

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 the data have passed through all segments. The name "pipeline" implies a

flow of information analogous to an industrial assembly line. It is 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.

Perhaps the simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational circuit. The register holds the data and the combinational circuit performs the 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.

The pipeline organization will be demonstrated by means of a simple example. Suppose that we want to perform the combined multiply and add operations with a stream of numbers.

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

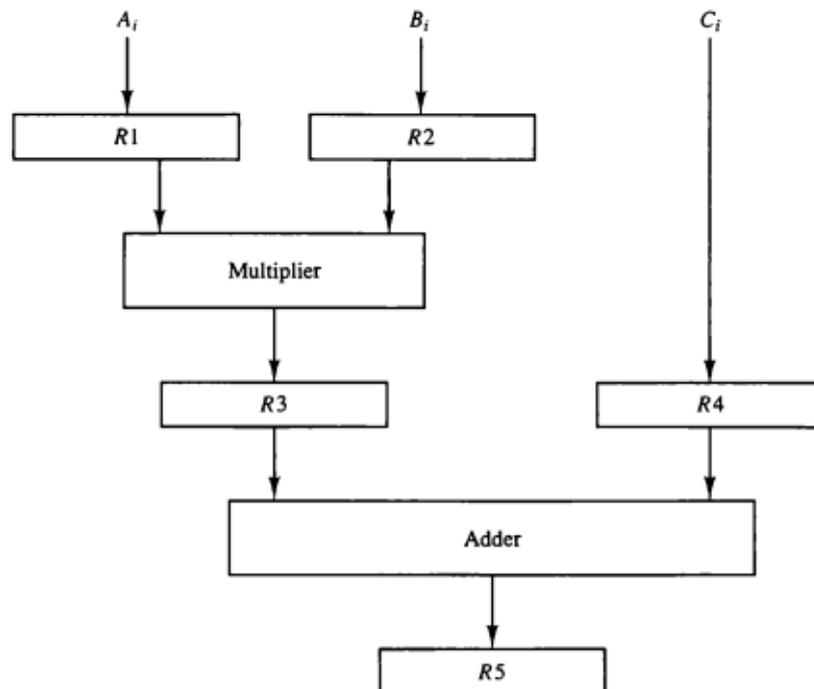
Each suboperation is to be implemented in a segment within a pipeline.

Each segment has one or two registers and a combinational circuit as shown in Fig. 2. R1 through R5 are registers that receive new data with every clock pulse. The multiplier and adder are combinational circuits.

The suboperations performed in each segment of the pipeline are as follows:

$R1 \leftarrow A_i, R2 \leftarrow B_i$	Input A_i and B_i
$R3 \leftarrow R1 * R2, R4 \leftarrow C_i$	Multiply and input C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table 1. The first clock pulse transfers A_i and B_i into R1 and R2.



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	—	—	—	—	$A_7 * B_7 + C_7$

The second clock pulse transfers the product of R1 and R2 into R3 and Q into R4. The same clock pulse transfers A2 and B2 into R1 and R2. The third clock pulse operates on all three segments simultaneously. It places A3 and B3 into R1 and R2, transfers the product of R1 and R2 into R3, transfers C2 into R4, and places the sum of R3 and R4 into R5. It takes three clock pulses to fill up the pipe and retrieve the first output from R5. From there on, each clock produces a new output and moves the data one step down the pipeline. This happens as long as new input data flow into the system. When no more input data are available, the clock must continue until the last output emerges out of the 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. A pipeline multiplier is essentially an array multiplier as described in Figure below, with special adders designed to minimize the carry propagation time through the partial products. Floating-point operations are easily decomposed into suboperations.

We will now show an example of a pipeline unit for floating-point addition and subtraction. The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

$$X = A \times 2^a$$

$$Y = B \times 2^b$$

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 in Fig. 9-6. 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.

This follows the procedure outlined in the flowchart of Fig. 3 but with some variations that are used to reduce the execution time of the suboperations.

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.

The following numerical example may clarify the suboperations performed in each segment. For simplicity, we use decimal numbers, although Fig. 9-6 refers to binary numbers. Consider the two normalized floating-point numbers:

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

$$Y = 0.8200 \times 10^2$$

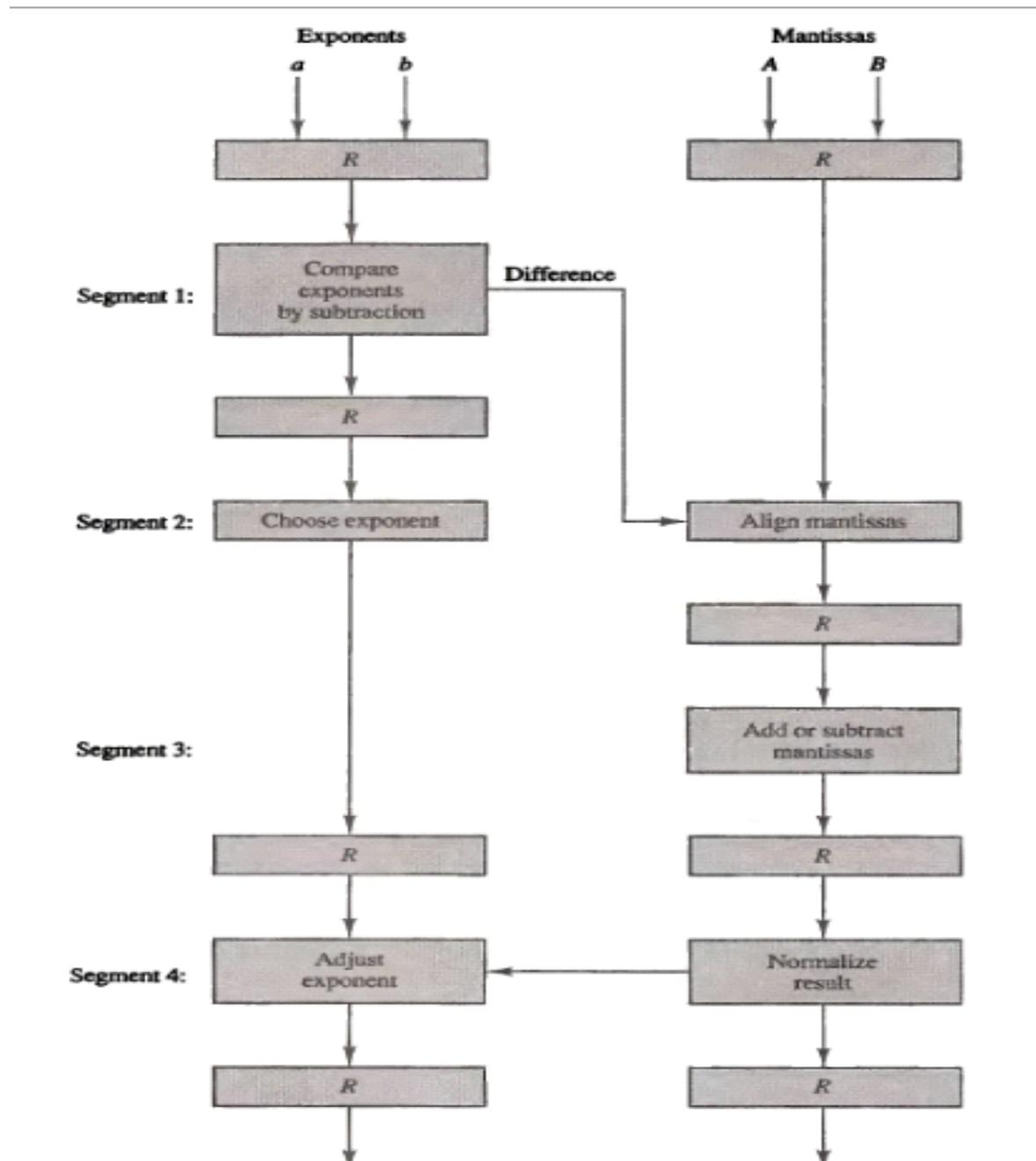
The two exponents are subtracted in the first segment to obtain $3-2 = 1$. The larger exponent 3 is chosen as the exponent of the result. The next segment shifts the mantissa of Y to the right to obtain

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

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

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.

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