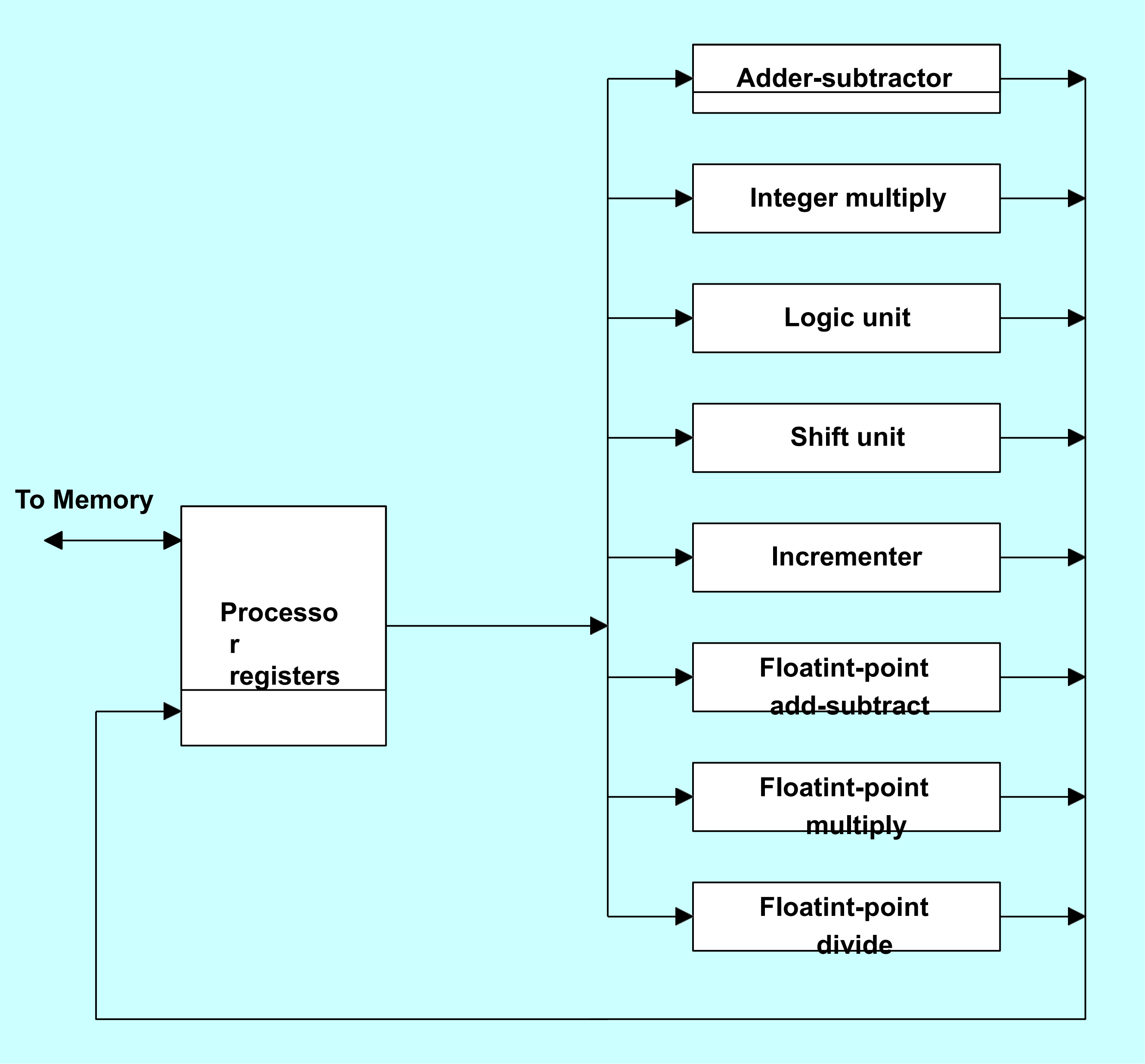
Arithmetic Pipeline

- Main topics in Pipeline processing is

  - **Arithmetic pipeline :**
    - fixed Arithmetic pipeline
    - floating point
  - **Vector processing : adder/multiplier pipeline**
  - **Array processing : array processor**
    - Attached array processor
    - SIMD Array Processor

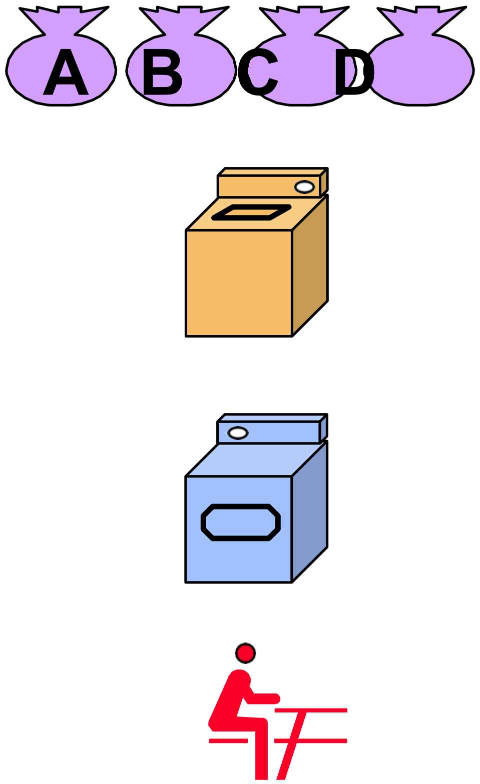
Parallel Processing

- *Simultaneous* data processing tasks for the purpose of increasing the computational speed
- Perform *concurrent* data processing to achieve faster execution time
- Multiple Functional Unit :
  - ***Separate the execution unit into eight functional units operating in parallel.***

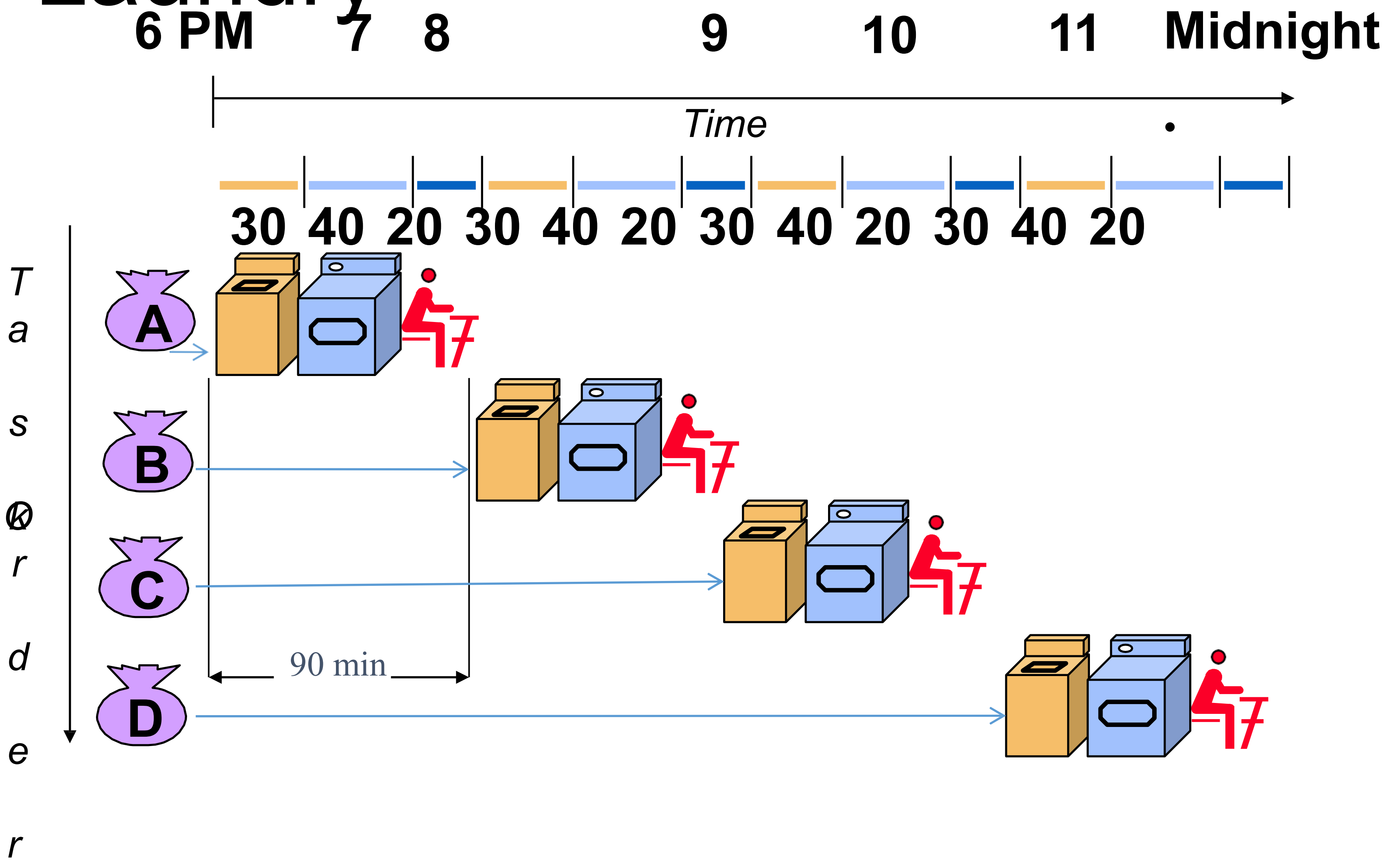


Pipelining: Laundry Example

- Small laundry has one washer, one dryer and one operator, it takes 90 minutes to finish one load:
  - Washer takes 30 minutes
  - Dryer takes 40 minutes
  - “operator folding” takes 20 minutes



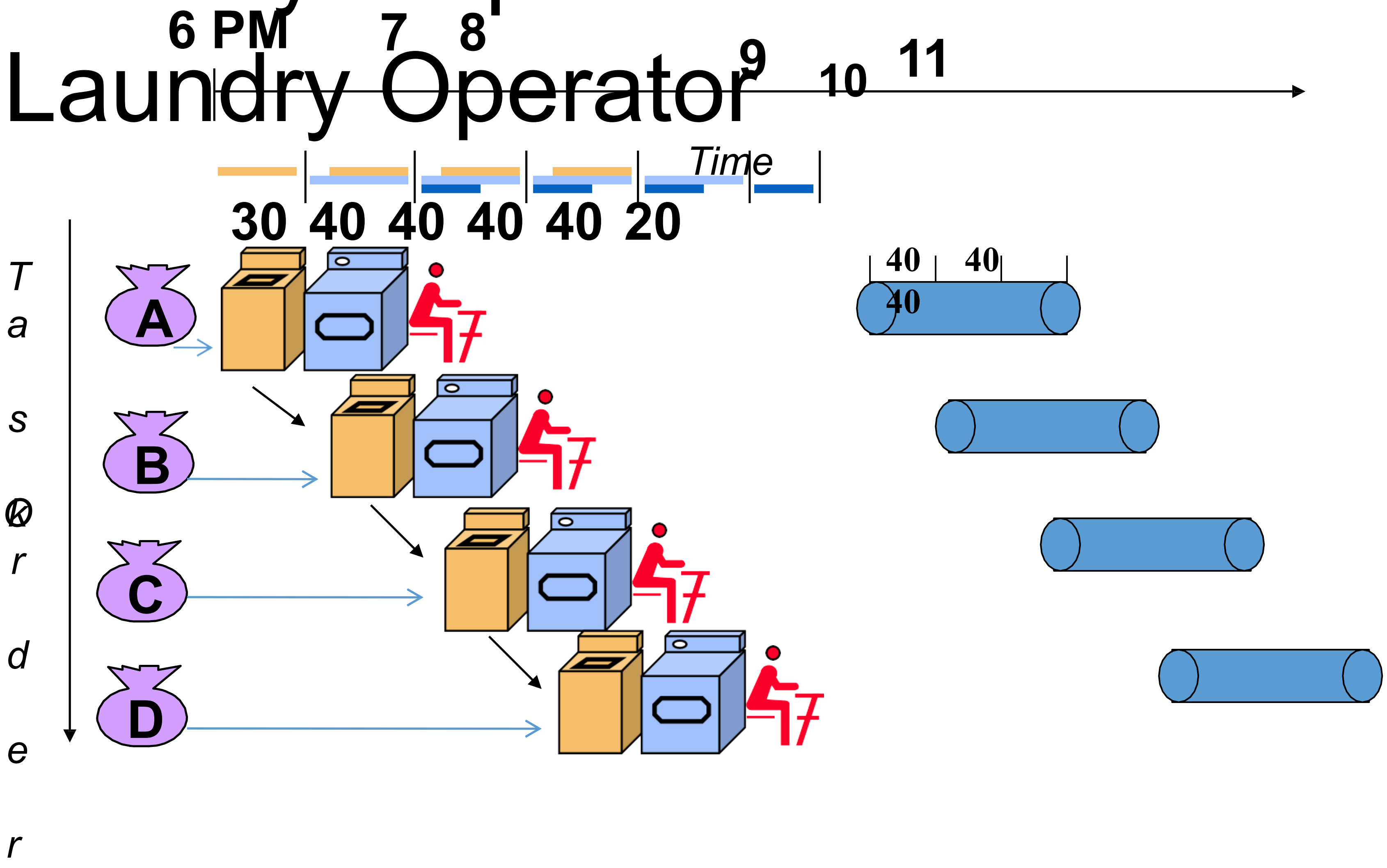
Sequential Laundry



This operator scheduled his loads to be delivered to the laundry every 90 minutes which is the time required to finish one load.

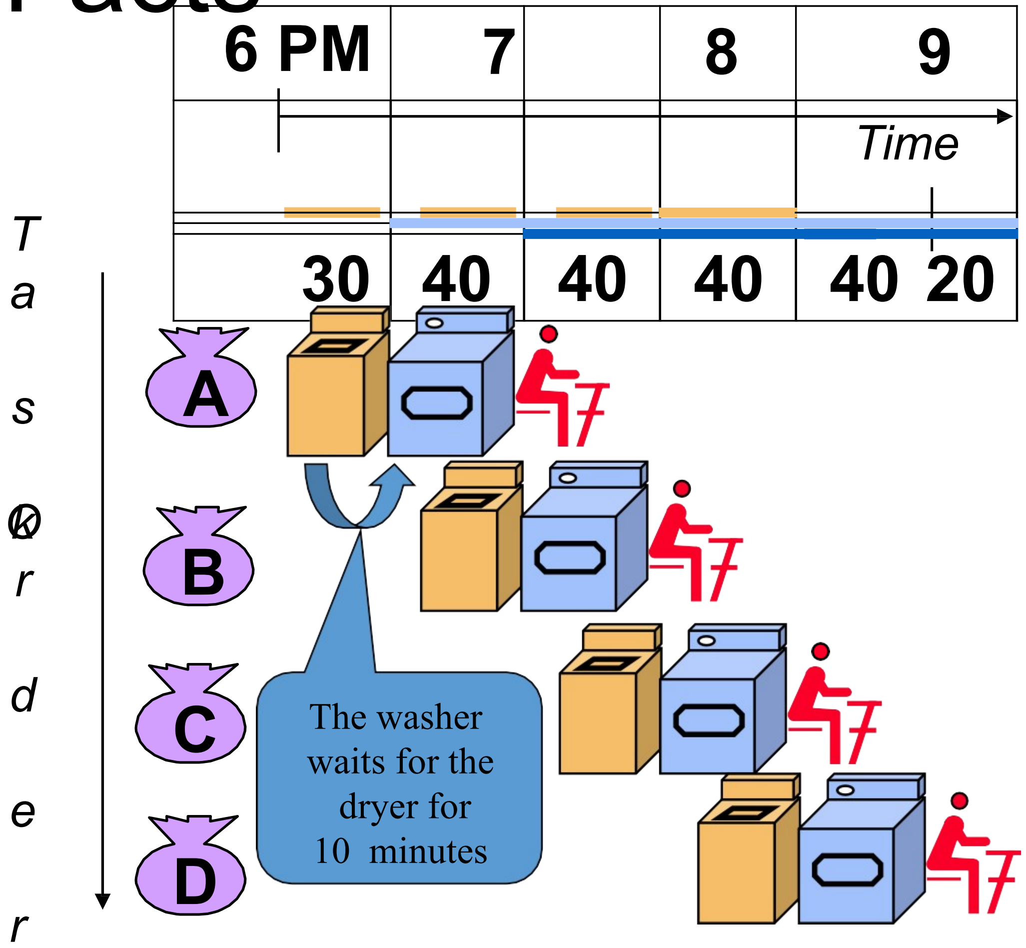
- In other words he will not start a new task unless he is already done with the previous task
- The process is sequential. Sequential laundry takes 6 hours for 4 loads

Efficiently scheduled laundry: Pipelined Laundry Operator



- Another operator asks for the delivery of loads to the laundry every 40 minutes!?.
- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Facts



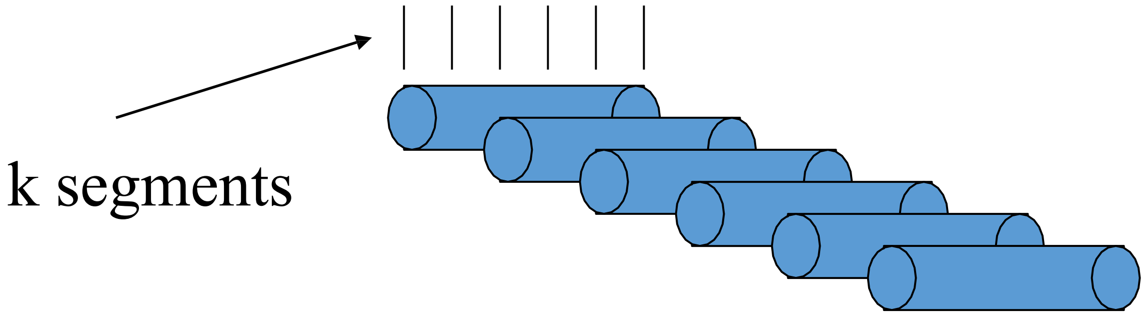
- Multiple tasks operating simultaneously
- Pipelining doesn't help latency (response time) of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Potential speedup = Number of

  pipe stages
- Unbalanced lengths of pipe stages reduces

Pipelining

Decomposing a sequential process into suboperations
Each subprocess is executed in a special dedicated segment concurrently

- Instruction execution is divided into k segments or stages
  - Instruction exits pipe stage k-1 and proceeds into pipe stage k
  - All pipe stages take the same amount of time; called **one processor cycle**
  - Length of the processor cycle is determined by the slowest pipe stage



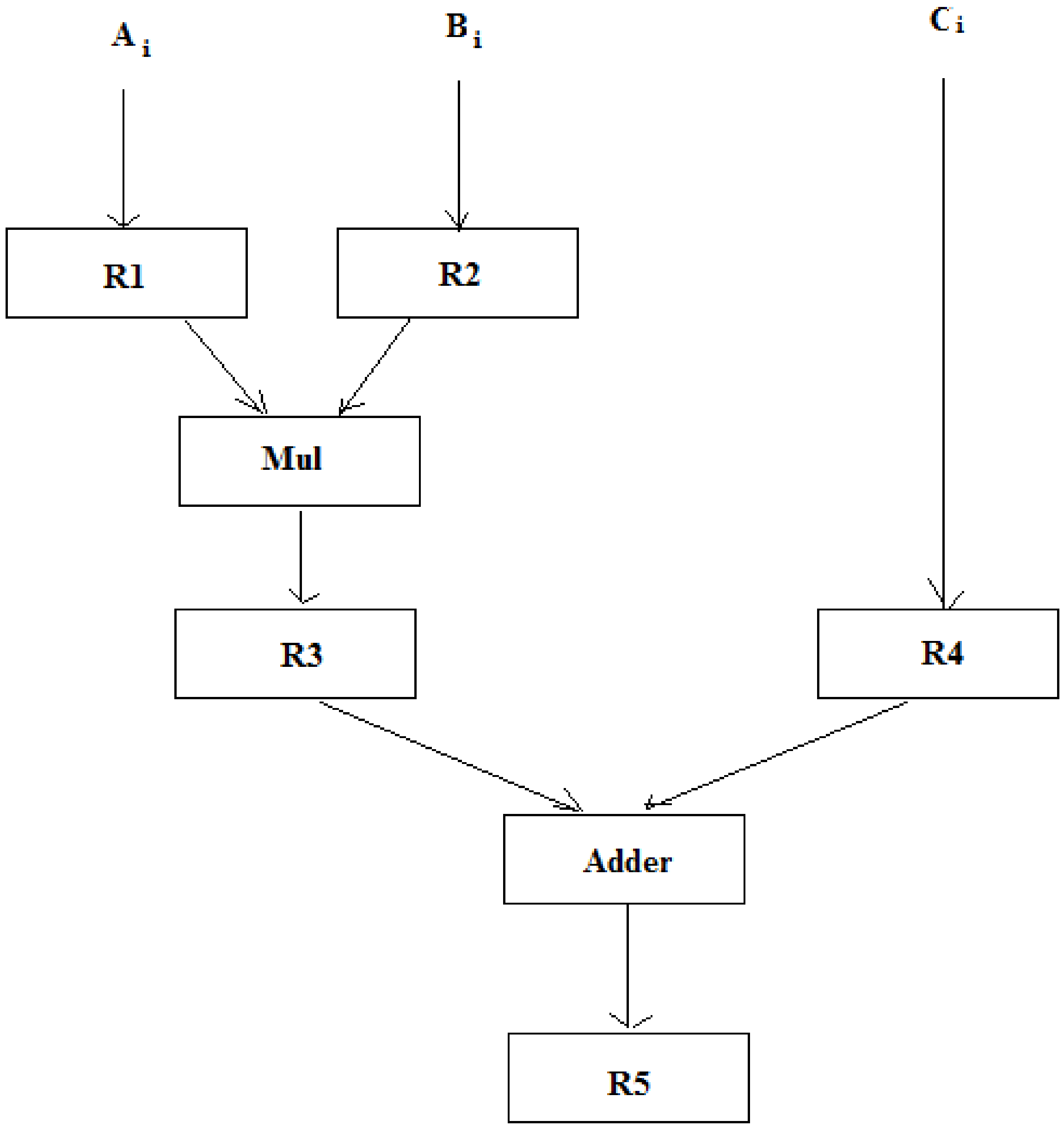
Pipelining

- Suppose we want to perform the combined multiply and add operations with a stream of numbers:
- $A_i * B_i + C_i$ for i =1,2,3,…,7

- The sub operations performed in each segment of the pipeline are as follows:
  - $R1 \leftarrow A_i$ $R2 \leftarrow B_i$
  - $R3 \leftarrow R1 * R2$ $R4 \leftarrow C_i$
  - $R5 \leftarrow R3 + R4$

TABLE 9-1 Content of Registers in Pipeline Example

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	—	—	—	—	$A_7 * B_7 + C_7$



Arithmetic Pipeline

- Pipeline arithmetic units are usually found in very high speed computers.
- Arithmetic pipelines are constructed for

  : simple fixed-point
  floating-point arithmetic operations.

- For implementing the arithmetic pipelines we generally use following two types of adder:
- i) **Carry propagation adder (CPA):** It adds two numbers such that carries generated in successive digits **are propagated**.
- ii) **Carry save adder (CSA):** It adds two numbers such that carries generated are
  not propagated rather these are saved in a carry vector.

Fixed Arithmetic pipeline

- We take the example of multiplication of fixed numbers.
- Two fixed-point numbers are added by the ALU using add and shift operations.
- This sequential execution makes the multiplication a slow process.
- Observe that this is the process of adding the multiple copies of shifted multiplicands as show below:

Fixed Arithmetic pipeline

$$
\begin{array}{ccccccccccc|l}
 & & & & & X_5 & X_4 & X_3 & X_2 & X_1 & X_0 & = X \\
 & & & & & Y_5 & Y_4 & Y_3 & Y_2 & Y_1 & Y_0 & = Y \\
\hline
 & & & & & X_5Y_0 & X_4Y_0 & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 & = P_1 \\
 & & & & X_5Y_1 & X_4Y_1 & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 & & = P_2 \\
 & & & X_5Y_2 & X_4Y_2 & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 & & & = P_3 \\
 & & X_5Y_3 & X_4Y_3 & X_3Y_3 & X_2Y_3 & X_1Y_3 & X_0Y_3 & & & & = P_4 \\
 & X_5Y_4 & X_4Y_4 & X_3Y_4 & X_2Y_4 & X_1Y_4 & X_0Y_4 & & & & & = P_5 \\
X_5Y_5 & X_4Y_5 & X_3Y_5 & X_2Y_5 & X_1Y_5 & X_0Y_5 & & & & & & = P_6 \\
\hline
\hline
\end{array}
$$

Now, we can identify the following stages for the pipeline:

- The first stage generates the partial product of the numbers, which form the six rows of shifted multiplicands.
- In the second stage, the six numbers are given to the two CSAs merging into four numbers.
- In the third stage, there is a single CSA merging the numbers into 3numbers.
- In the fourth stage, there is a single number merging three numbers into 2numbers.
- In the fifth stage, the last two numbers are added through a CPA to get the final product.

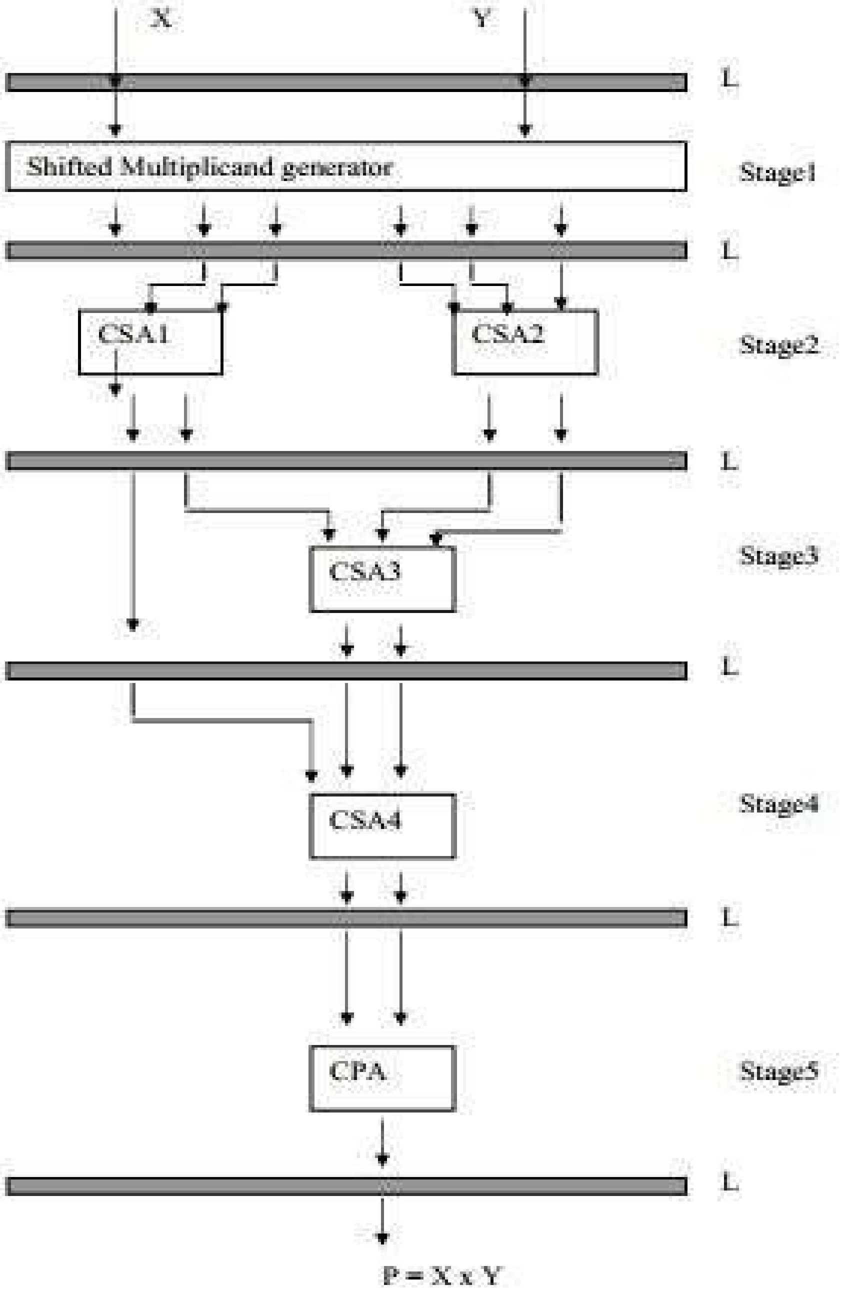


Figure 5: Arithmetic pipeline for Multiplication of two 6-digit fixed numbers

Floating point operations.

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

Mantiss

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

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

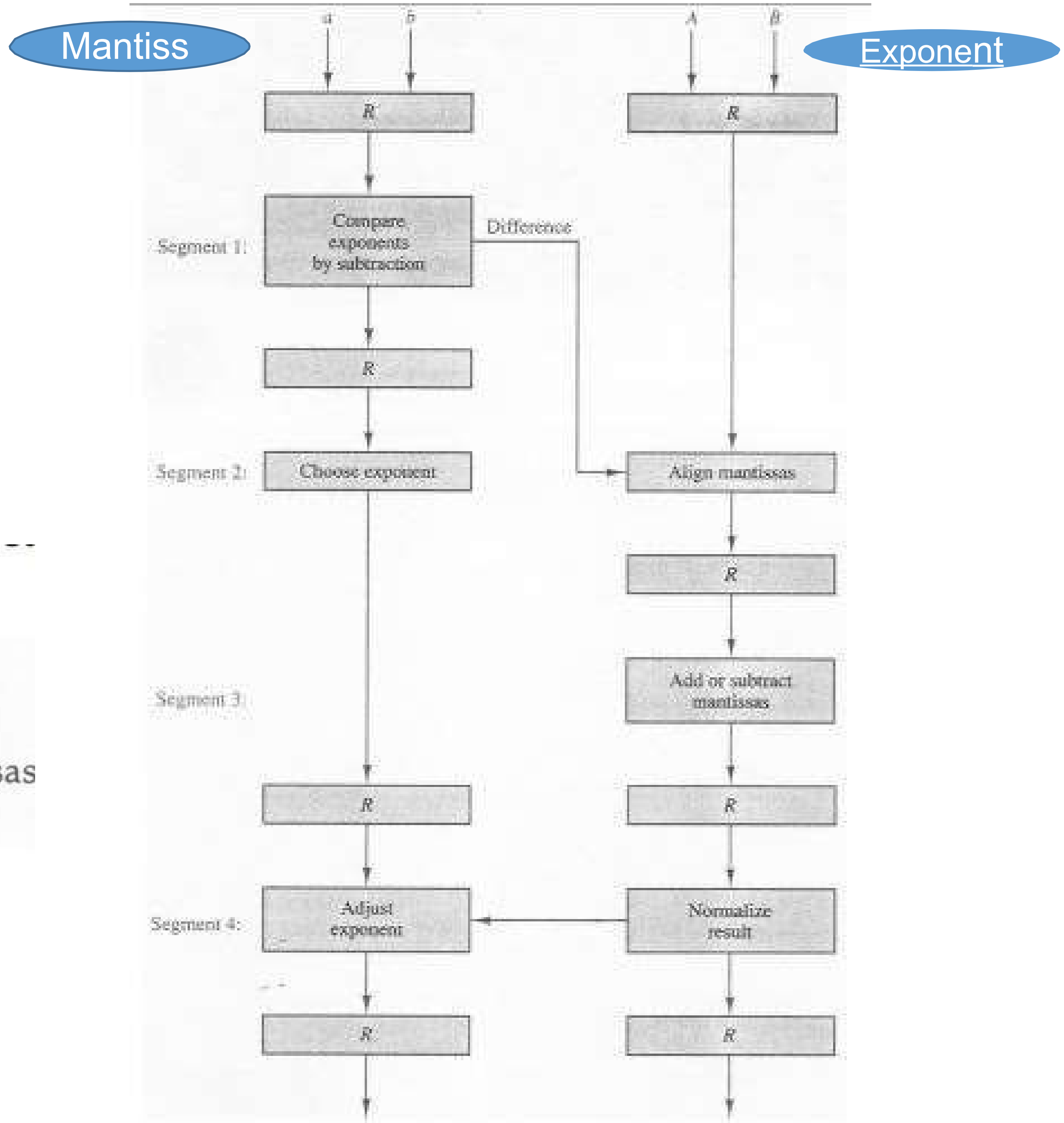
Exponent

- A and B are mantissas and a and b are the exponents.
- The floating point addition and subtraction can be performed in four segments.

Floating-Point Add/Subtraction Pipeline:

4-Segment Pipeline :

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



Floating-point Add/Subtraction Pipeline Example :

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

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

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.

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

Floating-point Add/Subtraction Pipeline Example (Cont.) :

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 nonpipeline floating-point 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.

Vector Processing

- **Science and Engineering Applications**
  - Long-range weather forecasting,
  - Petroleum explorations,
  - Seismic data analysis
  - Medical diagnosis ,
  - Aerodynamics and space flight simulators,
  - Artificial intelligence and expert systems,
  - Mapping the human genome, Image processing

Vector Processing

Vector Operations :

following Fortran DO loop:

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

This is a program for adding two vectors *A* and *B* of length 100 to produce a vector *C*. This is implemented in machine language by the following sequence of operations.

```
      Initialize I = 0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = I + 1
      If I ≤ 100 go to 20
      Continue
```

Vector Instruction Format :

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length

0

Matrix Multiplication

3 x 3 matrices multiplication : $n^2 = 9$ inner product

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

$$c_{11} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$$

: inner product

9

Cumulative multiply-add operation : $n^3 = 27$ multiply-add

$$c = c + a \times b$$

: Three such multiply-add

$$c_{11} = c_{11} + a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$$

therefore 9 X 3 multiply-add = 27

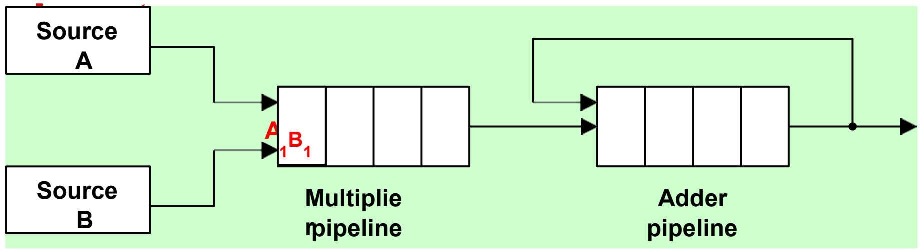
C_{11} initial value = 0

① ① ② ② ③ ③

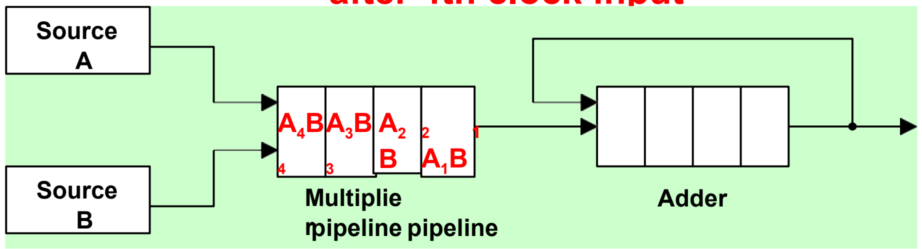
- Pipeline for calculating an inner product :
- Floating point multiplier pipeline : 4 segment
- Floating p[...]ment

Example: $C = A_1B_1 + A_2B_2 + A_3B_3 + \Lambda + A_kB_k$

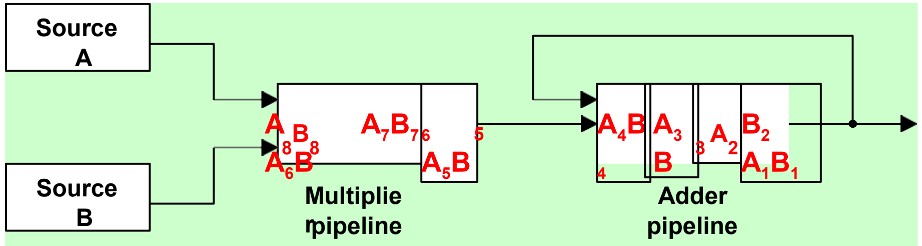
- after 1st clock



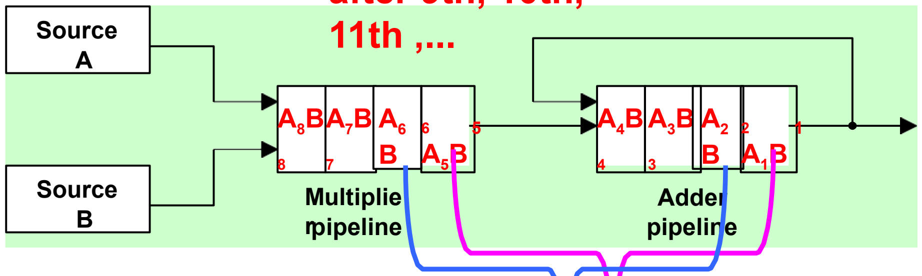
- after 4th clock input



- after 8th clock



- after 9th, 10th, 11th ,....



$A_2B_2 + A_6B_6$ ⑩

$A_1B_1 + A_5B_5$ ⑨

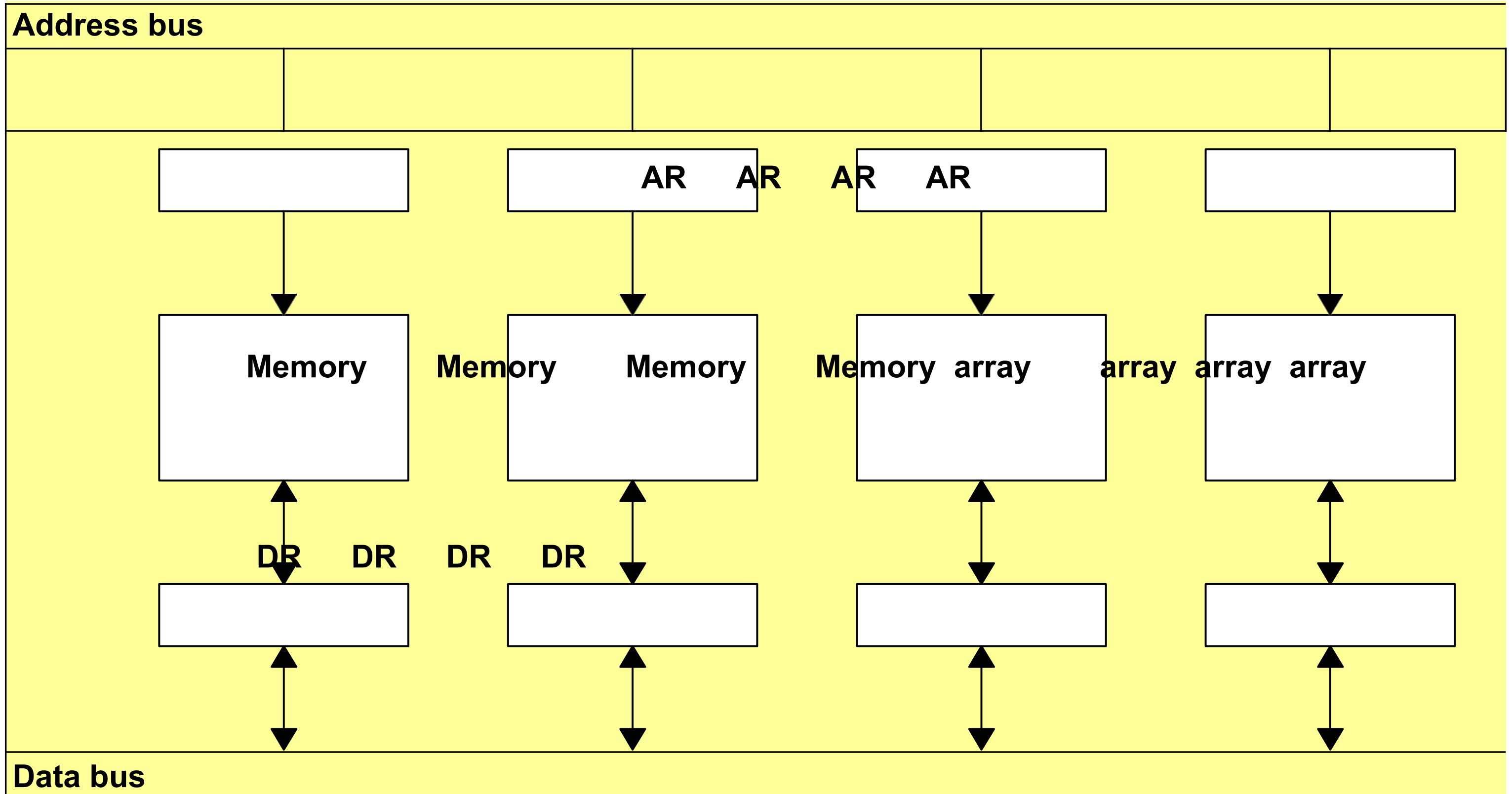
, , ,

- The four partial sum are added to form the final sum

$$\begin{aligned} C = {} & A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \Lambda \\ & + A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \Lambda \\ & + A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \Lambda \\ & + A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \Lambda \end{aligned}$$

Memory Interleaving

- Memory Interleaving :
  - *Simultaneous* access to memory from two or more source using *one memory bus system.*
  - Select one of 4 memory modules using lower 2 bits of AR
  - Example) Even / Odd Address Memory Access



Array Processor

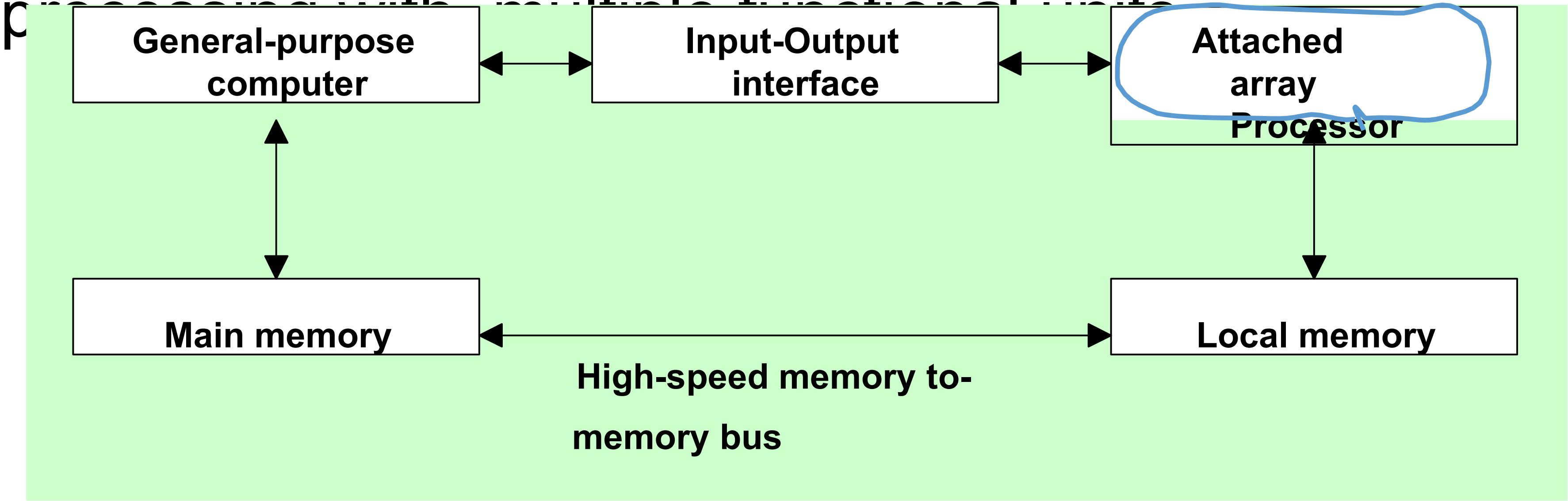
- Processor that performs the computations on large arrays of data.

Vector processing : Adder/Multiplier pipeline use Array processing: using a separate array processor

- There are two different types of (array processor) :
  - Attached Array Processor
  - SIMD Array Processor

Attached Array Processor

- It is designed as a peripheral for a conventional host computer.
- Its purpose is to enhance the performance of the computer by providing vector processing.
- It achieves high performance by means of parallel processing with multiple functional units.



SIMD Array Processor

- It is processor which consists of multiple processing unit operating in parallel.
- The processing units are synchronized to perform the same task under control of common control unit.
- Each processor elements(PE) includes an ALU , a floating