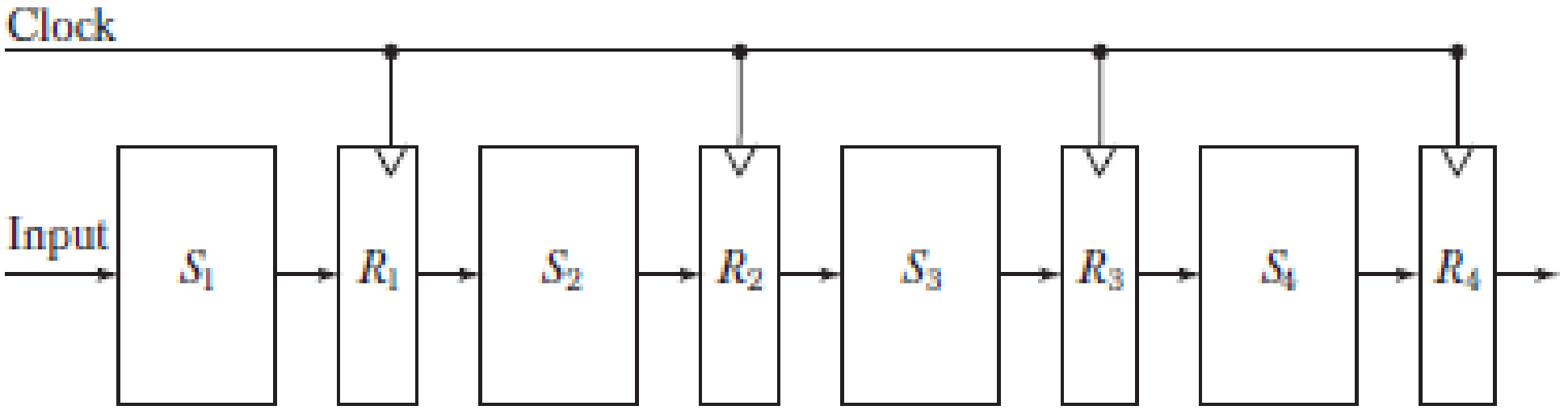
Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

When no more input data are available the clock must continue until the last output emerges out of the pipeline

General Considerations

- Any operation that can be decomposed into a sequence of suboperations about the same complexity can be implemented by a pipeline processor
- The technique is efficient for those applications that need to repeat the same task many times with different sets of data

Four-Segment Pipeline

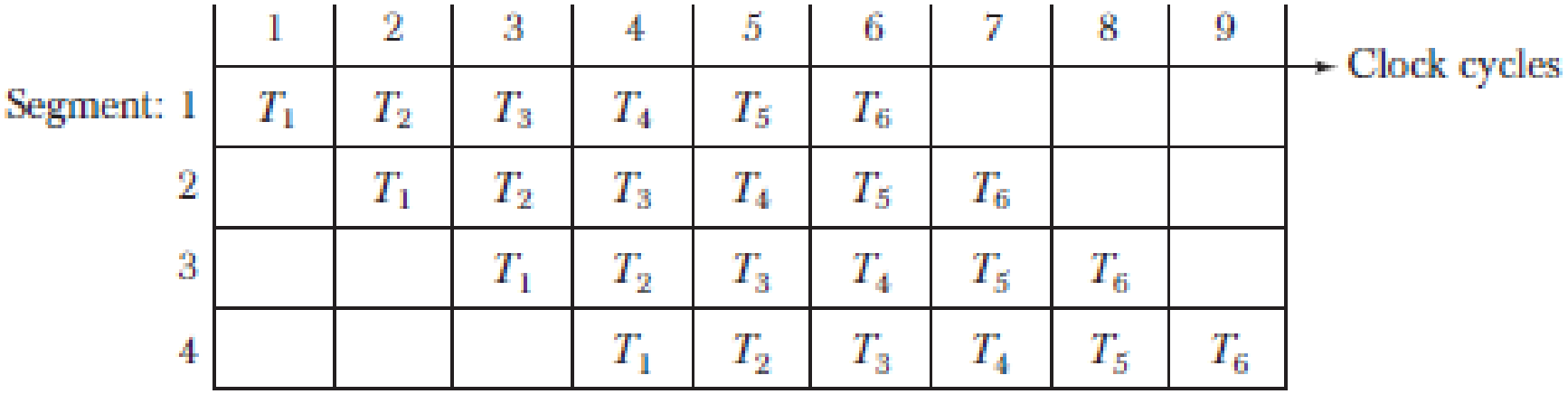


The segments are separated by registers that had intermediate results between the stages

Task

- We define Task as the total operation performed going through all the segments in the pipeline

Space-Time Diagram



The behavior of the pipeline can be illustrated with a space-time diagram.

This is a diagram that shows the segment utilization as a function of time.

Figure shows that six tasks are executed in four segments.

Note

Segment:	1	2	3	4	5	6	7	8	9	→ Clock cycles
1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

No matter how many segments there are in the system once the pipeline is full it takes only one clock period to obtain an output

Consider K-Segment Pipeline,

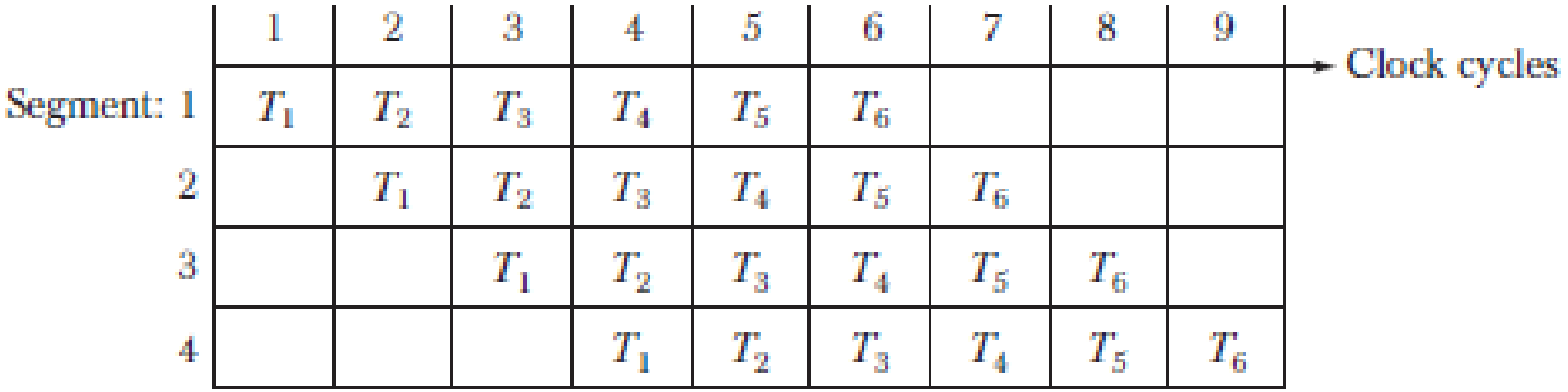
Clock cycle time= t_p **(How you decide this?)**

Total Tasks = n

The first task T_1 requires time= K * t_p

The remaining tasks time = (n-1) * t_p

To complete n tasks using a K-segment pipeline requires K + (n-1) Clock cycles



Calculate Number of Clock cycles?

Answer

Number of segments = 4

Number of tasks = 6

Number of clock cycles = 4 + (6-1)= 9 Clock cycles

Speed up

Consider a non-pipeline unit that performs same operation and takes a time equal to t_n to complete each task

Total time required for n tasks = n * t_n

The speedup of a pipeline processing over an equivalent non-pipeline processing is defined by ratio

$$S = \frac{n * t_n}{(K + (n-1)) * t_p}$$

Speed up

$$S = \frac{n * t_n}{(K + (n-1)) * t_p}$$

As number of tasks increases

$n >> K-1$ and $K+n-1 \approx n$

Under this condition, $S = \frac{n * t_n}{n * t_p} = \frac{t_n}{t_p}$

Assume, $t_n = K * t_p$

(Assuming that the time it takes to process a task is the same in pipeline and non-pipeline circuits)

$$S = \frac{K * t_p}{t_p} = K$$

This shows that theoretical maximum speedup that a pipeline can provide is K, Where K is the number of segments in the pipeline

Arithmetic Pipeline and Instruction Pipeline

There are two areas of computer design where the pipeline organization is applicable.

An ***arithmetic pipeline*** divides an arithmetic operation into suboperations for execution in the pipeline segments.

An ***instruction pipeline*** operates on a stream of instructions by overlapping the fetch, decode, and execute phases of the instruction cycle.

Arithmetic Pipeline

Arithmetic Pipeline

- Pipeline arithmetic units are usually found in very high speed computers.
- They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.
- We will now show an **example of a pipeline unit for floating-point addition and subtraction.**

Pipeline for floating-point addition and subtraction

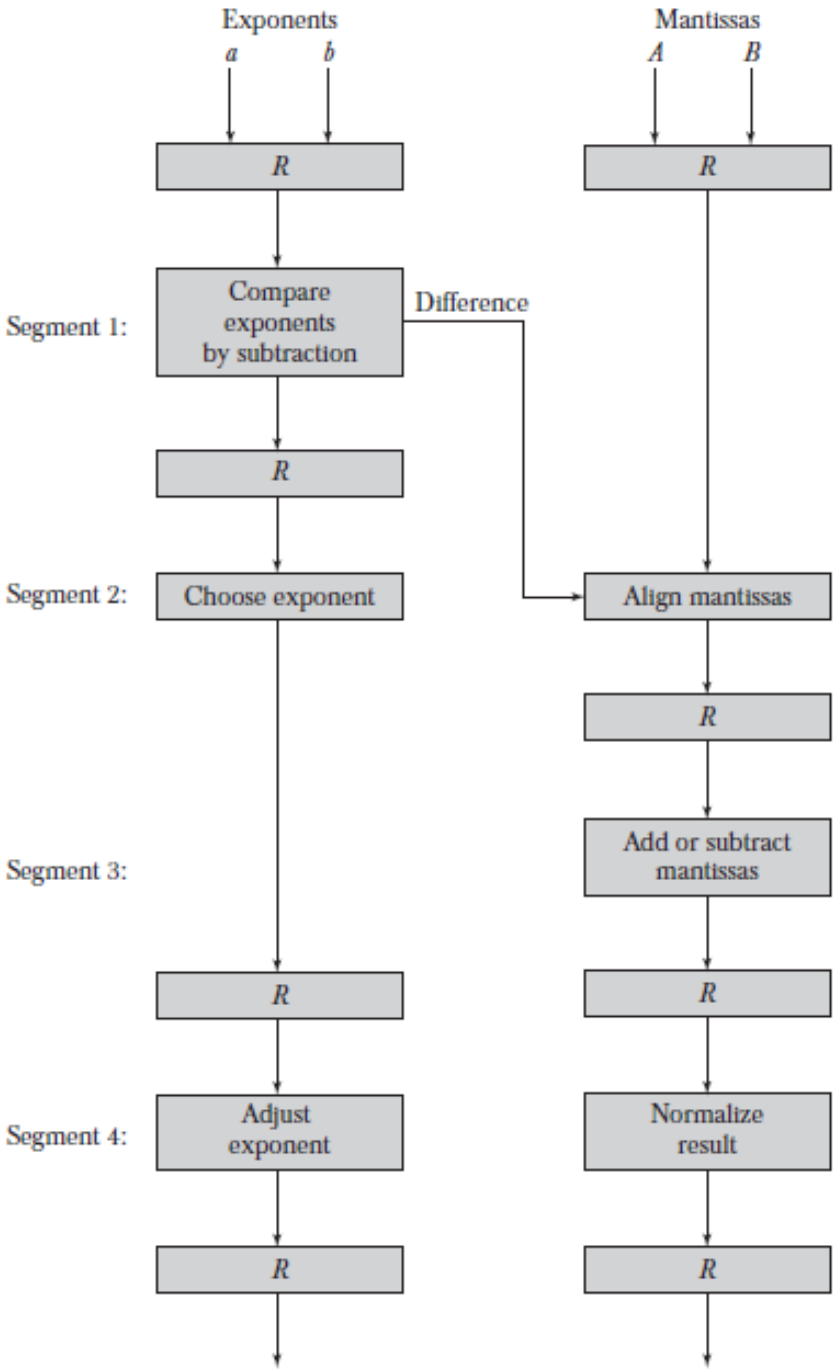
The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

$$X = A * 2^a$$
$$Y = B * 2^b$$

A and *B* are two fractions that represent the mantissas and *a* and *b* are the exponents.

The floating-point addition and subtraction can be performed in four segments, as shown
The registers labeled *R* are placed between the segments to store intermediate results.
The suboperations that are performed in the four segments are:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.



Explanation

- The exponents are compared by subtracting them to determine their difference.
- The larger exponent is chosen as the exponent of the result.
- The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right.
- This produces an alignment of the two mantissas.
- It should be noted that the shift must be designed as a combinational circuit to reduce the shift time.
- The two mantissas are added or subtracted in segment 3.
- The result is normalized in segment 4. When an overflow occurs, the mantissa of the sum or difference is shifted right and the exponent incremented by one.
- If an underflow occurs, the number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent.

Numerical Example

The following numerical example may clarify the suboperations performed in each segment. For simplicity, we use decimal numbers, although Fig. refers to binary numbers. Consider the two normalized floating-point numbers:

$$X = 0.9504 \times 10^3$$
$$Y = 0.8200 \times 10^2$$

The two exponents are subtracted in the first segment to obtain $3 - 2 = 1$. The larger exponent 3 is chosen as the exponent of the result. The next segment shifts the mantissa of Y to the right to obtain

$$X = 0.9504 \times 10^3$$
$$Y = 0.0820 \times 10^3$$

This aligns the two mantissas under the same exponent. The addition of the two mantissas in segment 3 produces the sum

$$Z = 1.0324 \times 10^3$$

The sum is adjusted by normalizing the result so that it has a fraction with a nonzero first digit. This is done by shifting the mantissa once to the right and incrementing the exponent by one to obtain the normalized sum.

$$Z = 0.10324 \times 10^4$$

Analysis

The comparator, shifter, adder-subtractor, incrementer, and decrementer in the floating-point pipeline are implemented with combinational circuits. Suppose that the time delays of the four segments are $t_1 = 60$ ns, $t_2 = 70$ ns, $t_3 = 100$ ns, $t_4 = 80$ ns, and the interface registers have a delay of $t_r = 10$ ns. The clock cycle is chosen to be $t_p = t_3 + t_r = 110$ ns. An equivalent non-pipeline floatingpoint adder-subtractor will have a delay time $t_n = t_1 + t_2 + t_3 + t_4 + t_r = 320$ ns. In this case the pipelined adder has a speedup of $320/110 = 2.9$ over the nonpipelined adder.