Lecture – 12

Pipeline Processing

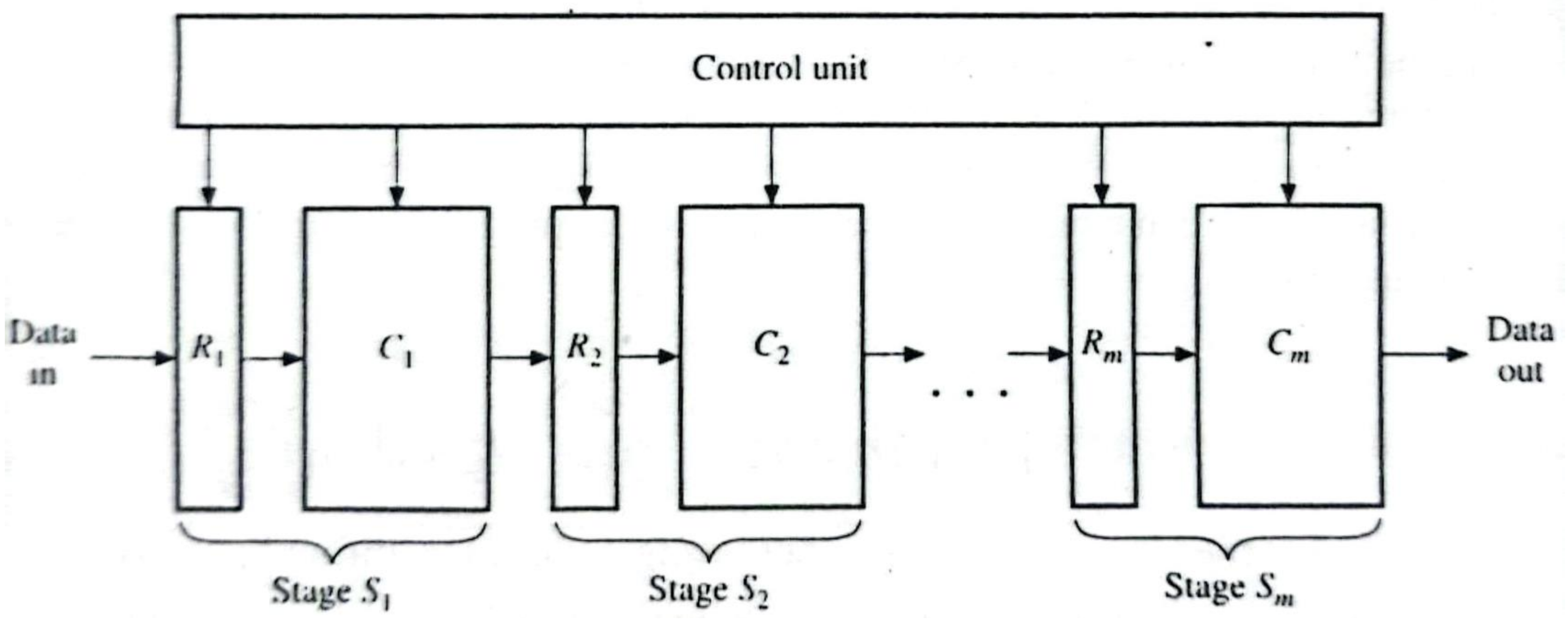
- Pipelining is a general technique for increasing processor throughput without requiring large amounts of extra hardware.
- It is applied to the design of the complex datapath units such as multipliers and floating-point adders.
- It is also used to improve the overall throughput of an instruction set processor.

Pipeline Processor

- A pipeline processor consists of a sequence of *m* data-processing circuits, called **stages** or **segments**, which collectively perform a single operation on a stream of data operands passing through them.

- Some processing take place in each stage, but a final result is obtained only after an operand set has passed through the entire pipeline.

Structure of a Pipeline Processor



S_i = Stage, R_i = Multiword input register

C_i = Datapath Circuit, D_{i-1} = Received by R_i from S_{i-1}.

- In each clock period, every stage transfers its previous results to the next stage and computes a new set of results.

Advantage of Pipeline Processor

- An *m*-stage pipeline can simultaneously process up to *m* independent sets of data operands.
- These data sets move through the pipeline stage by stage so that when the pipeline is full, *m* separate operations are being executed concurrently, each in a different stage.
- A new final result emerges from the pipeline every clock cycle.

Pipeline Processor

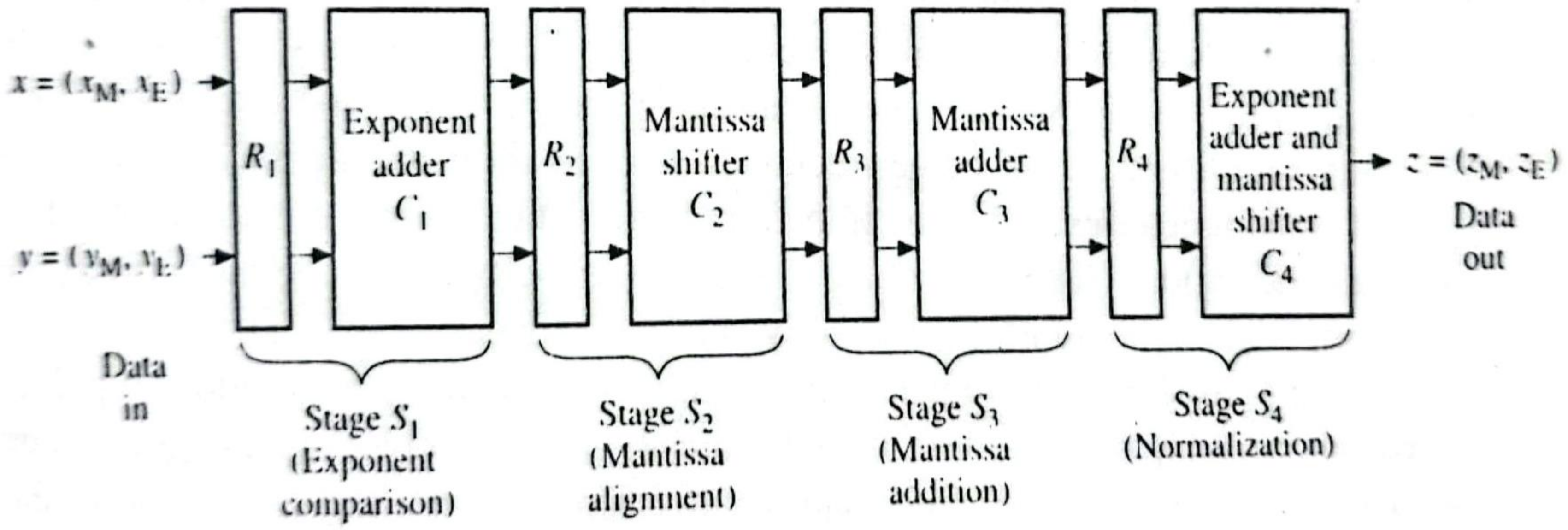
- An m-stage pipeline can simultaneously process up to m independent sets of data operands.
- Suppose that each stage of the *m* stage pipeline takes *T* seconds to perform its local suboperation and store its results.
- Then *T* is the pipeline's clock period.
- The *delay* or *latency* of the pipeline is *mT*.
- The *throughput* (maximum number of operations completed per second) of the pipeline is *1/T.*
- The *CPI* is one.
- Any operation that can be decomposed into a sequence of sub-operations of about the same complexity can be realized by a pipeline processor.

Four-stage Floating-point Adder Pipeline

- The addition of two normalized floating-point numbers x and y can be implemented by four step sequence:

  1. compare the exponents.
  2. Align the mantissa.
  3. Add the mantissa.
  4. Normalize the result.

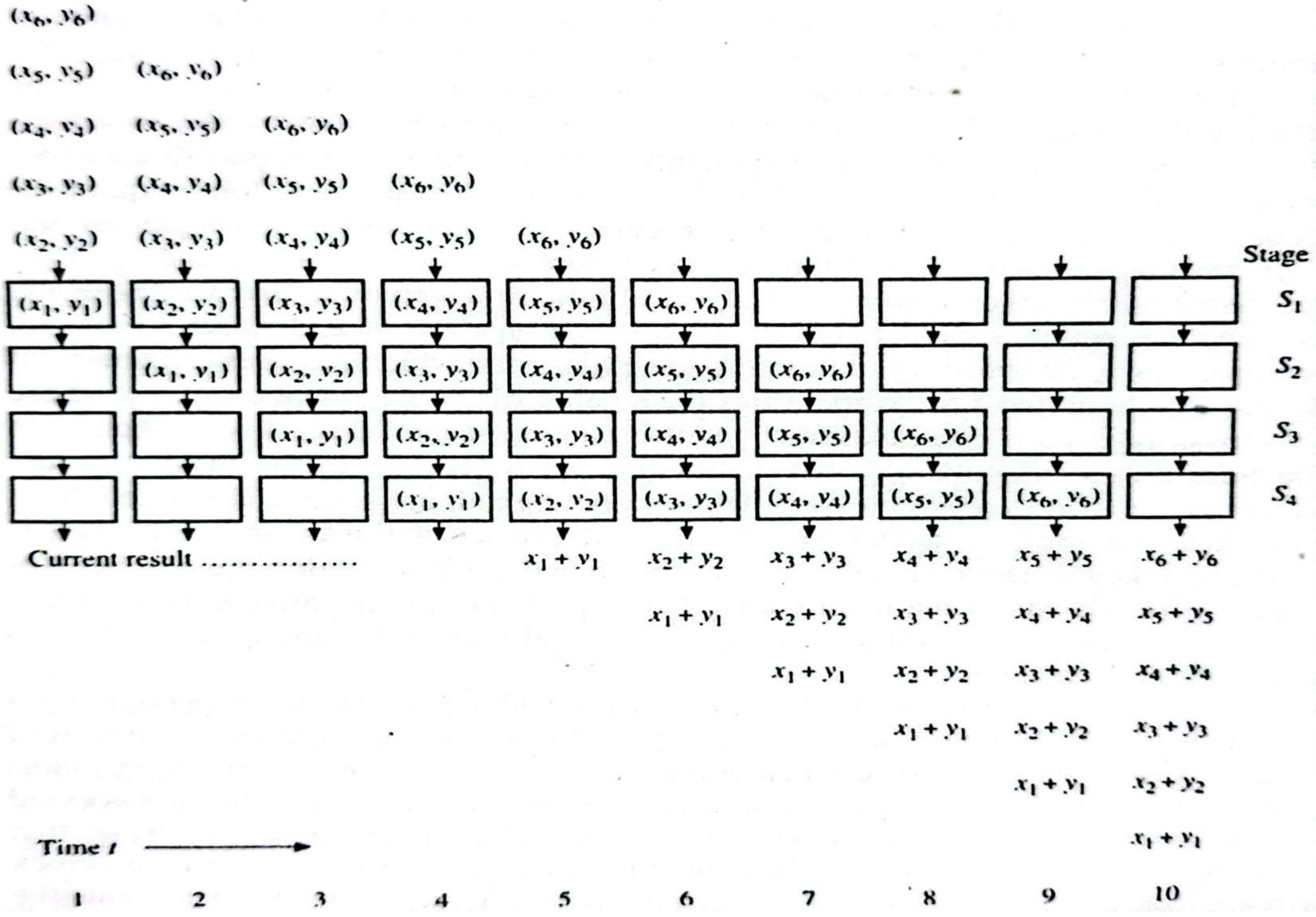
Four-stage Floating-point Adder Pipeline


$x = (x_M, x_E)$
$y = (y_M, y_E)$
Data in
R_1
Exponent adder C_1
R_2
Mantissa shifter C_2
R_3
Mantissa adder C_3
R_4
Exponent adder and mantissa shifter C_4
$z = (z_M, z_E)$
Data out
Stage S_1 (Exponent comparison)
Stage S_2 (Mantissa alignment)
Stage S_3 (Mantissa addition)
Stage S_4 (Normalization)

Four-stage Floating-point Adder Pipeline

- It takes *4T* time to compute a single sum.
- When all the four stages have been filled with data, a new sum can be computed in every *T* seconds.
- So *N* consecutive additions can be done in *(N+3)T* seconds.
- So the four-stage pipeline’s speedup is $S(4) = 4N / (N+3)$
- For large N, $S(4) \approx 4$.

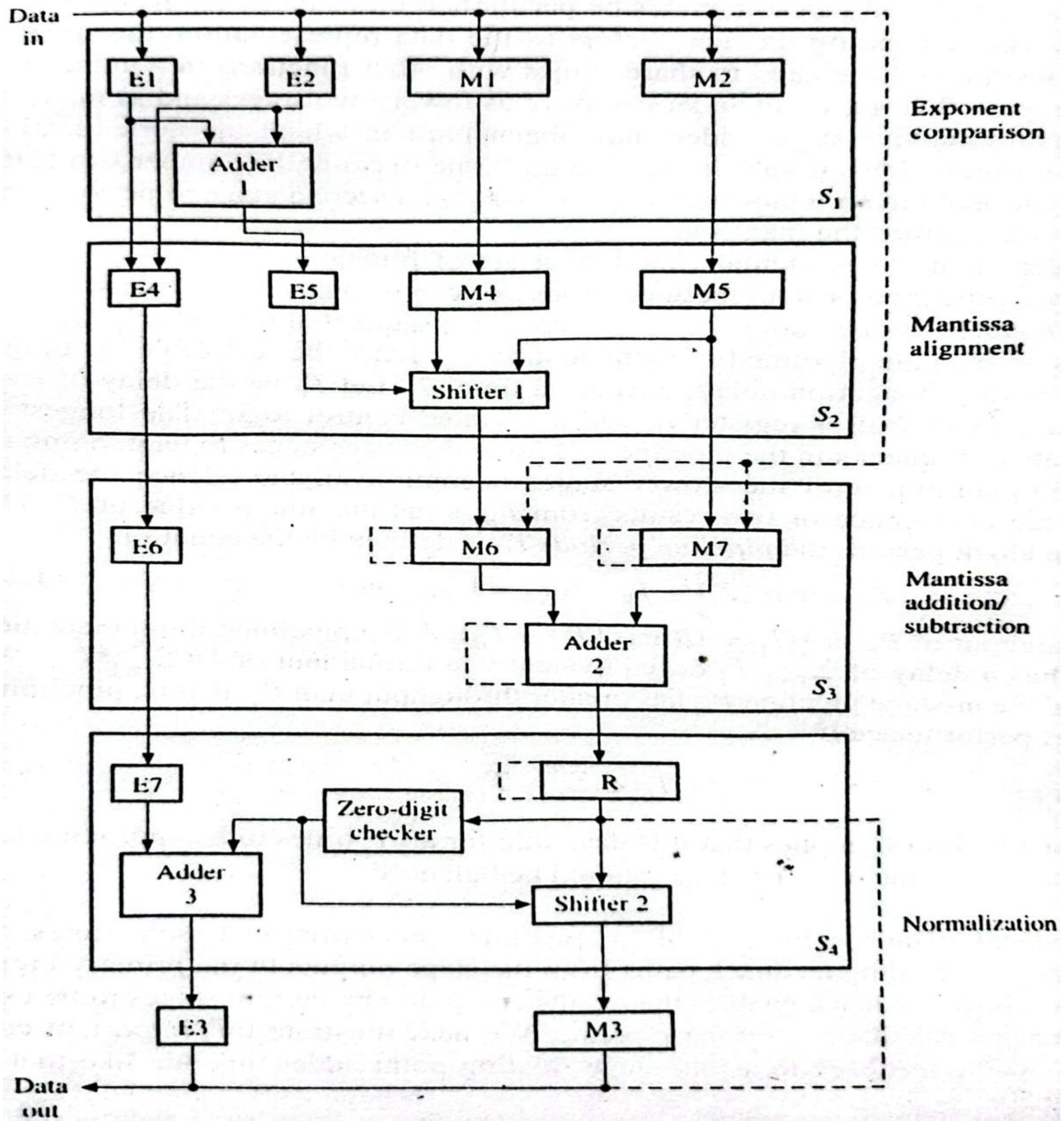
Operations of the Four-stage Floating-point Adder Pipeline



Pipeline Design

- Designing a pipelined circuit for a function involves first finding a suitable multistage sequential algorithm to compute the given function.
- This algorithm steps should be balanced in the sense that they should all have roughly the same execution time.
- Fast buffer registers are placed between the stages to transfer data from stage to stage without interfering with one another.
- The buffers are designed to be clocked.

Pipelined Version of the Floating-point Adder



Data in
E1
E2
M1
M2
Adder 1
Exponent comparison
S_1
E4
E5
M4
M5
Shifter 1
Mantissa alignment
S_2
E6
M6
M7
Adder 2
Mantissa addition/ subtraction
S_3
E7
R
Zero-digit checker
Adder 3
Shifter 2
Normalization
S_4
E3
M3
Data out