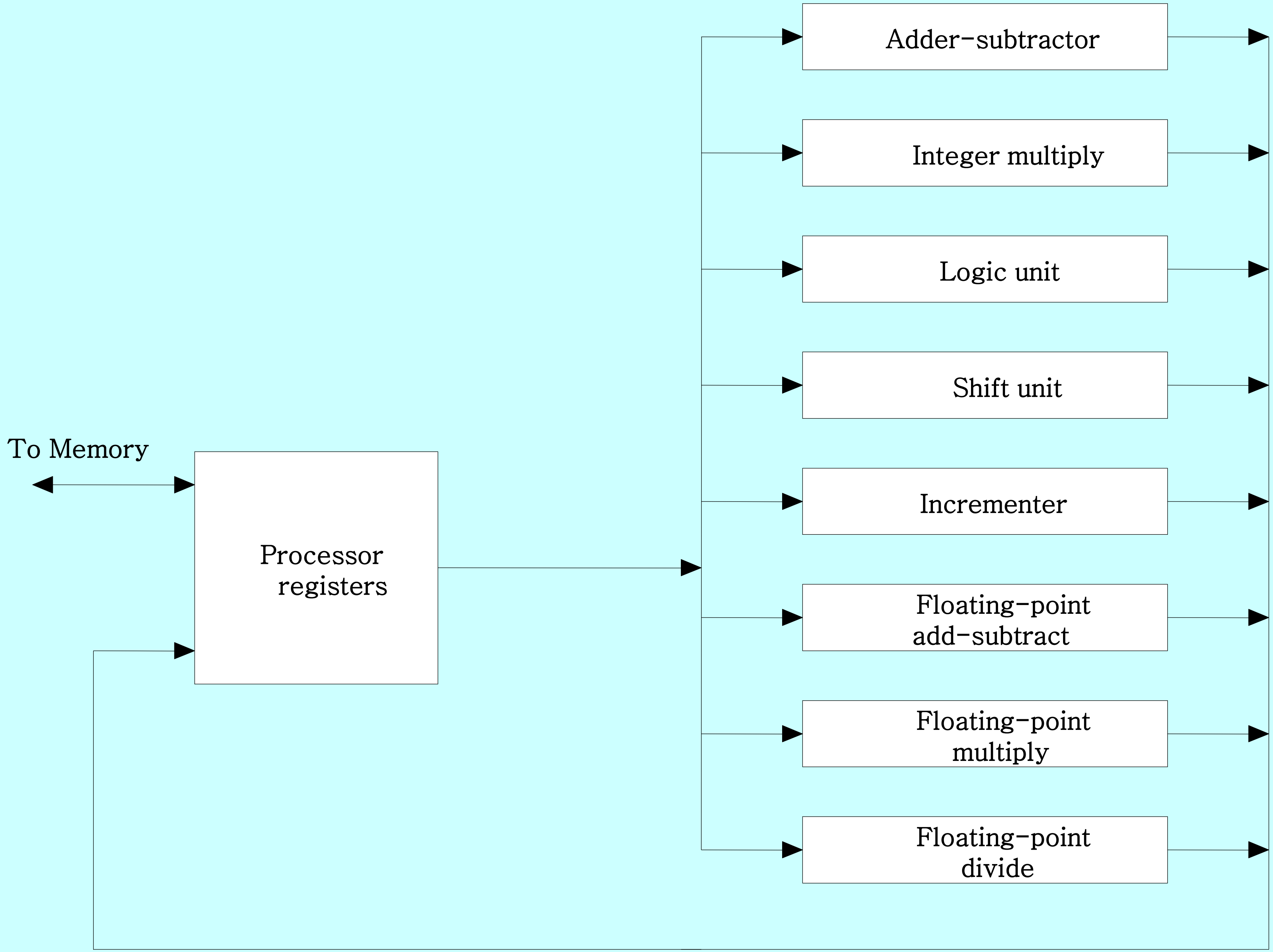
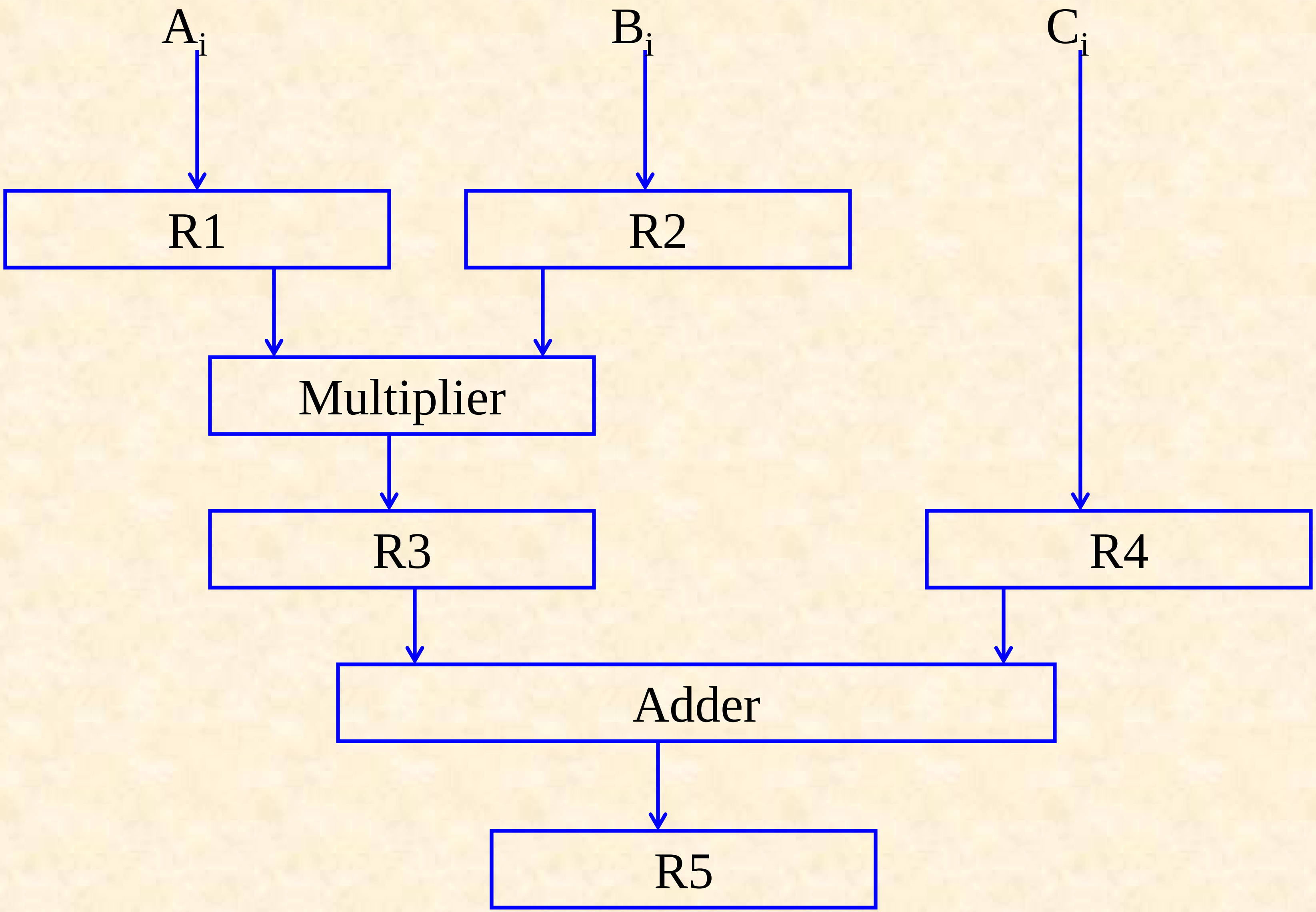
Pipeline and Vector Processing

- 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***


To Memory
Processor registers
Adder-subtractor
Integer multiply
Logic unit
Shift unit
Incrementer
Floating-point add-subtract
Floating-point multiply
Floating-point divide

- **Pipelining :** it is the process of Decomposing a sequential process into suboperations with Each subprocess is executed in a special dedicated segment concurrently with all other segments.
- It is a collection of processing segments through which binary information flows. Where each segment performs partial processing dedicated by the way the task is partioned.
  - Pipelining의 예제 : ***Fig. 9-2***
    - Multiply and add operation : $Ai * Bi + Ci$ (for i = 1, 2, …, 7)
    - 3 개의 Suboperation Segment로 분리
      - 1) $R1 \leftarrow Ai,\ R2 \leftarrow Bi$: Input Ai and Bi
      - 2) $R3 \leftarrow R1 * R2,\ R4 \leftarrow Ci$: Multiply and input Ci
      - 3) $R5 \leftarrow R3 + R4$: Add Ci
    - Content of registers in pipeline example : ***Tab. 9-1***


A_i
B_i
C_i
R1
R2
Multiplier
R3
R4
Adder
R5

Clock pulse Number	Segment1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	-	-	-
2	A2	B2	A1*B1	C1	-
3	A3	B3	A2*B2	C2	A1*B1+C1
4	A4	B4	A3*B3	C3	A2*B2+C2
5	A5	B5	A4*B4	C4	A3*B3+C3
6	A6	A6	A5*B5	C5	A4*B4+C4
7	A7	A7	A6*B6	C6	A5*B5+C5
8	-	-	A7*B7	C7	A6*B6+C6
9	-	-	-	-	A7*B7+C7

◆General considerations

- 4 segment pipeline : the operand pass through all four segments in a fixed sequence. Each segment consists of a combinational ckt Si that performs a sub operation over the data stream. The segments are separated by the registers to hold the intermediate results.

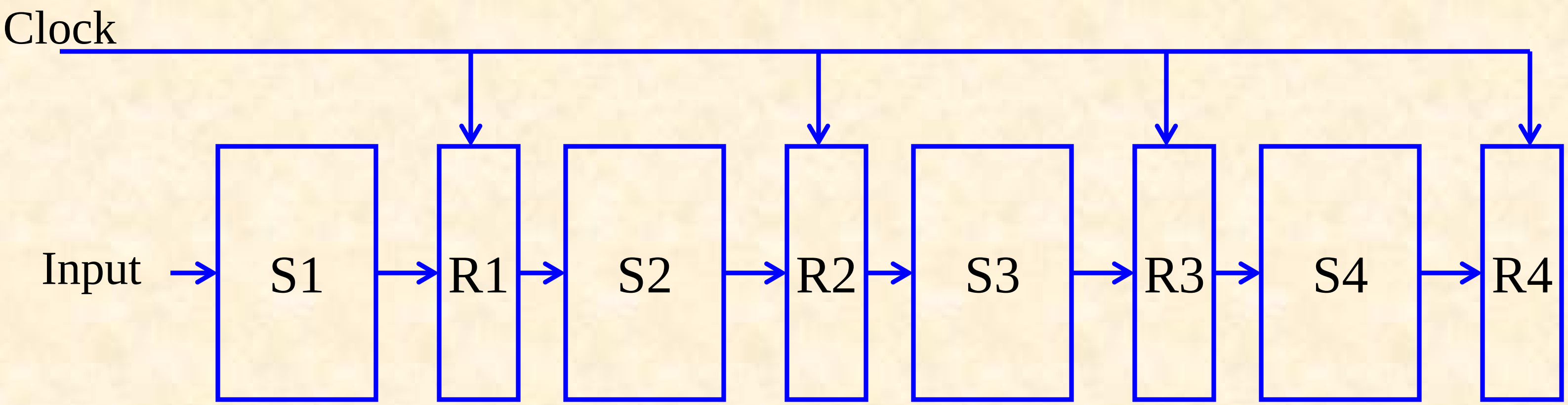


Fig.: Four Segment pipeline

- Space-time diagram :
  - »Show segment utilization as a function of time
- Task : T1, T2, T3,…, T6 executed in four segments.
  - »Total operation performed going through all the segment

Pipeline에서의 처리 시간 = 9 clock cycles

Segment \ Clock cycles	1	2	3	4	5	6	7	8	9
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

- Speedup S : Nonpipeline / Pipeline
  - $\mathbf{S = n \cdot t_n / (k + n - 1) \cdot t_p} = 6 \cdot 6\, t_n / (4 + 6 - 1) \cdot t_p = 36\, t_n / 9\, t_n = 4$
    - n : *task number* (6)
    - t_n : *time to complete each task in nonpipeline* (6 cycle times = 6 t_p)
    - t_p : *clock cycle time* (1 clock cycle)
    - k : *segment number* (4)

k + n - 1 ≈ n

  - If $n \rightarrow \infty$ 이면, $\mathbf{S = t_n / t_p}$
  - If we assume that the time it takes to process a task is the same in the pipeline and nonpipeline circuits then we have

nonpipeline ($\mathbf{t_n}$) **=** ***pipeline*** ($\mathbf{k \cdot t_p}$)

$\mathbf{S} = \mathbf{t_n / t_p} = \mathbf{k \cdot t_p / t_p} = \mathbf{k}$

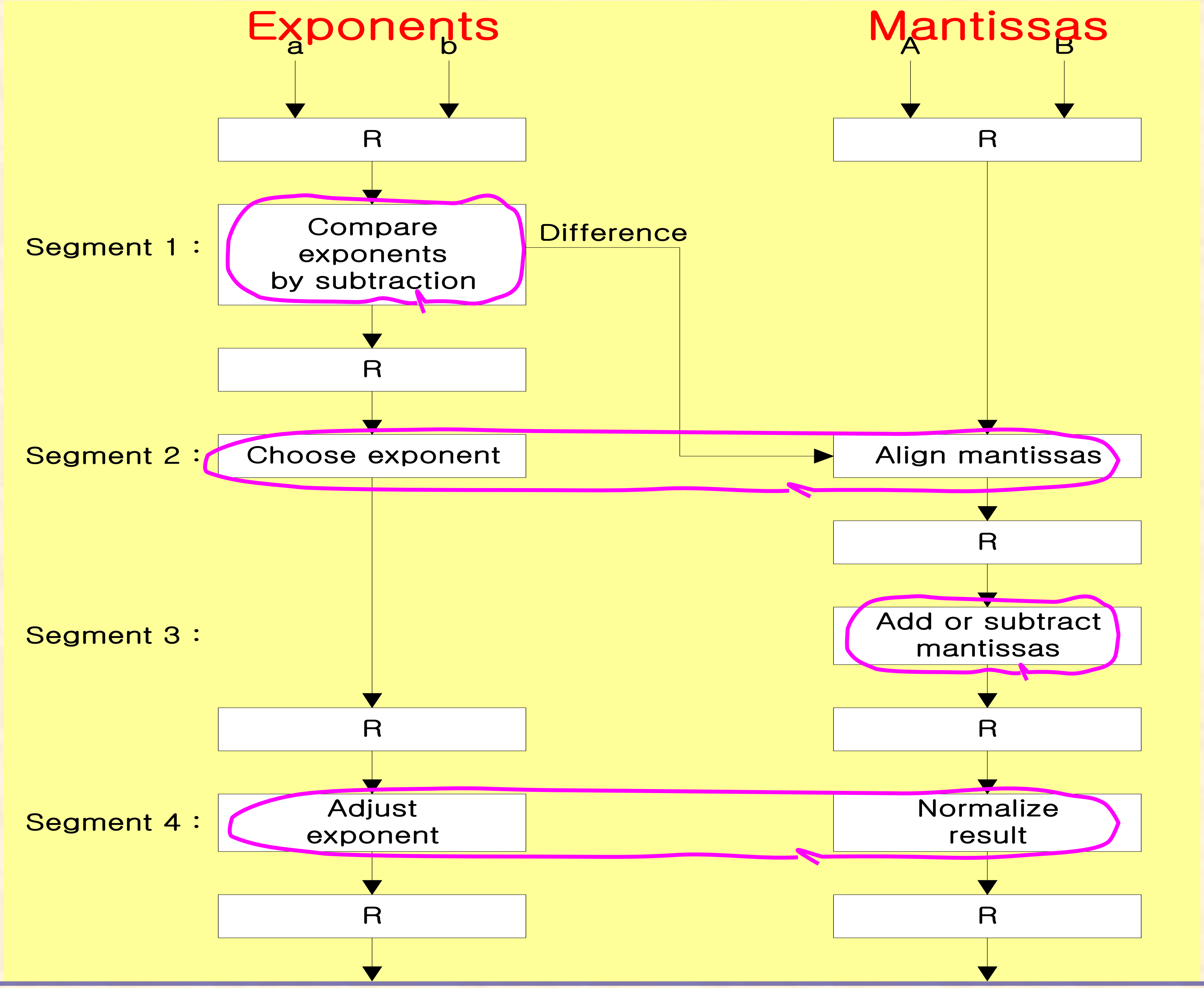
Where k is the number of segments.

Arithmetic Pipeline

- Floating-point Adder Pipeline Example :
  - Add / Subtract two normalized floating-point binary number
    - $X = A \times 2^a = 0.9504 \times 10^3$
    - $Y = B \times 2^b = 0.8200 \times 10^2$

- 4 segments suboperations
  - 1) Compare exponents by subtraction :

      3 - 2 = 1
    - X = 0.9504 x 10^3
    - Y = 0.8200 x 10^2
  - 2) Align mantissas
    - X = 0.9504 x 10^3
    - Y = 0.08200 x 10^3
  - 3) Add mantissas
    - Z = 1.0324 x 10^3
  - 4) Normalize result
    - Z = 0.1324 x 10^4


Exponents
Mantissas
a
b
A
B
R
R
Segment 1 :
Compare exponents by subtraction
Difference
R
Segment 2 :
Choose exponent
Align mantissas
R
Segment 3 :
Add or subtract mantissas
R
R
Segment 4 :
Adjust exponent
Normalize result
R
R

Instruction Pipeline

- Instruction Cycle

  1) Fetch the instruction from memory

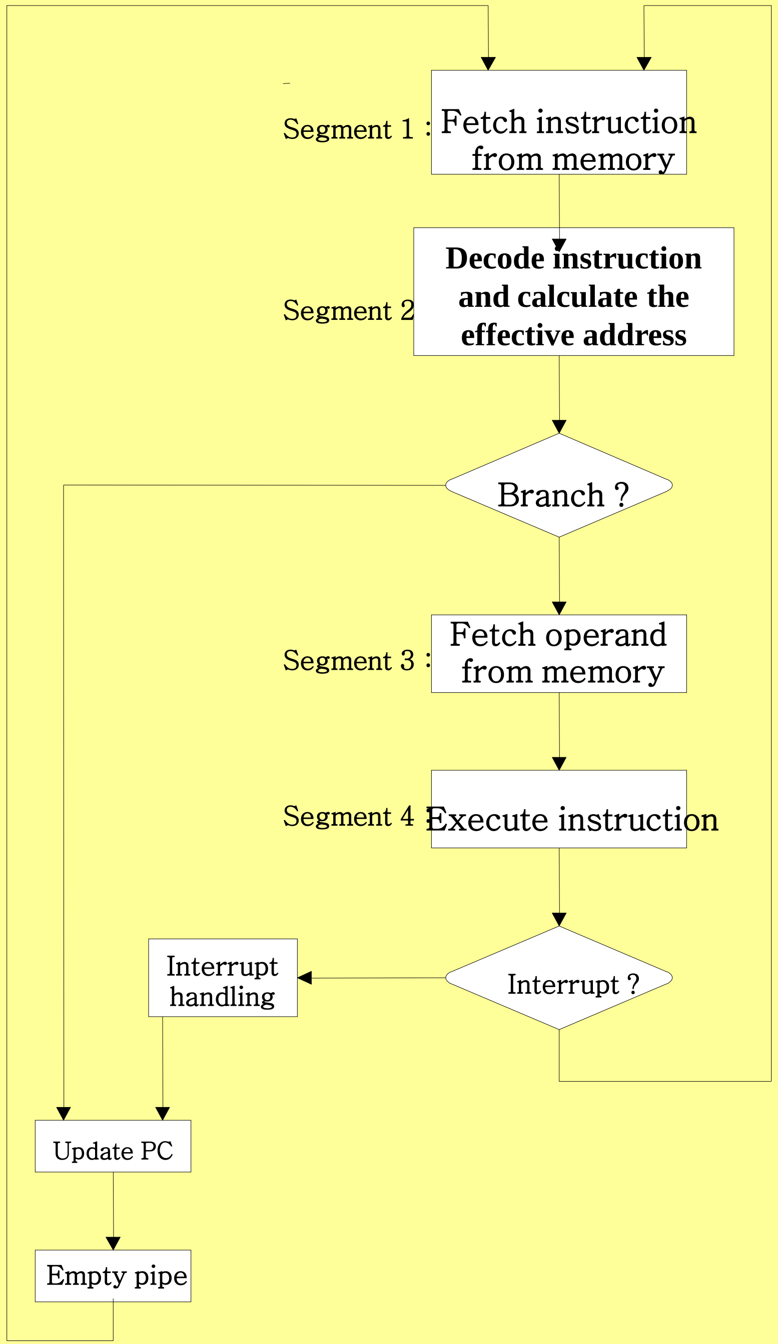
  2) Decode the instruction

  3) Calculate the effective address

  4) Fetch the operands from memory

  5) Execute the instruction

  6) Store the result in the proper place


Segment 1 :
Fetch instruction from memory
Segment 2
Decode instruction and calculate the effective address
Branch ?
Segment 3 :
Fetch operand from memory
Segment 4 :
Execute instruction
Interrupt ?
Interrupt handling
Update PC
Empty pipe

Example : Four-segment Instruction Pipeline

- Four-segment CPU pipeline :
  - » 1) **FI** : Instruction Fetch
  - » 2) **DA** : Decode Instruction & calculate EA
  - » 3) **FO** : Operand Fetch
  - » 4) **EX** : Execution
- Timing of Instruction Pipeline :

Step :		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction :	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	—	—	FI	DA	FO	EX			
	5					—	—	—	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

No Branch

Branch

- Pipeline Conflicts : 3 major difficulties
  - 1) Resource conflicts
    - memory access by two segments at the same time.
    - Can be avoided by using separate instruction stream and data memories.
  - 2) Data dependency
    - when an instruction depend on the result of a previous instruction, but this result is not yet available
  - 3) Branch difficulties
    - branch and other instruction (interrupt, ret, ..) that change the value of PC
- Data Dependency 해결 방법
  - Hardware 적인 방법
    - Hardware Interlock
      - previous instruction의 결과가 나올 때 까지 Hardware 적인 Delay를 강제 삽입
    - Operand Forwarding
      - previous instruction의 결과를 곧바로 ALU 로 전달 (정상적인 경우, register를 경유함)
  - Software 적인 방법
    - Delayed Load
      - previous instruction의 결과가 나올 때 까지 No-operation instruction 을 삽입

Assignment

- What do you mean by pipeline and parallel processing.
- Explain vector processing.