Enhanced Binary Floating Point Interval Adder with Decorations

By:
Abdelrahman Elskahwy
Kareem Ismail
Maha Zohdy

Under the supervision of:
Associate Prof. Hossam A. H. Fahmy
IEEE P1788 committee member

Contact : elskhawy.a@gmail.com

Presented by:
Amin Maher

AGENDA

- Motivation
- Adder Design
  - Parallel Adder Design
  - Decorations
  - Interval Adder with Decorations
- Testing
  - Test Vectors Generation
- Results & Comparisons
- Conclusion
- What’s next ?

MOTIVATION

1. The need for an enhanced adder for high speed applications.

2. Unlike conventional operations on discrete floating point numbers, operations on floating point intervals are flagged with Decorations.

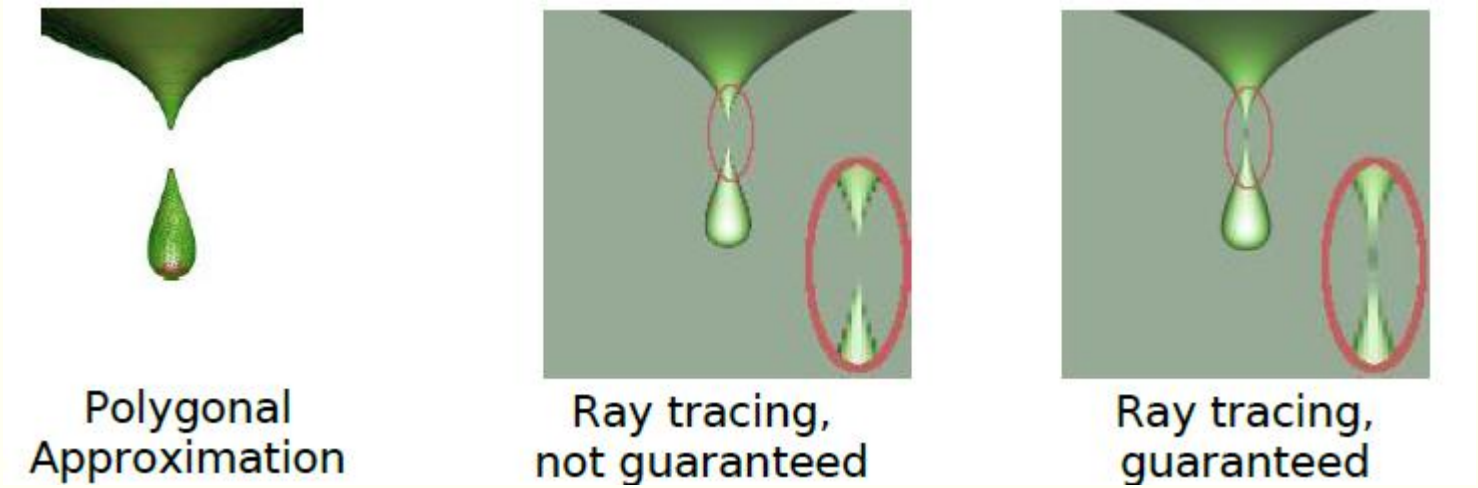
Achievements:

1. Floating point interval adder with enhanced speed and area over the 1st and only implementation of binary floating point interval adder.

2. 1st Implementation of Decorations in Interval Adder

INTERVAL APPLICATIONS

- Solves error due to rounding
  - Result obtained may be totally wrong
- Saves simulation time
  - Component's tolerance
- Computer graphics



PARALLEL ADDER DESIGN

- We adopted the tow-path design, as we were targeting the speed.
- Some floating point addition/subtraction characteristics

Full length alignment & normalization shift.

Rounding & conversion addition.

SPECIAL CASES

- **Infinities according to IEEE 754 standard.**

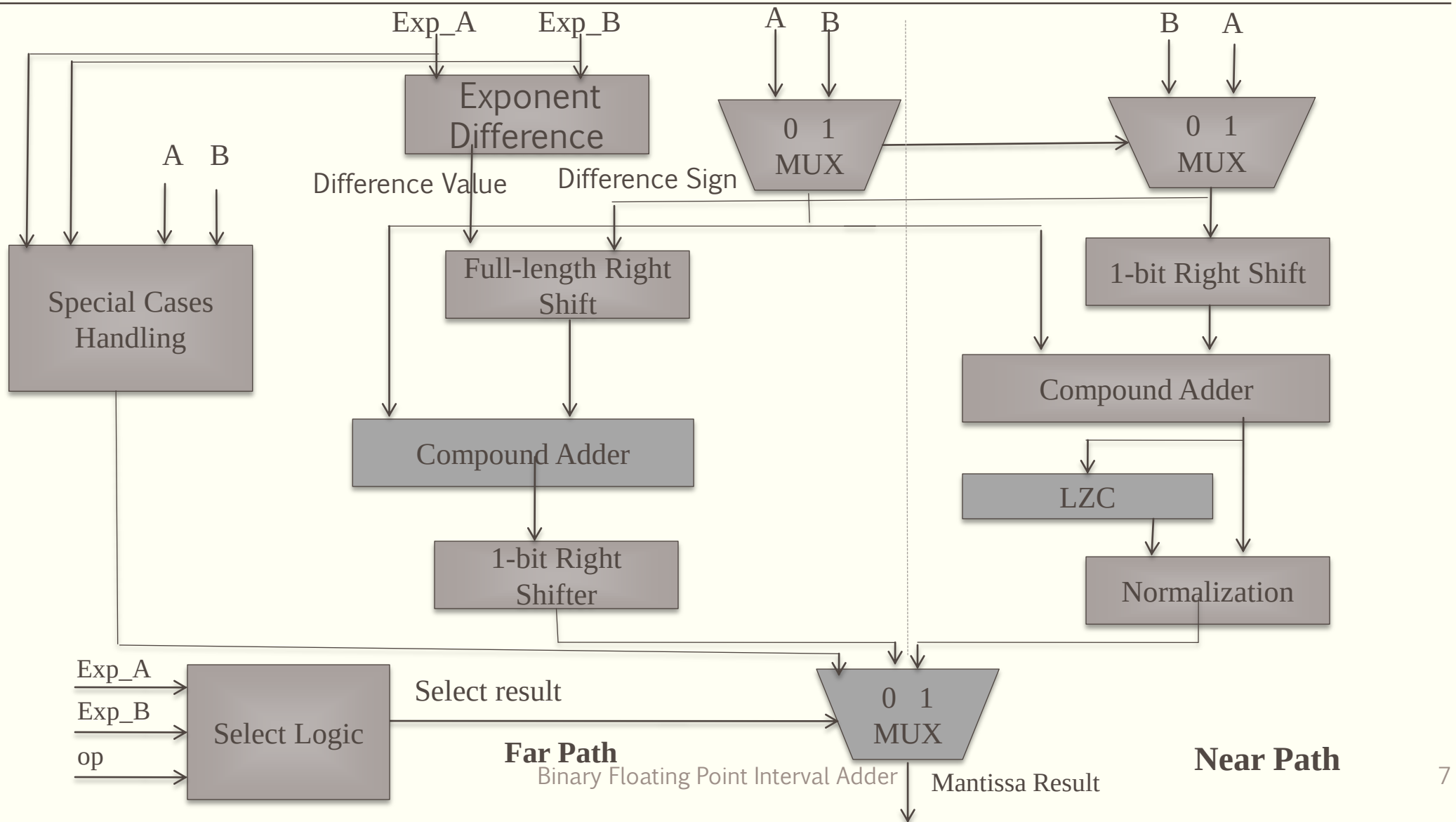
Double-Format Bit Pattern	Value
s = 0; e = 2047; sig = 0 (all bits in f are zero)	+INF (positive infinity)
s = 1; e = 2047; sig = 0 (all bits in sig are zero)	-INF (negative infinity)

- **Nan (Not A Number):**

If any of the two operands is Nan, the result will be a canonical Nan. (Nan_sig==>13'h4000000000000)

Single-Format Bit Pattern	Value
s = x; e =2047; sig≠0 (at least one bit in sig is nonzero)	Nan (Not-a-Number)

PARALLEL DESIGN BLOCK DIAGRAM



DECORATIONS

- Decorations is used to describe a ***property*** , not of the interval it is attached to but of the function evaluated on the i/p intervals.
- Example : If a code defines the expression *f* (x,y)= $\sqrt{y^2 - x * y}$, then decorated-interval gives information about **definedness**, **continuity**, etc. of the function *f* (x,y) over the intervals (x,y).
- IEEE P1788's Decorations replace the status flags (Invalid, Inexact, Overflow, and underflow) of IEEE754 standard.
- Decoration of the o/p interval should not depend on i/p intervals' decorations, but on the i/p intervals.

DECORATIONS

Value	Short description	Property	Definition
• com	• Common	• P_{com} (f, x)	• x is a bounded, nonempty subset of Dom(f); f is continuous at each point of x; and the computed interval f(x) is bounded.
• dac	• Defined & continuous	• P_{dac} (f, x)	•x is a nonempty subset of Dom(f), and the restriction of f to x is continuous.
• def	• Defined	• P_{def} (f, x)	• x is a nonempty subset of Dom(f).
• trv	• Trivial	• P_{trv} (f, x)	• Always true (so gives no information).
• ill	• Ill-formed	• P_{ill} (f, x)	• Not an Interval; formally Dom(f) =φ .

DECORATIONS

- **Example:**

For the function $f(x)$= **sqrt(x)** applied on interval x as follows:

- [0,1], then f is decorated **Dac**
- [-1,1], then f is decorated **Trv**
- $(-1-x^2)$, then f is decorated **Ill-formed**

Implementation:
Decorations are implemented as **3-bits** attached to the interval.

Decoration Bits (Logical values)	Decoration value
000	Com
001	Dac
010	Def
011	Trv
100	Ill

DECORATIONS

- Finally for any function φ that is continuous everywhere

  $\varphi(x_1, x_2, \ldots, x_k)$, and for all inputs non empty, is Decorated :

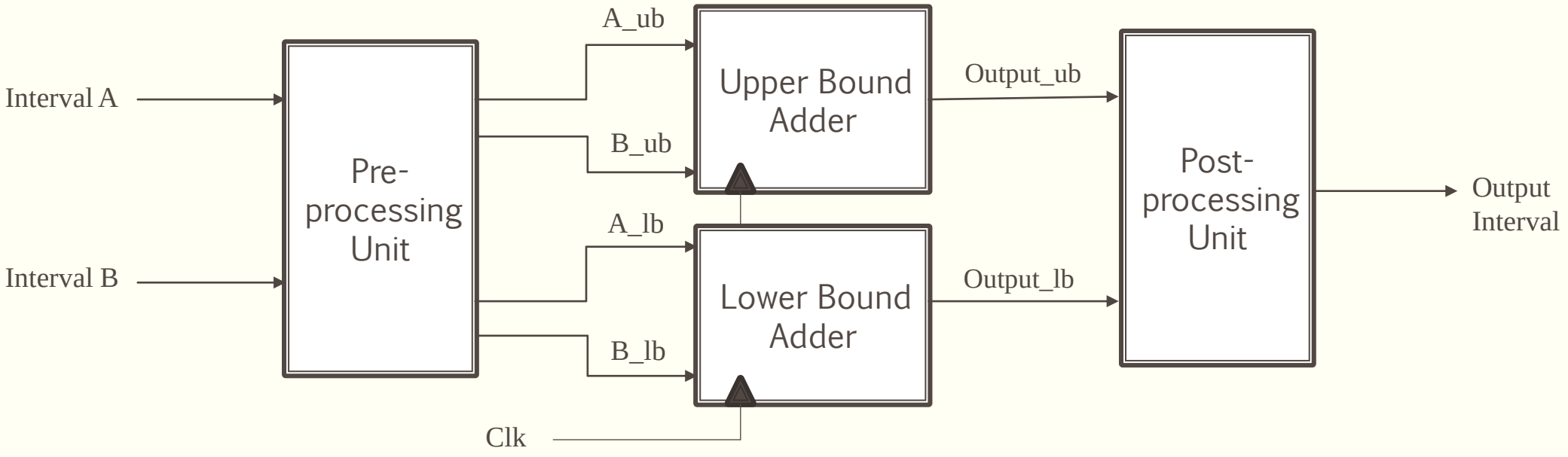
  - com ,if inputs bounded and result bounded
  - dac, otherwise.

- Addition and subtraction is defined and continuous on all R , given normal input intervals, thus they always give **Com** or **D**ac decorated interval.

INTERVAL ADDER WITH DECORATIONS

Two proposed designs:

1. Parallel design using Two Path Algorithm Adder.

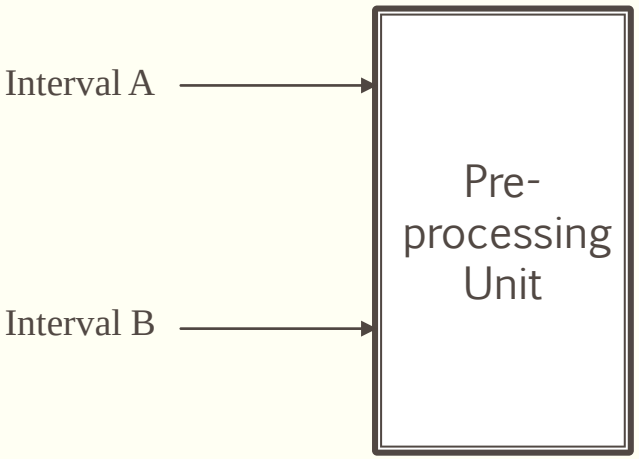


Parallel operation -> Fast speed, higher area.

INTERVAL ADDER WITH DECORATIONS

Two proposed designs:

1. Parallel design using Two Path Algorithm Adder.

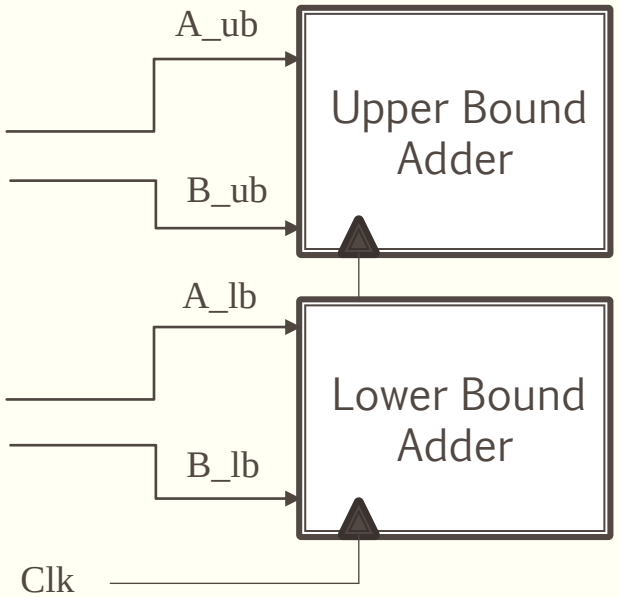


Divides the interval operands into two parallel floating point addition/subtraction operations with the appropriate rounding mode for each operation, and extract the **Decorations** of each interval.

INTERVAL ADDER WITH DECORATIONS

Two proposed designs:

1. Parallel design using Two Path Algorithm Adder.



Double precision floating point adder built from scratch, to enhance it's speed, using **Verilog** according to the aforementioned design.

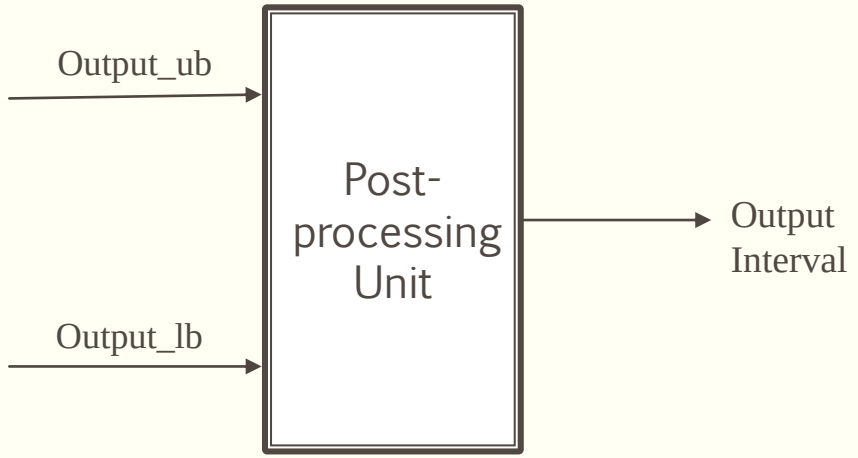
One for the upper bound and the other for the lower bound.

INTERVAL ADDER WITH DECORATIONS

Two proposed designs:

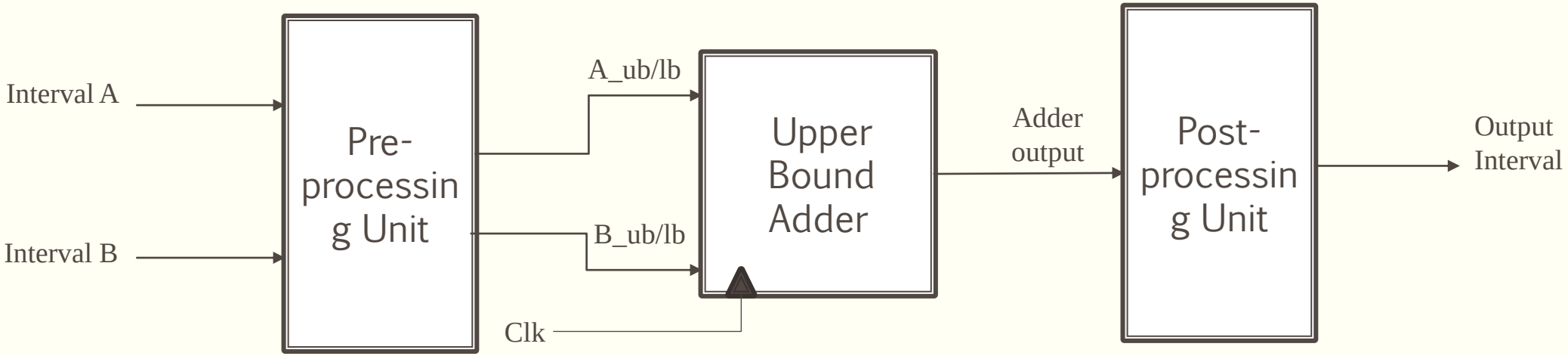
1. Parallel design using Two Path Algorithm Adder.

Collects the two floating point results into one interval result, attaches the calculated Decoration value to the interval, then raises a flag for ready result



INTERVAL ADDER WITH DECORATIONS

2. Serial design using Single Path Algorithm Adder.



- The units have the same functions, but they operate serially on the upper and lower bounds.
- Serial operation -> lower speed, and small area.

GENERATING TEST VECTORS

A manually created algorithm to generate Decimal Floating Point numbers, save these numbers in memory in the IEE754 format, read these numbers byte by byte in its binary form.

Soft Float Library

Maltab & IEEE745 Function

C++ Algorithm

ADVANTAGES

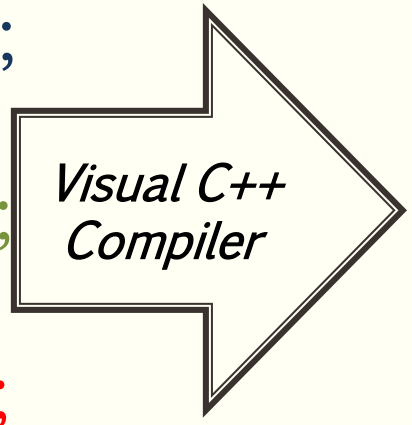
- Double data type is stored in memory according to IEEE754's Format.
- Accurate results as the ADD/SUB operation deals with the internal FPU of the machine's microprocessor.
- The ability to control the rounding mode, and exceptions handling.
- The ability to read the exceptions flags from the internal FPU.
- The ability to change the value and the range of input numbers.

VECTORS GENERATION USING C++ CODE

Double X;
+/-
Double Y;
=
Double Z;

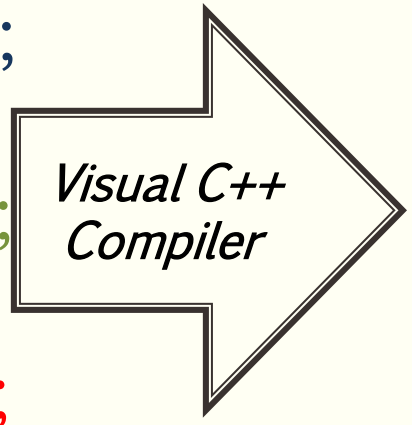
VECTORS GENERATION USING C++ CODE

Double X;
+/-
Double Y;
=
Double Z;



VECTORS GENERATION USING C++ CODE

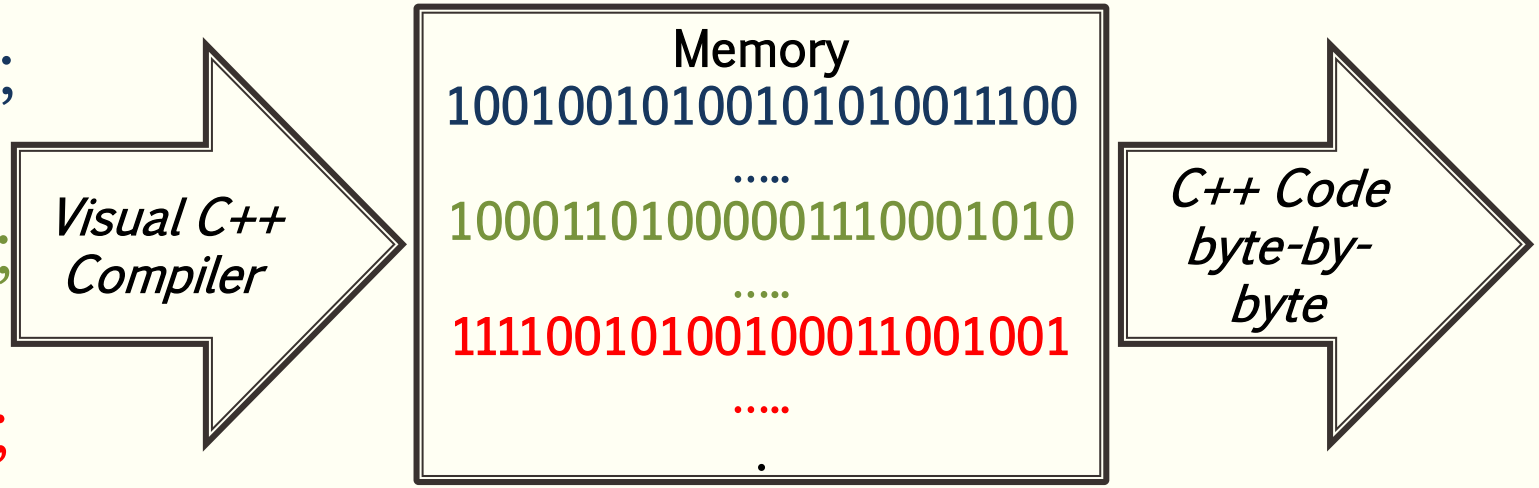
Double X;
+/-
Double Y;
=
Double Z;



Memory
10010010100101010011100
.....
10001101000001110001010
.....
11110010100100011001001
.....
.

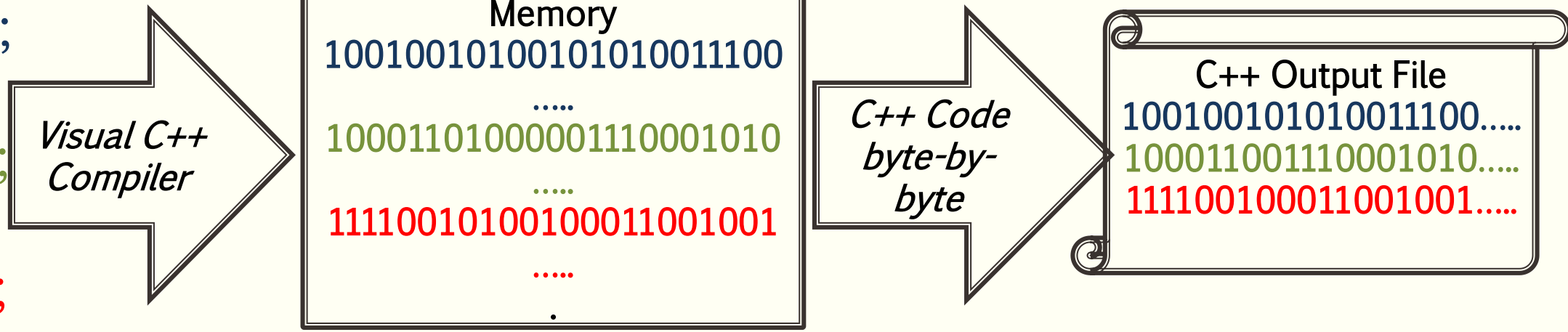
VECTORS GENERATION USING C++ CODE

Double X;
+/-
Double Y;
=
Double Z;



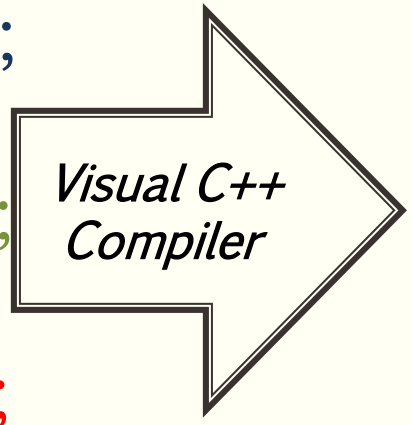
VECTORS GENERATION USING C++ CODE

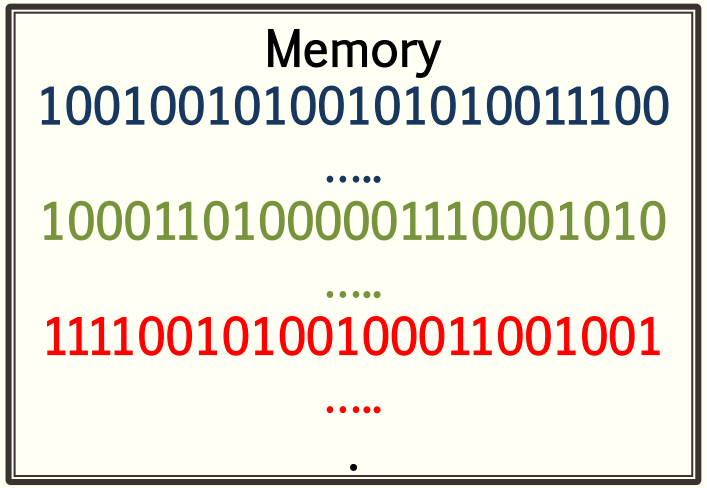
Double X;
+/-
Double Y;
=
Double Z;

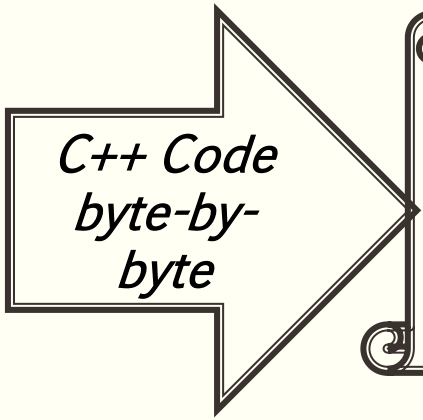


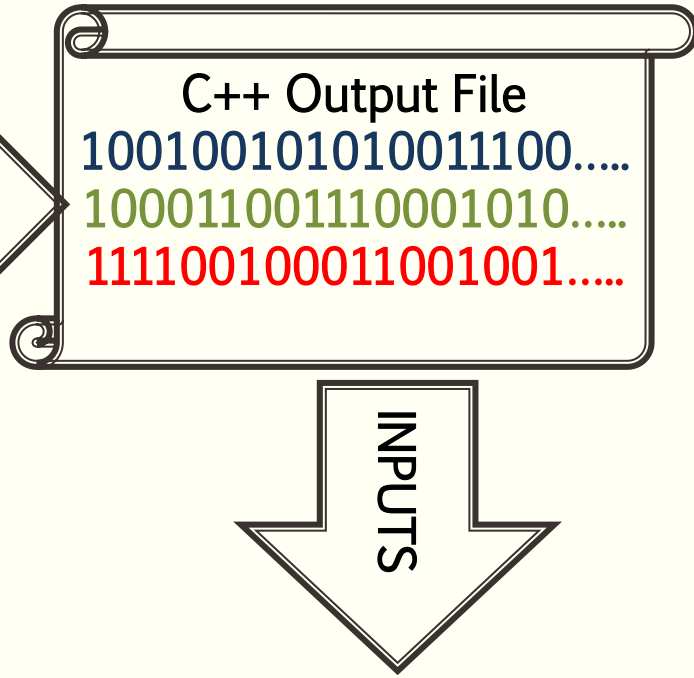
VECTORS GENERATION USING C++ CODE

Double X;
+/-
Double Y;
=
Double Z;



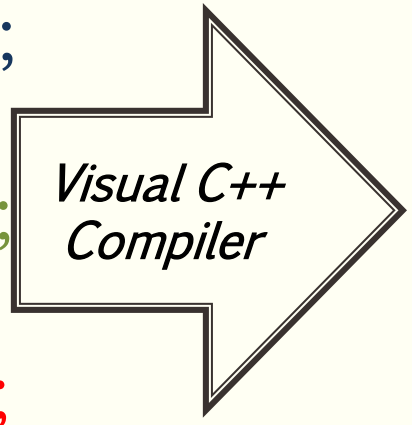


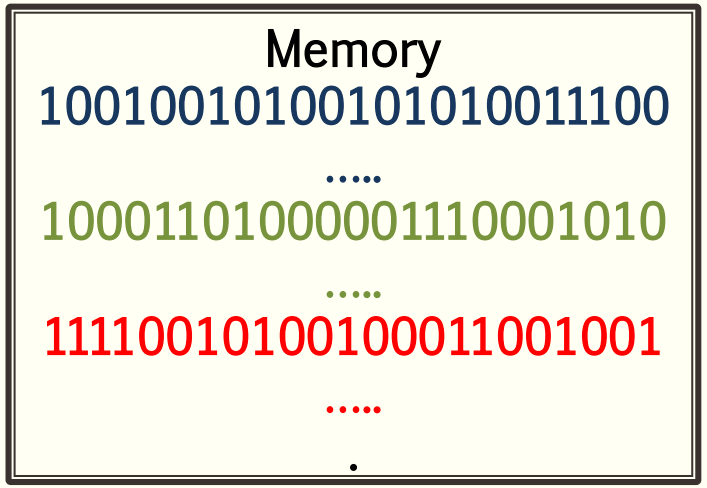


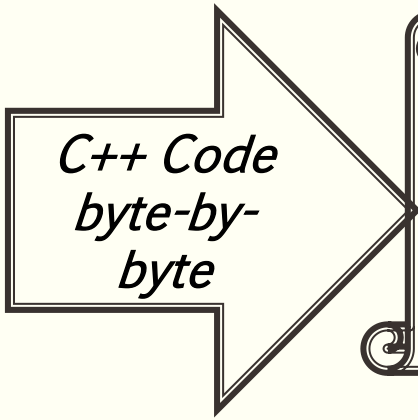


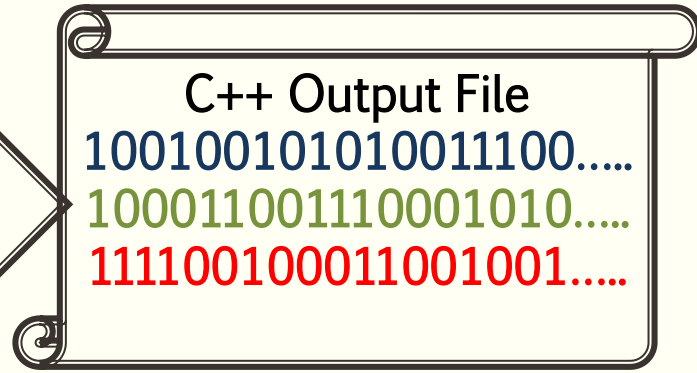
VECTORS GENERATION USING C++ CODE

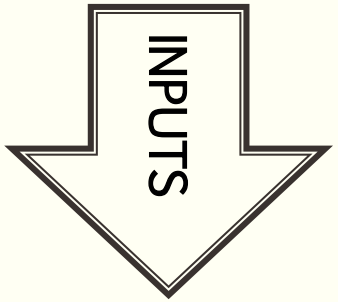
Double X;
+/-
Double Y;
=
Double Z;











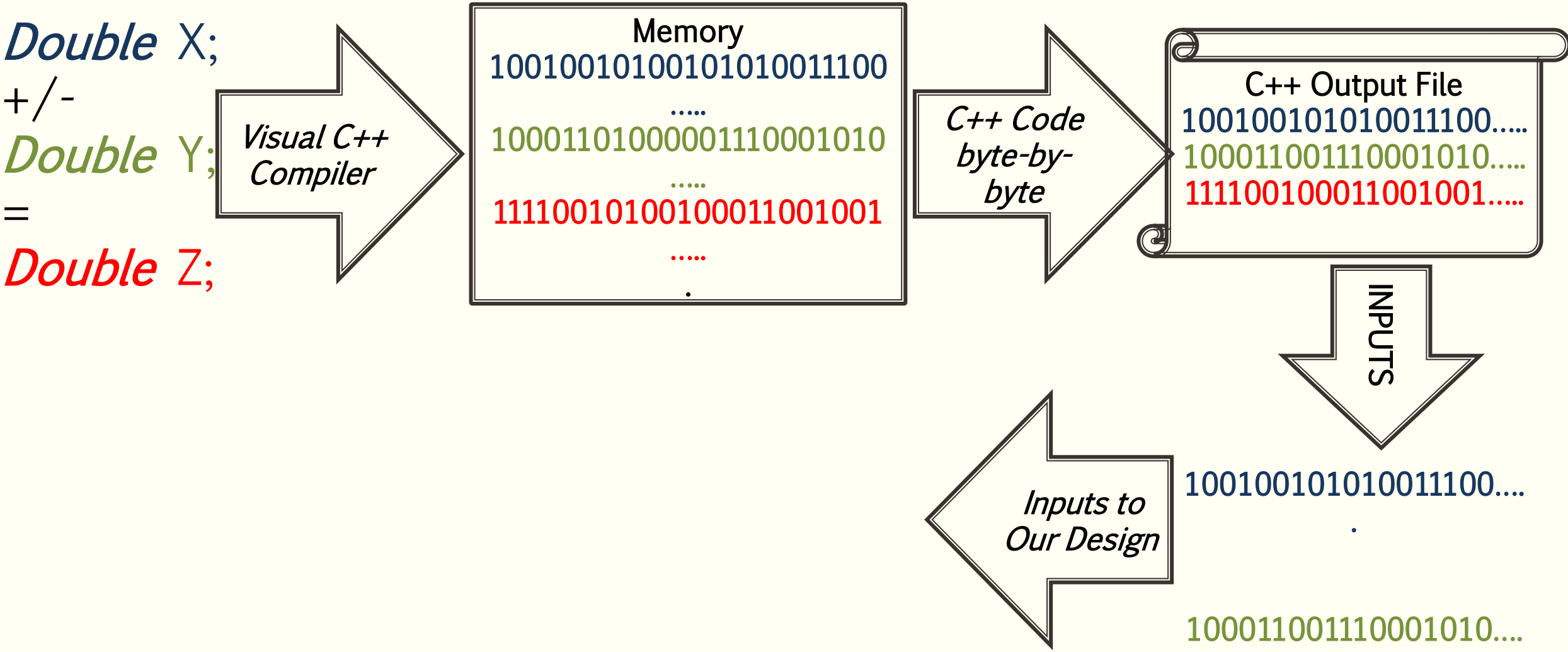
10010010101001110O....

.

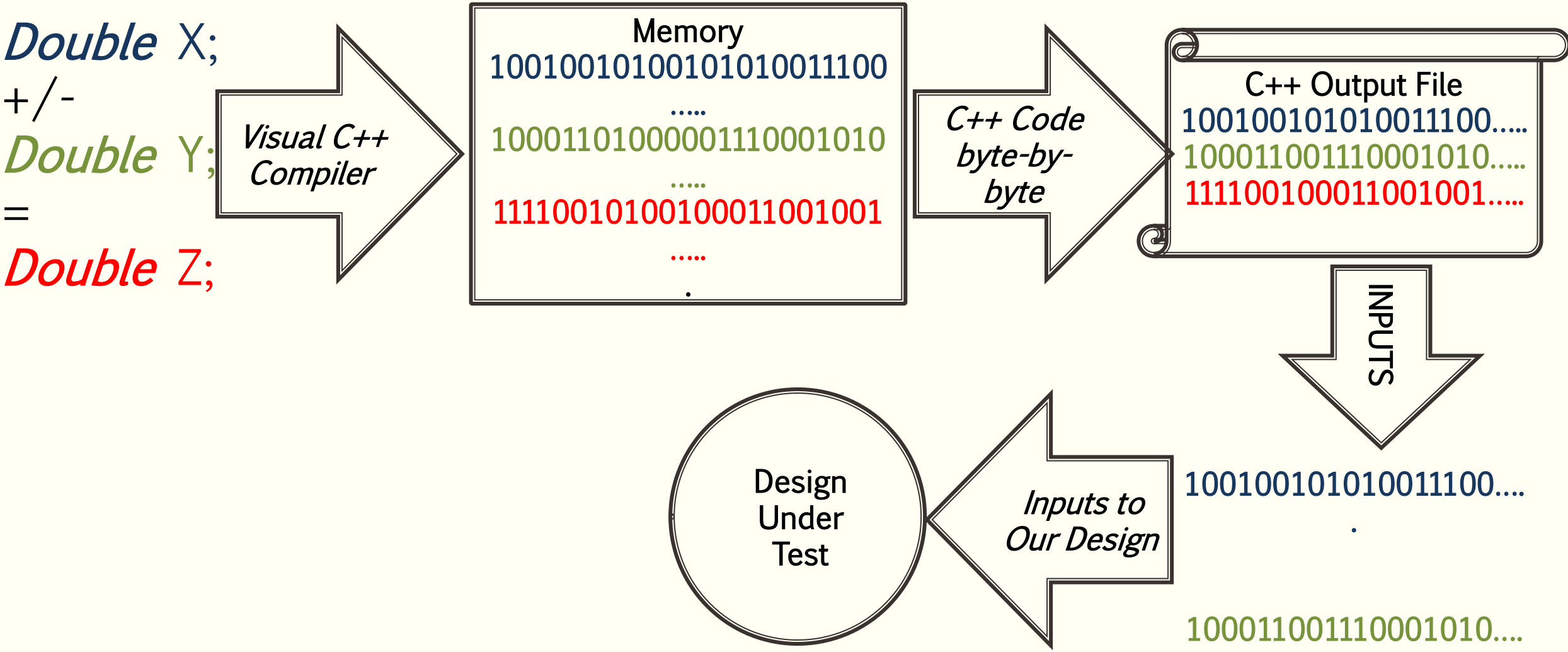
10001100111000101O....

.

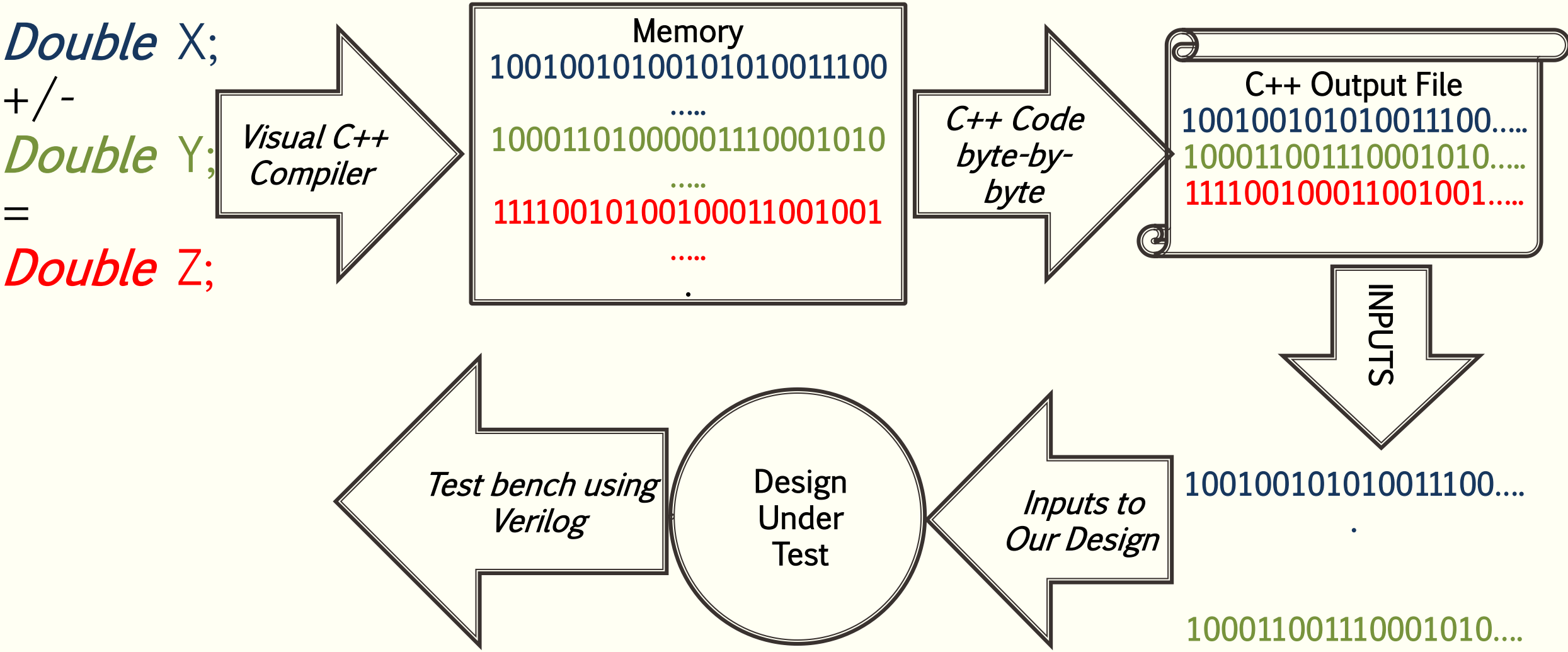
VECTORS GENERATION USING C++ CODE



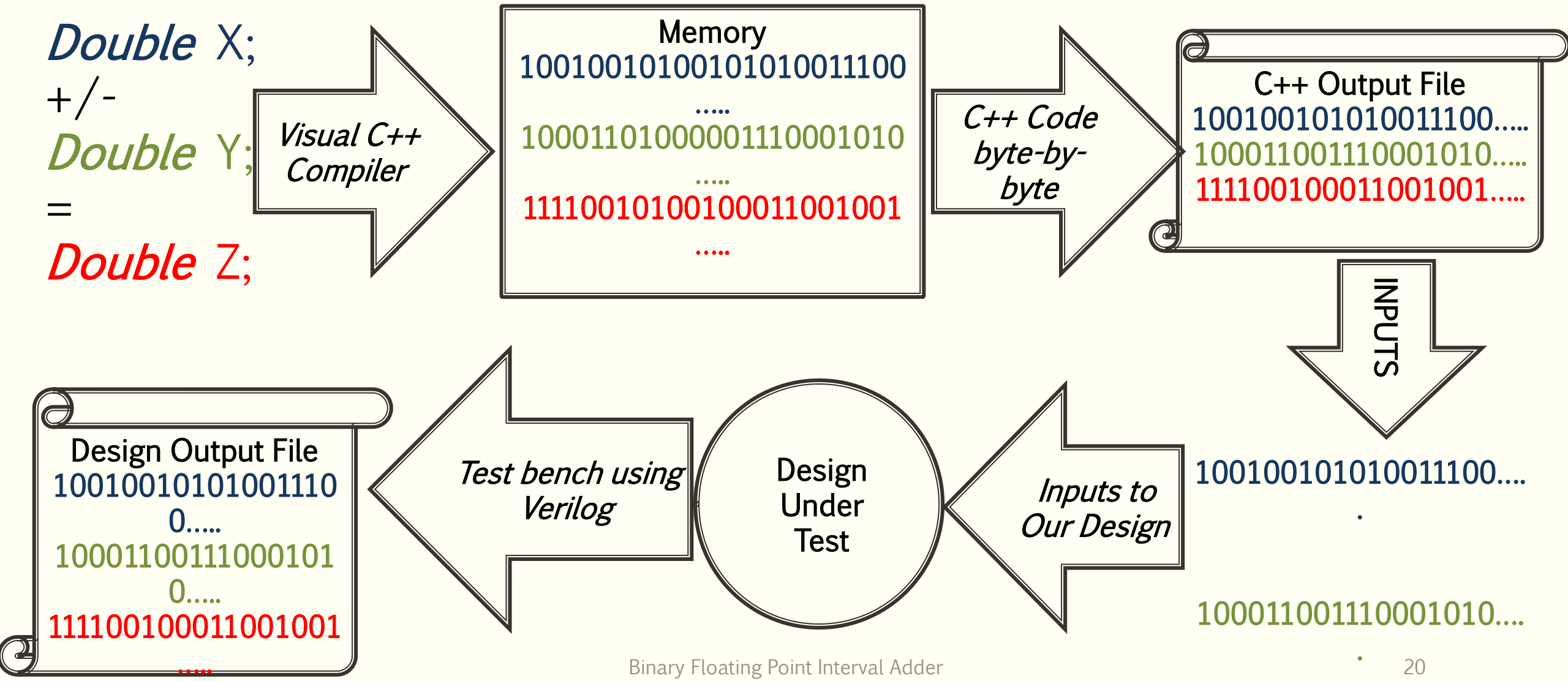
VECTORS GENERATION USING C++ CODE



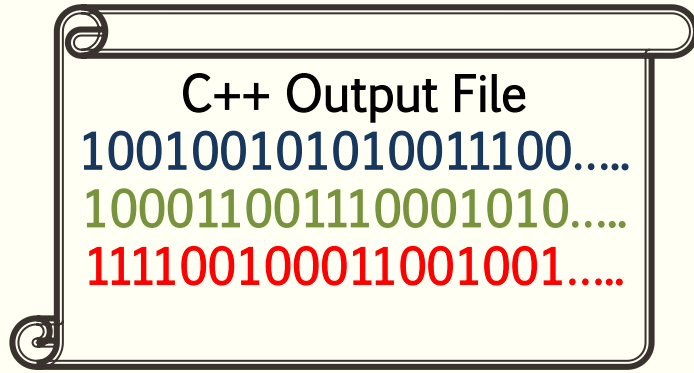
VECTORS GENERATION USING C++ CODE

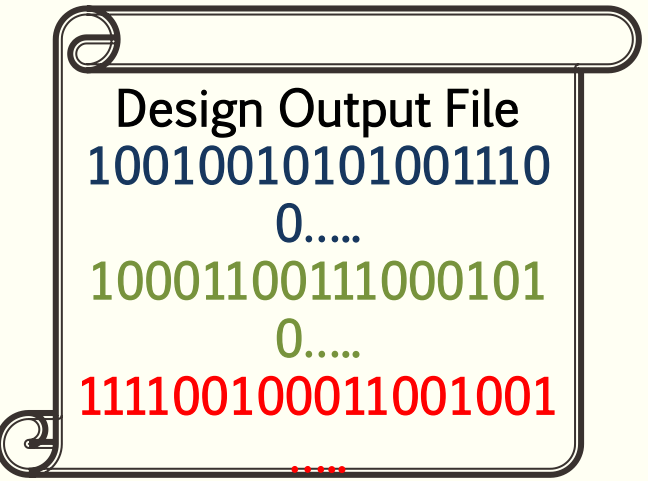


VECTORS GENERATION USING C++ CODE

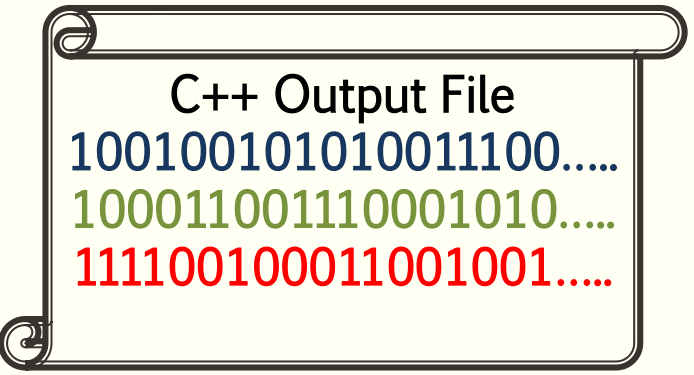


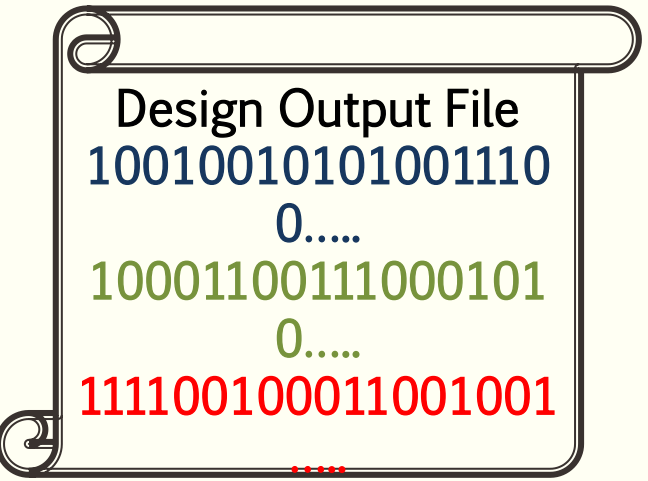
VECTORS GENERATION USING C++ CODE



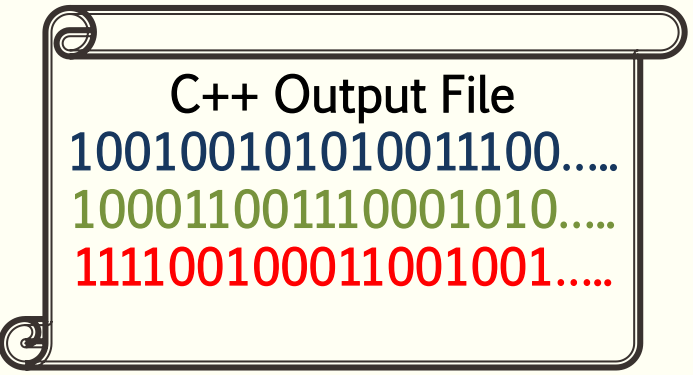


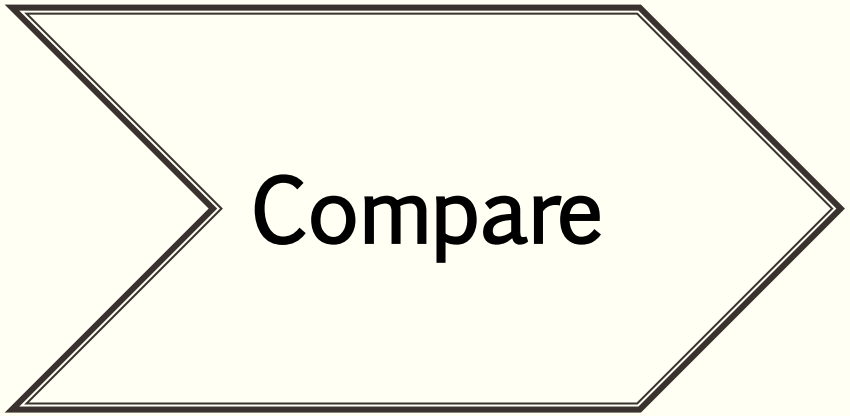
VECTORS GENERATION USING C++ CODE

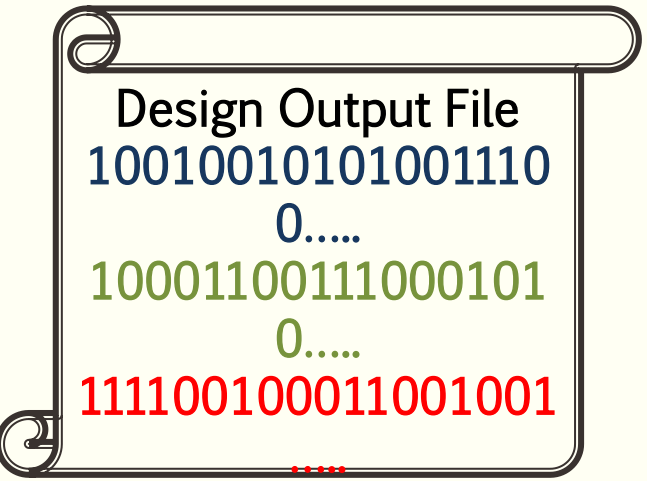




VECTORS GENERATION USING C++ CODE







VECTORS GENERATION USING C++ CODE

C++ Output File
10010010101001110 0.....
10001100111000101 0.....
11110010001100100 1.....

Compare

Design Output File
10010010101001110 0.....
10001100111000101 0.....
11110010001100100 1.....

Result Test File

Total No. of lines :
No. of Matched Lines :
No. of Unmatched Lines :

TESTING VECTORS

- Our Testing Vectors are formed from the different combinations of the following :
  - +ve/-ve ADD/SUB +ve/-ve .
  - Very Large Numbers / Very Small Numbers.
  - Near Path Testing / Far Path Testing .
  - Rounding Up / Rounding Down .
  - Sequential Numbers / Random Numbers .
  - Exceptions (NaNs , Subnormal , Denormalized , ….)

- The design passed over 25 test vectors of ~100,000 Inputs each.

Results & comparisons

Design / Parameter	Area		Clock Frequency (MHZ)	Pipelining Depth (Cycles)	Pipelining Throughput
	No. of internal ALUTs	No. of Reg.			
Single Path Adder/Subtractor	254	220	116.72	4	0.5

Note: Stratix III Family is used

Results & comparisons

Two path Design / Parameter	Area		Clock Frequency (MHZ)	Pipelining Depth (Cycles)	Pipelining Throughput
	No. of internal ALUTs	No. of Reg.			
Proposed Design	1125	798	283.05	4	1
Ayman's	1178	745	250	7	1

Note: Stratix III Family is used

Results & comparisons

MIBFP Adder/ Subtractor	Area		Clock Frequency (GHZ)	Pipelining Depth (Cycles)	Pipelining Throughput
	Combinational Area (µm)	Non-combinational Area (µm)			
Proposed design (65nm)	35588	21249	1.126	4	1
Proposed design (45nm)	17056	10184	1.624	4	1
Ayman's (45nm)	11300	8900	1.176	8	1

ASIC Simulation Results

Results & comparisons

Notes on Simulation results:

- Ayman’s design is the 1st implementation of Modal interval Binary floating point ADD/SUB.
- Our design was implemented using 65nm technology, and the values provided in 45nm technology are approximate.
- Area of the proposed design is a little bit larger than Ayman’s one, but the speed is higher.

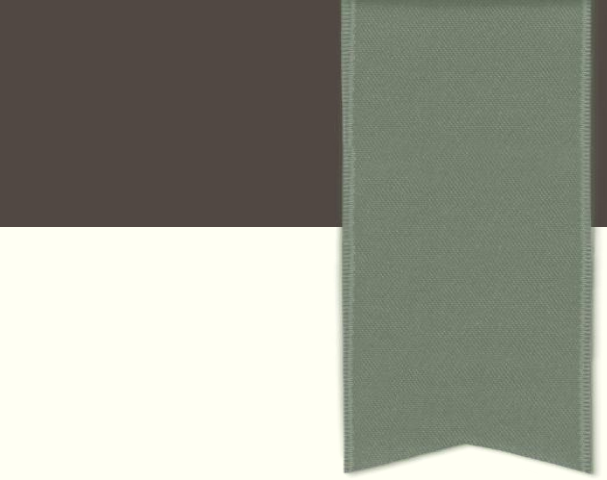
CONCLUSION

- This is the second hardware implementation of modal interval floating point adder.
- The proposed implementation features faster speed, and smaller area.
- This is the 1st implementation of the Decoration system.

What’s next?

- More enhancement of the speed and area of the proposed design.
- Implementing Decorations on Interval Floating point multiplier.
- Basic interval functions:
  - Trigonometric
  - Exponential
  - Logarithmic
- Multiple precision modal interval units.

Questions ?



Thank you