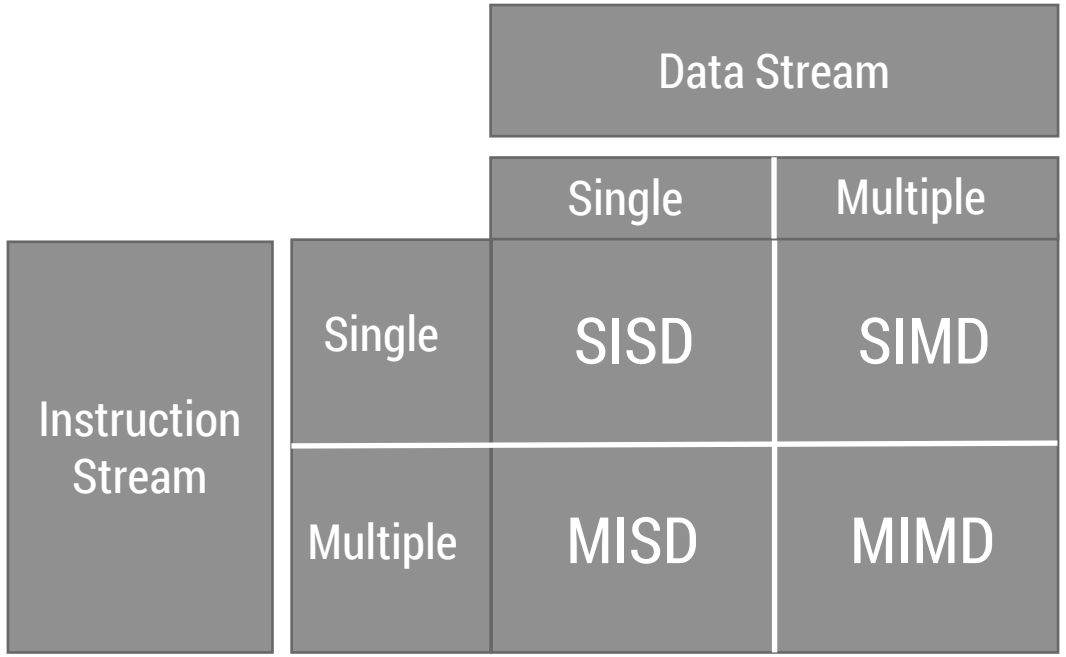
Pipeline & Vector Processing

Flynn's taxonomy



Single Instruction Single Data & Single Instruction Multiple Data

Single Instruction Single Data (SISD)

- SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.
- Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.

Single Instruction Multiple Data (SIMD)

- SIMD represents an organization that includes many processing units under the supervision of a common control unit.
- All processors receive the same instruction from the control unit but operate on different items of data.

Multiple Instruction Single Data & Multiple Instruction Multiple Data

Multiple Instruction Single Data (MISD)

- There is no computer at present that can be classified as MISD.
- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

Multiple Instruction Multiple Data (MIMD)

- MIMD organization refers to a computer system capable of processing several programs at the same time.
- Most multiprocessor and multicomputer systems can be classified in this category.
- Contains multiple processing units.
- Execution of multiple instructions on multiple data.

Parallel Processing

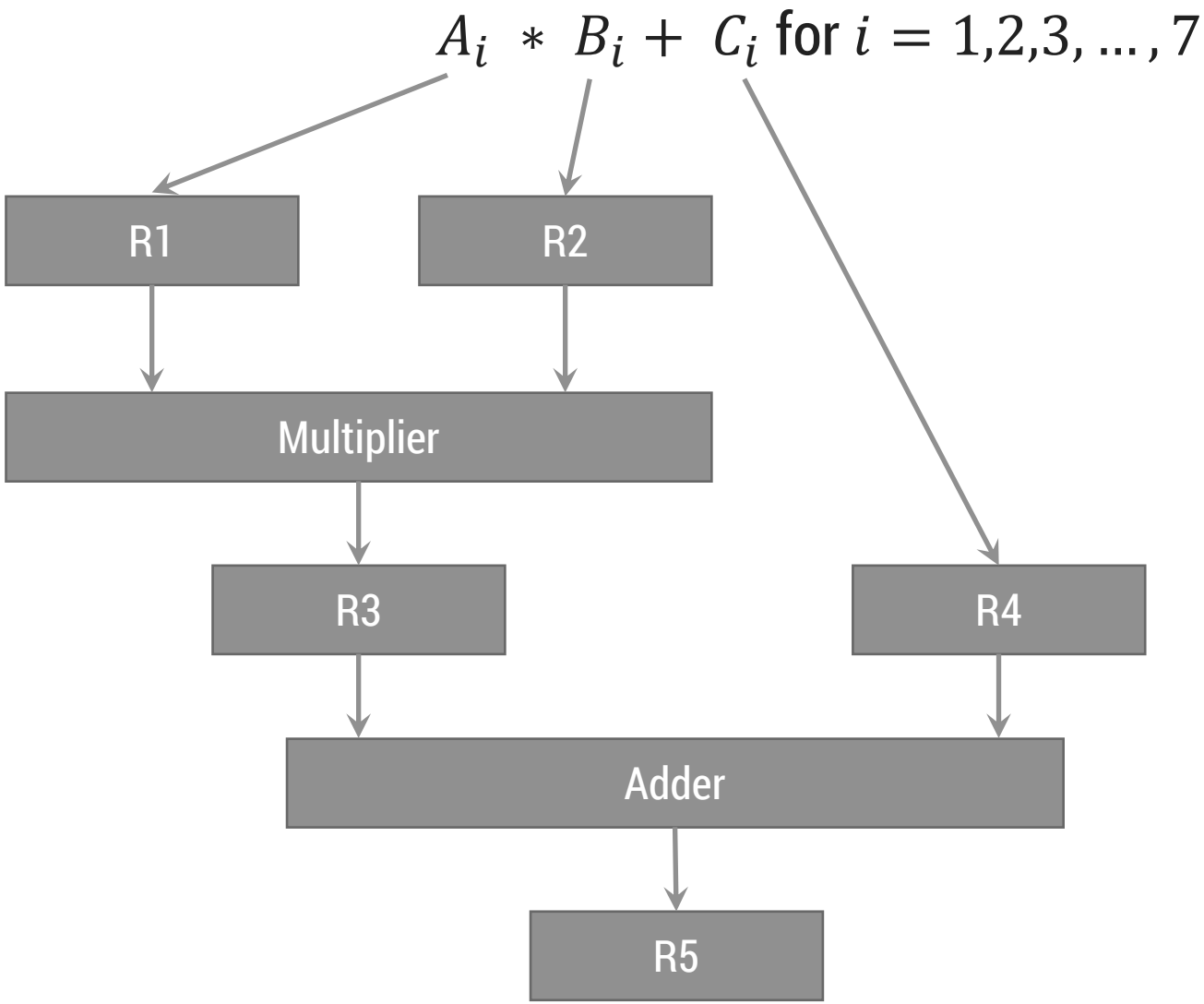
- Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.
- Purpose of parallel processing is to speed up the computer processing capability and increase its throughput.
- Throughput:

  The amount of processing that can be accomplished during a given interval of time.

Pipelining

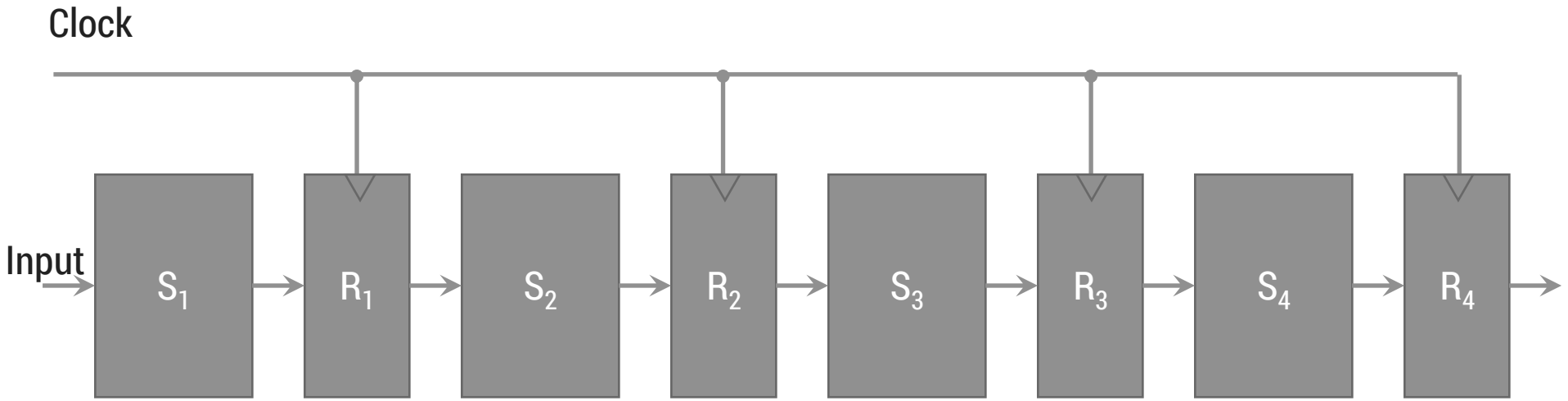
- Pipeline is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.
- A pipeline can be visualized as a collection of processing segments through which binary information 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 registers provide isolation between each segment.
- The technique is efficient for those applications that need to repeat the same task many times with different sets of data.

Pipelining example



Pipelining

- General structure of four segment pipeline



Space-time Diagram

Segment	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	T_1				T_2				T_3				T_4			
2		T_1				T_2				T_3				T_4		
3			T_1				T_2				T_3				T_4	
4				T_1				T_2				T_3				T_4

Non Pipelined Architecture

Segment	1	2	3	4	5	6	7
1	T_1	T_2	T_3	T_4			
2		T_1	T_2	T_3	T_4		
3			T_1	T_2	T_3	T_4	
4				T_1	T_2	T_3	T_4

→ Clock cycles

Pipelined Architecture

Speedup

- Speedup of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio

$$S = \frac{nt_n}{(k + n - 1)t_p}$$

- If number of tasks in pipeline increases w.r.t. number of segments then n becomes larger than $k - 1$, under this condition speedup becomes

$$S = \frac{nt_n}{nt_p} = \frac{t_n}{t_p}$$

- Assuming time to process a task in pipeline and non-pipeline circuit is same then

$$S = \frac{kt_p}{t_p} = k$$

- Theoretically maximum speedup achieved is the number of segments in the pipeline.

Arithmetic Pipeline

- Usually found in high speed computers.
- Used to implement floating point operations, multiplication of fixed point numbers and similar operations.
- Example:
- Consider an example of floating point addition and subtraction.

$$X = A \times 10^{a}$$
$$Y = B \times 10^{b}$$

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

Example of Arithmetic Pipeline

- Consider the two normalized floating-point numbers:

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

- Segment-1: The larger exponent is chosen as the exponent of result.
- Segment-2: Aligning the mantissa numbers

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

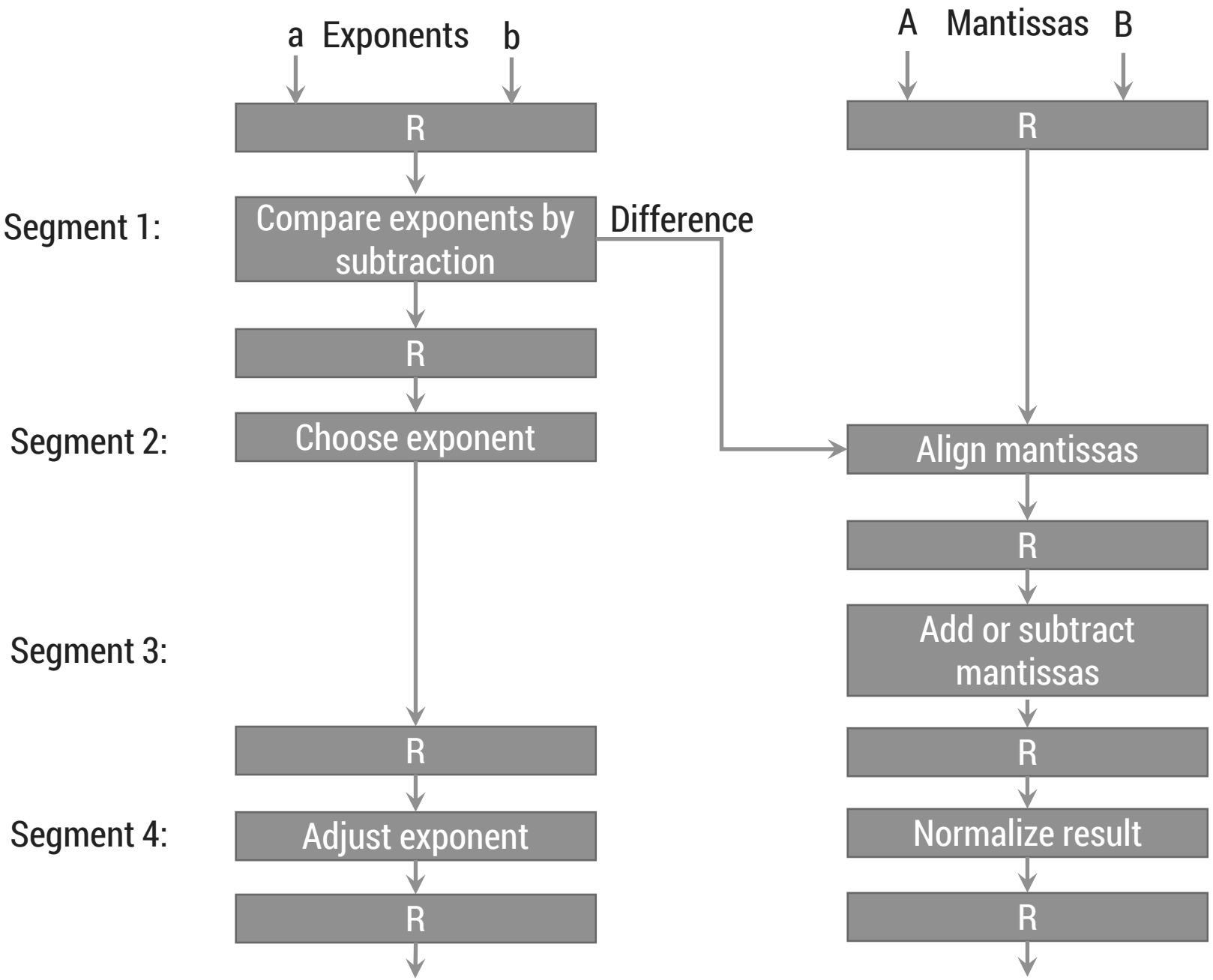
- Segment-3: Addition of the two mantissas produces the sum

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

- Segment-4: Normalize the result

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

- The sub-operations 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


a
Exponents
b
A
Mantissas
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

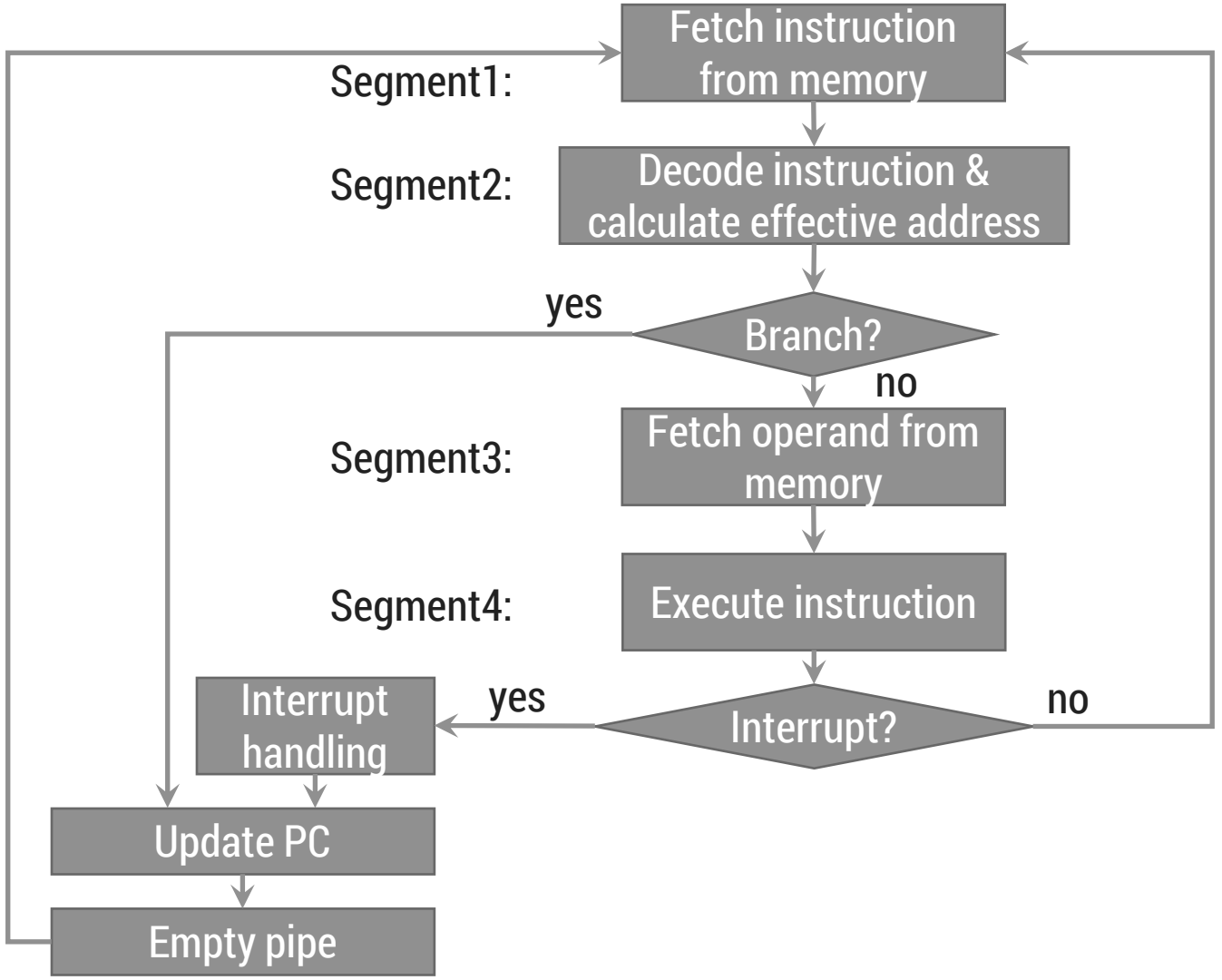
Instruction Pipeline

- In the most general case, the computer needs to process each instruction with the following sequence of steps
  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.
- Different segments may take different times to operate on the incoming information.
- Some segments are skipped for certain operations.
- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration.

Instruction Pipeline

- Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment.
- Assume further that most of the instructions place the result into a processor registers so that the instruction execution and storing of the result can be combined into one segment.
- This reduces the instruction pipeline into four segments.
  1. FI: Fetch an instruction from memory
  2. DA: Decode the instruction and calculate the effective address of the operand
  3. FO: Fetch the operand
  4. EX: Execute the operation

Four segment CPU pipeline



Space-time Diagram

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								
3			FI	DA	FO	EX							
4				FI	DA	FO	EX						
5					FI	DA	FO	EX					
6						FI	DA	FO	EX				
7							FI	DA	FO	EX			

Space-time Diagram

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

Pipeline Conflict & Data Dependency

Pipeline Conflict

- There are three major difficulties that cause the instruction pipeline conflicts.
  1. Resource conflicts caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
  2. Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
  3. Branch difficulties arise from branch and other instructions that change the value of PC.

Data Dependency

- Data dependency occurs when an instruction needs data that are not yet available.
- Pipelined computers deal with such conflicts between data dependencies in a variety of ways as follows:
  1. Hardware Interlocks
  2. Operand forwarding
  3. Delayed load

Handling Branch Instructions

- The branch instruction breaks the normal sequence of the instruction stream, causing difficulties in the operation of the instruction pipeline.
- Hardware techniques available to minimize the performance degradation caused by instruction branching are as follows:
  - Pre-fetch target
  - Branch target buffer
  - Loop buffer
  - Branch prediction
  - Delayed branch

RISC Pipeline (Three segment instruction pipeline)

- Because of the fixed-length instruction format, the decoding of the operation can occur at the same time as the register selection.
- Since all the operands are in registers, there is no need for calculating an effective address or fetching of operands from memory.
- RISC has main two facilities:
  - Single-cycle instruction execution
  - Compiler support
- Mainly three types of instructions in RISC:
  1. Data manipulation instructions
  2. Data transfer instructions
  3. Program control instructions
- The instruction cycle can be divided into three sub-operations and implemented in three segments:
  - I : Instruction fetch
  - A : ALU operation
  - E : Execute instruction

Delayed Load

Pipeline timing with data conflict

Clock cycles:	1	2	3	4	5	6
Load R1	I	A	E			
Load R2		I	A	E		
Add R1+R2			I	A	E	
Store R3				I	A	E

Clock cycles:	1	2	3	4	5	6	7
Load R1	I	A	E				
Load R2		I	A	E			
No-operation			I	A	E		
Add R1+R2				I	A	E	
Store R3					I	A	E

Pipeline timing with delayed load

Delayed Branch

Clock cycle:	1	2	3	4	5	6	7	8	9	10
Load R1	I	A	E							
Increment R2		I	A	E						
Add R3+R4			I	A	E					
Subtract R6-R5				I	A	E				
Branch to X					I	A	E			
No operation						I	A	E		
No operation							I	A	E	
Instruction in X								I	A	E

Using no-operation instructions

Delayed Branch

Clock cycle:	1	2	3	4	5	6	7	8
Load R1	I	A	E					
Increment R2		I	A	E				
Branch to X			I	A	E			
Add R3+R4				I	A	E		
Subtract R6-R5					I	A	E	
Instruction in X						I	A	E

Rearranging the instructions