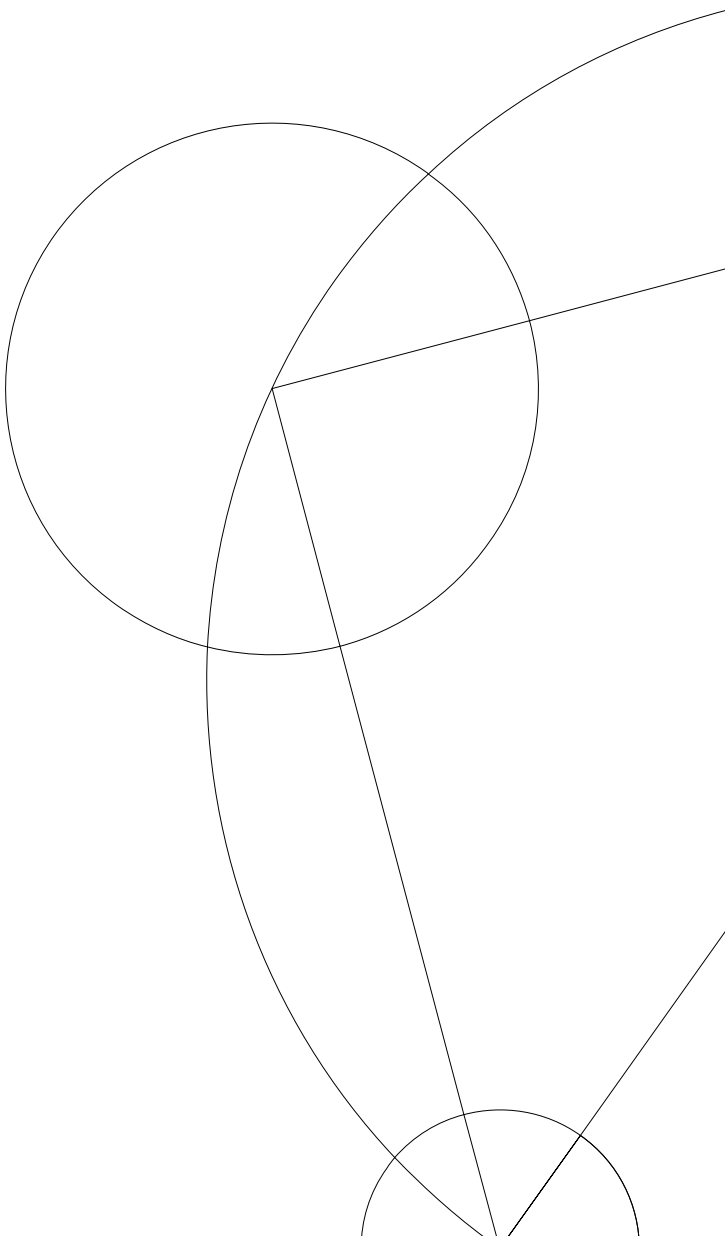
UNIVERSITY OF COPENHAGEN
Department of Computer Science (DIKU)

Project Outside the Course Scope

Closure Plots for Basic Arithmetics

Amr El Sayed
KuId - vwj159

September 18, 2016



Closure Plots for Basic Arithmetics

DIKU - Project Outside the Course Scope

Students Names	:	Amr El Sayed - VWJ159
University	:	University of Copenhagen
Institution	:	Department of Computer Science (DIKU)
General Supervisor	:	Michael Kirkedal Thmosen
Practical Supervisor	:	Oleksandr Shturmov
Period	:	Block 5
Year	:	2016
Pages	:	
Github	:	`https://github.com/Amr116/ClosurePlots`
Publishing site	:	`https://amr116.github.io/ClosurePlots/`

Certificate

This is to certify that the work contained in the thesis entitled "Closure Plots for Basic Arithmetics" by Amr El Sayed has been carried out under our supervision and that this work has not been submitted elsewhere.

General Supervisor
Michael Kirkedal Thmosen
m.kirkedal@di.ku.dk
Department of Computer Science (DIKU)
University of Copenhagen

Practical Supervisor
Oleksandr Shturmov
oleks@di.ku.dk
Department of Computer Science (DIKU)
University of Copenhagen

1 Abstract

Resume

Contents

2 Introduction

I am a third year computer science student at the University of Copenhagen. I'm currently enrolled in the undergraduate (bachelor) part of the education. This report is the final product of a 1-block Project Outside the Course Scope. This report covers the development process of Closure Plots for Basic Arithmetics on the aspects of floating-point that have a direct impact on the result of arithmetic operations of computer systems.

The main use of this software (Closure Plots for Basic Arithmetics which will be referred to as "CPBA" throughout this report.) is to illustrate the behaviour of basic arithmetic operations in Floating-Point arithmetic with different bit-width of web based software, so the user can access it virtually from anywhere.

This software is based on the principle of float in the floating point which is derived from the fact that there is no fixed number of digits before and after the decimal point. That is, the decimal point can float; therefore, the implementation of this project focuses on creating a visualization tool for illustrating the behaviour of basic arithmetic operations (Addition, Subtraction, Multiplication, Division) to integer with a fixed-size of bit string, which refer to the same size of bit-width (Half, Single, Double, Quadruple). Also, to create the infrastructure that will allow for future development; to make it possible to extend the project of illustrating the behaviour of decimal numbers, rational numbers, and all the irrational numbers in Floating-Point eg. IEEE Floating-Point Standard or something else with this approach.

The idea of this project inspired from John L. Gustafson book [3] (THE END of ERROR Unum Computing), where there is a graphical view of illustrating the behaviour of bit-width representation for basic arithmetic operations as shown in figure. 1.

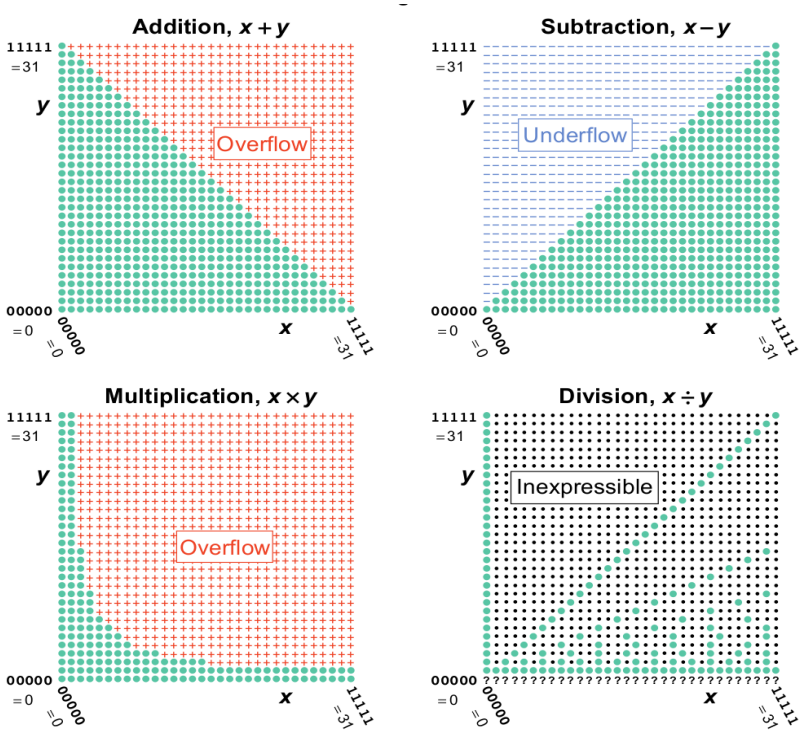


Figure 1: A graphical view of bit strings: Value and closure plots from THE END of ERROR Unum Computing book at page number 9

2.1 Intended Audience and Reading Suggestions

This report is intended for readers who have studied Computer Science or similar, or are simply interested in the field of IT. Readers are assumed to have basic programming knowledge and some familiarity with web development technologies.

2.2 Purpose

The purpose of this report is divided into two purposes:

The first purpose is to describe the UI to CPBA application for instructors and Computer Science students at the Department of Computer Science - University of Copenhagen, so they can easily interact with the application.

The second purpose is to explain the requirement specifications for CPBA application and to explain the steps of implementing this project to help the developers to further develop the application.

2.3 Project Scope

CPBA will allow students to learn about the behaviour of basic arithmetic operations in floating point arithmetic with different Bit-Width through interaction with UI. The CPBA will also allow the instructors and creative students to modify the variables in arithmetic operations - to monitor behaviour of floating point.

The objective of the CPBA application is to create a visualization tool that illustrates the behaviour of basic arithmetic operations (e.g., Addition, Subtraction, Multiplication, Division) in floating point arithmetic with different bit-width (e.g. Half, Single, Double, Quadruple).

3 Overall Description

Floating point is the formulaic representation of a real number that includes all the rational numbers, such as the integer (negative, postive), the fraction, and all the irrational numbers, such as square root or transcendental numbers.

Floating-point arithmetic is a subject that is considered exciting for many people, because the result of Floating-point arithmetic operations depends on the bit-width of the Floating-point. This is rather surprising because the floating-point bit-width can be the reason why it gives a different result. Almost every programming language has a floating-point datatype. Therefore, the use of two or more programming languages (for some common arithmetic calculations to develop a project) depend on Floating-point arithmetic operations that can give surprising results. The reason for that is, these programming languages have different bit-width and Floating-point datatypes, and also, bit-width differs from one computer to another computer. c.[1]

The most popular code for representing real numbers is called the IEEE Floating-Point Standard c.[2], - it has three basic components: the sign, the exponent, and the mantissa. The mantissa is composed of the fraction and an implicit leading digit. The exponent base (2) is implicit and does not need to be stored.

3.1 Product Perspective

A simple way to illustrate the behaviour of basic arithmetic operations is to physically sit in front of a computer or with paper and write down the numbers and finish the calculation process. There are tools available, commercial or open source, that support arithmetic operations to digits or binary numbers. Many operating systems and applications come with calculator support with basic arithmetic operations. However, all these tools usually require a person physically sitting in front of a computer writing down the numbers one by one.

Thus, there is a need for a web based application such as a visualization tool for illustrating the behaviour of basic arithmetic operations.

4 Analysis

In this section I will give an analysis of the problem described above. The analysis will focus on narrowing in special areas of the problem domain that will be central to designing and implementing a working prototype.

4.1 Fixed Bit String

4.2 Rectangle Size vs Bit Pixel

The browser window is the rectangle that a web page fills on screen - it is basically the size of the browser window, without the toolbars and scrollbars [4]. I will use this window to visualize and illustrate the behaviour of basic arithmetic of Floating-point. But is it possible for the CPBA application to illustrate the behaviour of this basic arithmetic operations in this window !?

I want to calculate arithmetic operations (e.g. Addition, Subtraction, Multiplication, Division) with integer numbers that have different bit-width (Half, Single, Double, Quadruple). To do these calculations I need at least two variables. That is, if the variables X and Y have Single bit-width I have to represent the result of this arithmetic operations, which results in $65536^2 = 4294967296$ elements.

This huge number of elements can not be shown in the Viewport while maintaining a clear vision for each element separately. So I need some way to allow us to achieve a clear vision for each element, and also allow the user to take advantage of this visualization tool for illustrating the behaviour of basic arithmetic operations.

5 Design

I decided that it should be web-based so the application is easily accessible for students. In other words, it should be accessed via the internet by using a standard web browser or students can access the project files from a physical source (e.g. flash memory). This way there is no need for distributing the program and all the required packages for it to run, which I would need if I had to make a program that was not web-based. By making the application web-based I ensure its accessibility for most, if not all users. The solution will not target mobile/tablet users specifically.

Now that the project is a web-based application I need to use HTML and CSS - seeing that they are the two core technologies for building Web pages. However, to build the func-

tionality of this project I can choose between many programming languages, but JavaScript is the only one that fits the CPBA requirements; This way the users can access the CPBA online, and offline if they get the project from a physical source.

I discussed some of the problems that I will be confronted with in the design and implementation of the project in the above section (Analysis). Therefore, in the below subsection I will present the problems mentioned above and explain my ideas for the design and implementation of the solution to the problems.

5.1 Arithmetic operations & Different bit-width

The user needs a simple way to illustrate the behaviour of basic arithmetic operations (Addition, Subtraction, Multiplication, Division) with different bit-width (Half, Single, Double, Quadruple). Therefore, the project should make it possible for users to select and choose between different arithmetic operations and different bit-width. For this reason, I've decided that the front-end design of CPBA should include two <select> elements which create a drop-down list, and thereby I collect user input from the selected option.

5.2 Mathematical notation

Since I have chosen javascript as the primary programming language to the functionality of this project, it will provide access for the user whether it be online or offline. But does Javascript represent bit-width such as the way in which it represent in mathematical notation? Unfortunately, The answer is No because the maximum integer represented in JavaScript is $2^{31} - 1$, or 2147483647, otherwise the integer will be represented in scientific notation. That's why I decided to make CPBA is limited to bit-width Half and Single only.

5.3 Elements

Representing fixed bit string equal to the length of bit string at bit-width Half and Single would produce huge number of elements. Those numbers would need to be shown in viewport, but to represent all those elements at once will not make it possible for users to see and illustrate the behaviour of basic arithmetic operations. Therefore, the CPBA application needs to summarize those elements to miniature form.
Those miniature forms differ from one bit-width to another bit-width in the number of stages. In other words, for example at the addition operation to two variables of bit-width Half, that have 2^{16} elements for variable X and 2^{16} for variable Y, so the sum of those two variables is 2^{32} elements.

I decided to show 256 elements of this result at each stage, where each element represents 4 edges from the real results as shown in figure 2.

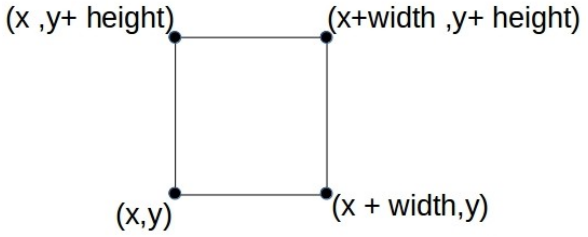


Figure 2: summarize form shows how each element represents in CPBA

The Half-bit-width will have four stages. At the first stage width and height will be equal to 4095, at the second stage they will be equal to 255, at the third stage they will be equal to 15, and at the fourth (last stage) they will be be equal to 0.

The Single-bit-width will have eight stages. At the first stage width and height will be equal to 268435455, at the second stage they will be equal to 16777215, at the third stage they will be equal to 1048575, at the fourth stage they will be be equal to 65535, at the fifth stage they will be be equal to 4095, at the sixth stage they will be be equal to 255, at the seventh stage they will be be equal to 15, and at the eighth stage (the last one) they will be be equal to 0

The below examples is illustrative examples to prove the concept of the addition operation with half bit-width at first stage form:

X = 4096, Y=53248

Leftmost bottom:	4096 + 53248	= 57344	,57344 is less then 2^{16}	Correct
Rightmost bottom:	(4096 + 4095) + 53248	= 61439	,61439 is less then 2^{16}	Correct
Leftmost upper:	4096 + (53248 + 4095)	= 61439	,61439 is less then 2^{16}	Correct
Rightmost upper:	(4096 + 4095) + (53248 + 4095)	= 65534	,65534 is less then 2^{16}	Correct

There are 4 correct and 0 Overflow, therefor i will show single Correct element represent this above range.

X = 4096, Y=57344

Leftmost bottom:	4096 + 57344	= 61440	,61440 is less then 2^{16}	Correct
Rightmost bottom:	(4096 + 4095) + 57344	= 65535	,65535 is less then 2^{16}	Correct
Leftmost upper:	4096 + (57344 + 4095)	= 65535	,65535 is less then 2^{16}	Correct
Rightmost upper:	(4096 + 4095) + (57344 + 4095)	= 69630	,69630 is more then 2^{16}	Overflow

There are 3 correct and 1 Overflow, therefor i will show single Mix element represent this above range.

X = 4096, Y=61440

Leftmost bottom:	4096 + 61440	= 65536	,65536 is more then 2^{16}	Overflow
Rightmost bottom:	(4096 + 4095) + 61440	= 69631	,69631 is more then 2^{16}	Overflow
Leftmost upper:	4096 + (61440 + 4095)	= 69631	,69631 is more then 2^{16}	Overflow
Rightmost upper:	(4096 + 4095) + (61440 + 4095)	= 73726	,73726 is more then 2^{16}	Overflow

There are 0 correct and 4 Overflow, therefore, i will show single Overflow element representing this above range.

5.4 Forward, Back & Reset

Since there will be many stages, whether it be for Half bit-width or Single bit-width. Therefore, users need to an mobility option, that can let the users move forward to next stage, back to previous stage or jump backward to first stage (Reset).

For achieving the mobility options to the user, I must provide a tool that allows the user to interact with presented data and the stages. This tools can be buttons, those buttons have to be available to user on UI.

To move forward to the next stage the user has to click on the element, thereafter, the clicked element will be the first element of the next stage, and so on until width and height are equal to zero. And, hereby the user has reached the final stage.

To move back to previous stage the user clicks on the back button - that button will provide the option of going back to the previous stage until width and height are equal to value 4095 in Half-bit-width or value 268435455 in Single-bit-width.

To reset or to jump back to the first stage will be done by clicking on reset button - that button will assign width and height to 4095 in Half-bit-width or 268435455 in Single-bit-width.

6 Implementation

In this section I will explain the involved steps of the CPBA application's implementation. The Implementation steps are split into two parts (User Interface-UI and Back-end)

6.1 UI

For the implementation of the user interface I have chosen to use Bootstrap[1]. Bootstrap is a front-end framework for web development, and helps me flesh out the user interface, ensuring that it looks the same no matter what browser the user is using to access the CPBA application.

In the UI I have implemented two select option elements, one to define the available options for bit-width, and the second one is to define the available options for arithmetic operations. The selection option element for the different bit-width has the onchange attribute that assigns to getBit-Width() JavaScript function - that function is responsible for creating arrays of the different width and height values. The HTML code for select option is shown in the code snippet 1 at Appendix section 9.

I defined "Plot my requests" button element at the same div element for the select option to get the value of the selected option. This button has attribute onclick, which is to assign to JavaScript functions getRequest(); and showService();. getRequest() function will get the value of the selected option and I will explain this function clearly at the Back-end section. But showService() function is responsible for showing Back and Reset buttons, because those buttons are invisible until the user has clicked on Plot my requests button. I assigned the style display of those buttons to "none", thereby showService() will assign style display to "block" instead of none, thereby those buttons will be shown in UI.
The HTML code for "Plot my requests" button is shown in the code snippet 2 at Appendix section 9, the HTML code for Back and Reset buttons is shown in the code snippet 3, and the JavaScript code for showService() function is shown in the code snippet 4 at Appendix section 9.

Here is the last version of UI that I created for the CPBA application. look at figure3

6.2 Back-end

In the design section I described that I would use javascript to implement the back-end functionality. The back-end consists of many functions, I will review these functions based on their order of the implementation to the CPBA application.

[1]Bootstrap, a front-end framework for web development ??

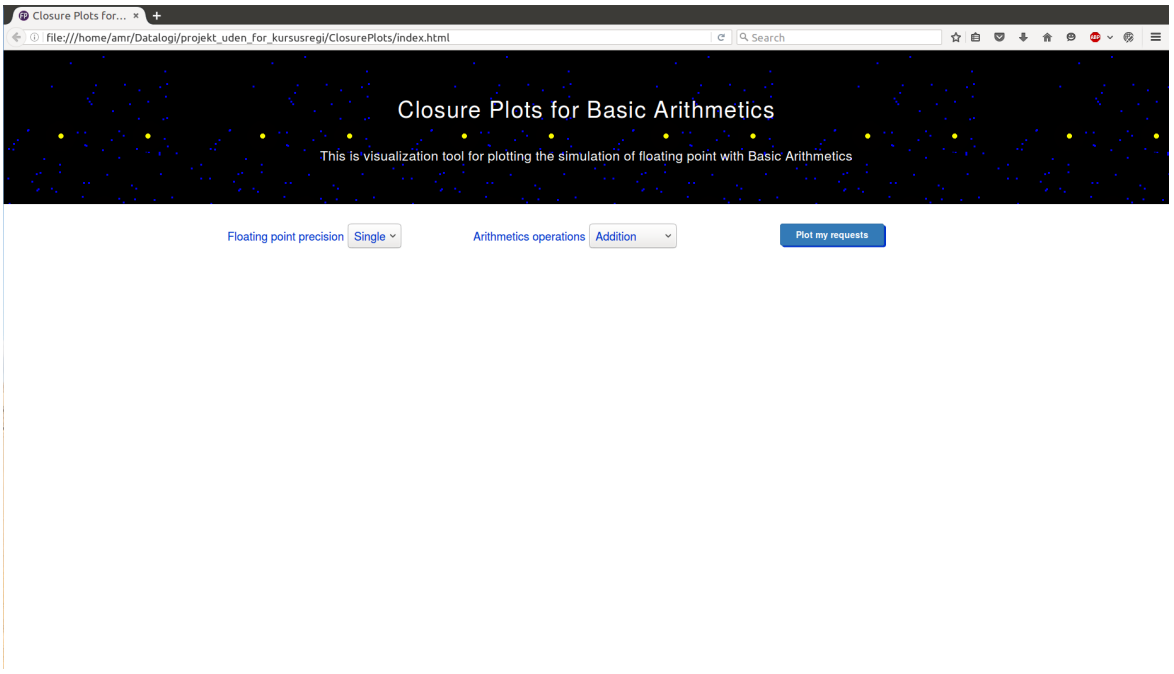


Figure 3: CPBA application UI

6.2.1 getBit-Width() function.

The first call occurs when the body of the webpage has been loaded. The second call occurs when the user changes the select option to bit-width. getbit-width receives the value of bit-width select option, which is 16 for Half bit-width or 32 for single bit-width. The function starts by initializing the global precArr variable to empty array, and initializing the global mobile variable to 0. Thereafter, assigns variable temp the return value of base 2 to the exponent power of bit-width value (16 or 32). Now the function is ready to create the width and height value to the selected bit-width, and that is done by a while loop as long as a specified condition ((temp/16) >= 1) is true. Inside the scope of while loop I divide the value of temp variable by 16, and then push the return value of temp -1 to precArr array. The while loop will create array of width and height values.

The return value for precArr variable when bit-width is half will be *[4095, 255, 15, 0]*, and the return value for precArr when bit-width is Single will be *[268435455, 16777215, 1048575, 65535, 4095, 255, 15, 0]*.

The JavaScript code for getBit-Width() function is shown in the code snippet5 at Appendix section 9.

6.2.2 getRequest() function.

This function is called when the user has clicked on the "Plot my requests" button. The function starts with defining a variable "op" and assigning this variable to the value of select option "operations", which are addition, subtraction, multiplication, or division. Thereafter, it defines a switch statement and evaluates variable "op " value to match the value to a case clause, and executes function associated with that case.
The JavaScript code for getBit-Width() function is shown in the code snippet6 at Appendix section 9.

6.2.3 function add()

The add() function calculates the sum of two variables or more in arithmetic operation. Also, it shows the behaviour of this arithmetic operation (Correct, overflow). It will be executed when it's called by function getRequest().
The body of add() function starts with a variable prec definition, this variable will be assigned to the value of the bit-width selected option. Thereafter, a variable range is defined - this variable will be assigned to the return value of base 2 to the exponent power of prec value. The height and width variables are assigned to the values of precArr[mobile], where precArr and mobile have been created from getBit-Width() function. Thereafter, variable index 0 is intialized to 0, and variable row is initialized to an empty array. The variable index will be used to insert specified items to array row. The variables f1 and f2 are defined to assign the values of X and Y to them after their values have been converted to binary format.

The next variable is getPixel, this variable declares a function expression in the current scope of add() function. getPixel will summarize 4 edges of 256 elements at each stage based on height and width value. In the current scope of getPixel function I define variable check. This variable declares a function expression - the function checks the sum of items at each edge for the 4 edges of 256 elements: if the result of the edge is less than the value of range variable the correct variable is increased by 1, otherwise the variable overflow is increased by 1.

The next step is to represent the result of the summarized data, and that is done by checking if the value of the correct variable is greater than 0 and the value of overflow variable is equal to 0, so the summarized data will be shown to user as it is correct data. Or, if the value of overflow variable is greater than 0 and the value of correct variable is equal to 0, so the summarized data will be shown to the user as it is overflow data. Other than that, the summarized data will be shown to the user as it is mixed data (correct and overflow).

At lines 40-42 of the snipped code, I defined variables xV and yV, those variables will be assigned to the value of variables xValue and yValue. The variables xValue and yValue have been defined as global and have been initialized to null value as shown in the snapped code 8 at Appendix section 9. Therefore, at the first stage xV and yV will be assigned to 0. However, if the user clicks on an item of the represented data, the values of this item will be assigned to xValue and yValue variables and thereby xV and yV will be assigned to xValue and yValue.

The getFormat variable is a declare function expression in the current scope of add() function. I defined getFormat function with one parameter, this parameter passes from prec variable to the decBin function 9, and the return value of decBin is assigned to the variables f1 and f2, where those variables will represent the binary value of x and y items of the shown data.

At lines 47-57 I define two for loops. The first loop represents the x-axis items, and the second loop represents the y-axis items. The first loop has 16 iterations, and in the beginning of the loop I started by assigning variable x to the value of variable xV. And, at each iteration the x value is updated to its value plus width value. Thereafter, I call getFormat function with x value to get the binary value of x, so i can show it to the user. The next variable is dicX, this variable is an object with two properties key and value, and it is defined like that to fit Google Chart API to represent the items. The value for v key is x value, and the value of f key is the binary value of f1 to f2.
The second loop is defined in the body of the first loop, it also has 16 iterations. I started the same way as the previous loop, but this time it assigns variable y to the value of variable yV, and at each iteration the y value updates to its value plus height value. Thereafter, I

call getFormat function with y value to get the binary value of y so i can show it to the user. I defined variable dicY as an object with two properties key and value. The value for v key is y value, and the value of f key is the binary value of f1 to f2. Now I can call getPixel to summarize the 4 edges, and that is done by passing variables x, y , height, width, dicX, and dicY to the getPixel function.
This two loop produces 256 elements, those elements were inserted at row array at getPixel function, therefore, at line 58 I called drawChart() function to visualize the data for the user. At the next subsection I will review and explain drawChart() 10 function and how it works.

6.2.4 function drawChart(arg1, arg2)

This function plots the received data from add, sub, mult, and divi functions. The function takes two arguments, the first argument is the desired display data, and the second argument is an array of legends to describe the shown data (e.g. correct, Overflow, Underflow, Inexpressible, infinity, and mix). While the function will get different lengths in the second argument, I need to check the length of the second argument to pass the right legend to the shown data. If the length of arg2 is equal to three the received data is from divi() function, and thereby the legend will refer to 4 different data (Correct, Inexpressible, Infinity, and Mix). Otherwise the legend will refer to 3 different data (Correct, (Overflow or Underflow), and Mix). Thereafter, adds new rows to the data table - the rows are arrays that have been passed to the drawChart parameter. Line 21 is to configure the options of the chart, where I defined the char title, width and height. At line 28 variable chart is an object of class google.charts.Scatter, where "chart_div1" div is used to contain the chart drawn.

In the scope of drawChart function I have defined selectHandler() function, this function will allow users to interact with the shown data by clicking on the items, and thereby I get the x and y values of the clicked item. Those values are assigned to the global variables xValue and yValue. At line 51 I have defined variable valid, and it's assigned to the return value of checkIndex() function. This function checks if the mobile variable has not run out of range to precArr length as shown in the snapped code 11 at Appendix section 9. If the return value of checkIndex is true then increase the value of variable clickIndex by 1, and call getRequest() function.

6.2.5 function sub()

The sub() function 12 calculates the subtraction of two variables or more in arithmetic operation. And, the function is to show the behaviour of this arithmetic operation (Correct, Underflow). The function will be executed when it's called from function getRequest() 6. The sub() function has the same definition like the add() function 7 except the logic for check() function.

At the scope of sub() function, the check() function checks if the result of left expression subtract the right-side expression for the items at the four edges of 256 elements. If the result is greater than zero and the result is less than or equal to range variable the correct variable increased by 1. Otherwise the variable underflow is increased by 1 as shown in the snapped code 12 lines 20-22 at Appendix section 9.

6.2.6 function mult()

The mult() function 13 calculates the multiplication of two variables in arithmetic operation. And, the function is to show the behaviour of this arithmetic operation (Correct

or Overflow). The function will be executed when it's called from function getRequest() 6. The mult() function has the same definition like the add() 7, and sub() 12 functions except the logic for check() function.

At the scope of mult() function, the check() function checks if the result of left-side expression multiplication the right-side expression for the items at the four edges of 256 elements. If the result is less than range variable the correct variable increased by 1. Otherwise the variable overflow is increased by 1 as shown in the snapped code 13 lines 20-22 at Appendix section 9.

6.2.7 function divi()

The divi() function 14 calculates the division of two variables in arithmetic operation. And, the function is to show the behaviour of this arithmetic operation (Correct, Inexpressible "Numbers do not divide evently", Infinity "Numbers divide by zero"). The function will be executed when it's called from function getRequest() 6. The divi() function has the same definition like the add() 7, sub() 12, and mult() 13 functions except the logic for check() function.

At the scope of divi() function, the check() function checks if the result of left-side expression divide by the right-side expression modulus 1 is equal to zero for the items at the four edges of 256 elements. If the condition is true the correct variable increased by 1, else if left-side expression divide by right-side expression is Not a Number (e.g. 0/0) or left-side expression divide by right-side expression is not finite (e.g. 1/0, 2/0 , .., n/0), then the fini variable increased by 1. Otherwise the variable Inexpressible increased by 1 as shown in the snapped code 13 lines 21-26 at Appendix section 9.

6.2.8 function reset()

The purpose of reset() 15 function is to reset chart and go back to first stage to show the previous data at this stage. The function reset() begins by reassign variable mobile value to zero, thereafter reassign global variables xValue and yValue values to null, and at last reassign variable clickIndex value to zero, then it call getRequest() function. The onclick event occurs when the user clicks on reset button at UI, then the reset() function will be execute.
The reset() function will be execute when onclick event occurs while the user clicks on reset button at UI. The code for reset function can see at 15 at the Appendix section 9.

6.2.9 function getBack()

7 Testing

8 Conclusion

References

[1]
Author : Goldberg, David
Title : What Every Computer Scientist Should Know About Floating-point Arithmetic
Journal : ACM Comput. Surv.
date : March 1991

[2]
Author : American National Standards Institute and Institute of Electrical and Electronice Engineers
Title : IEEE Standard for Binary Floating-Point Arithmetic
Date : 21.March 1985
url : http://www.wellposed.com/standards-documents-cache/IEEE754-1985.pdf

[3]
Author : John L. Gustafson
Title : THE END of ERROR Unum Computing
V-Date : 12.November 2014
ISBN : International Standard Book Number-13: 978-1-4822-3987-4 (eBook - PDF)

[4]
Author : James Royal-Lawson
Title : What is my viewport size and pixel density?
Date : 2013-05-05
url : http://beantin.se/what-is-my-viewport-size-pixel-density

Footnotes

- `http://steve.hollasch.net/cgindex/coding/ieeefloat.html`
- `https://developers.google.com/chart/`
- `http://getbootstrap.com/`

9 Appendix

Listing 1: Select option for bit-width and arithmetic operations

```
<div class="row">
    <div class="col-md-4">
      <font>Floating point bit-width</font>
      <select id="bit-width" onchange="getBit-Width();">
        <option value="16">Half </option>
        <option value="32">Single </option>
      </select>
    </div>

    <div class="col-md-4">
      <font>Arithmetics operations</font>
      <select id="operations" >
        <option value="addition">Addition</option>
        <option value="subtraction">Subtraction</option>
        <option value="multiplication">Multiplication</option>
        <option value="division">Division</option>
      </select>
    </div>
```

Listing 2: Plot my requests button to get the value of the select option

```
      <button type="button" class="btn btn-primary" onclick="getRequest();showService();"
        style="margin-top:1px;">
        <strong color="">Plot my requests</strong>
      </button>
</div>
```

Listing 3: Back and Reset buttons

```
  <div class="row" id="service" style="display:none;">
    <div class="col-md-2 col-md-offset-10">
      <div class="col-md-1" style="margin-right:10px;">
      <!--getBack()-->
        <button data-toggle="tooltip" data-placement="top" title="Back" type="button"
          class="btn btn-link btn-circle" onclick="getBack();"><!--getBack-->
          <img style="width: 43px; height: 43px;margin:-11px 2px 0px -6px;"
            src="img/playback.ico">
        </button>
      </div>

      <div class="col-md-1">
        <button data-toggle="tooltip" data-placement="top" title="Reset" type="button"
          class="btn btn-link btn-circle" onclick="reset();">
          <img style="width: 43px; height: 43px;margin:-12px 2px 0px -7px;" src="img/reset.ico">
        </button>
      </div>
    </div>
  </div>
```

Listing 4: showService function will change style of buttons to block

```
    function showService() {
      document.getElementById("service").style.display = "block";
    }
```

Listing 5: getBit-Width function

```
/* getBit-wWidth Function is to create array of the different dimensional to chart.
 * The function has to reset mobile variable to zero, which will point to the start of precArr.
 * The function will be execute onload of the body element and when a user changes the selected
 * option of a <select> element.
 */
var mobile = 0;
var precArr = [];

function getBit-Width(){
  precArr = [];
  mobile = 0;
  var temp= Math.pow(2, document.getElementById("bit-width").value);

  while((temp / 16) >= 1){
    temp /= 16;
    precArr.push(temp-1);
  }
}
```

Listing 6: getRequest function

```
/* getRequest function is to create object of the selected options
 * and call the appropriate function depending on user request.
 */
function getRequest(){
  var op = document.getElementById("operations").value;
  switch(op){
    case "addition":
      add();
      break;
    case "subtraction":
      sub();
      break;
    case "multiplication":
      mult();
      break;
    case "division":
      divi();
      break;
    default:
      alert('Something went wrong!\n'
        + 'Please contact the publisher and let the author know about the bug.\n'
        + 'amr11682@hotmail.com');
      break;
  }
}
```

Listing 7: add function

```
// function to create the simulation of Arithmetics operation ( addition )
function add(){
  var prec = parseInt(document.getElementById("bit-width").value);

  var range = Math.pow(2, prec);

  var height= precArr[mobile];
  var width = precArr[mobile];

```

```
  var index = 0;
  var rows = [];

  var f1; // X-axis data format
  var f2; // Y-axis data format

  var getPixel = function(x, y, height, width, dictX, dictY) {
    var correct = 0;
    var overflow = 0;

    var check = function(x,y) {
      if (x + y < range) { correct += 1; } else { overflow += 1; }
    }
    check(x, y);
    check(x, y + height);
    check(x + width, y);
    check(x + width, y + height);
    if (correct > 0 && overflow == 0) {
        // Correct data
      rows[index] = [dictX, dictY, null, null];

    } else if(overflow > 0 && correct == 0) {
      // Overflow data
      rows[index] = [dictX, null, dictY, null];

    } else {
      // Mix data
      rows[index] = [dictX, null, null, dictY];
    }
  }
  var xV;
  var yV;
  if(xValue == null && yValue == null){xV=0; yV=0;}else{xV = xValue; yV = yValue;}
  if(clickIndex > 0){clickedX.push(xV);clickedY.push(yV);}

  var getFormat = function(param){f1 = decBin(param, prec); f2 = decBin(param+height, prec);}
  //var x = 0
  for(var x = xV, i = 0; i < 16; i++, x += width+1){
    getFormat(x);
    var dictX = {v:x, f:'X: '+f1+' to '+f2};
    //var y = 0
    for(var y = yV, j = 0; j < 16; j++, y += height+1){
      getFormat(y);
      var dictY = {v:y, f:'Y: '+f1+' to '+f2};
      getPixel(x, y, height, width, dictX, dictY);
      index += 1;
    }
  }
  drawChart(rows, ['Overflow', 'Mix']);
}
```

Listing 8: Global variables

```
// Declare global variable, which will get their value from drawChart function,
// only if the user has interacted with chart ( click on (x,y) ).
var xValue = null;
var yValue = null;

var clickedX = [];
```

```
var clickedY = [];
var clickIndex = 0;
```

Listing 9: Convert (Decimal to Binary) with specify length of the output

```
/* decBin function is to convert (Decimal to Binary) with specify length of the output.
 * The function take 2 arguments ( dec, length)
 * dec : is the decimal number, which need to be convert to binary
 * length: is the desirable bit-width.
 * The function convert positive decimal as well as negative decimal
 */
function decBin(dec,length){
  var out = "";
  while(length--)
    out += (dec >> length ) & 1;
  return out;
}
```

Listing 10: drawChart function will visualize the passed data to user.

```
function drawChart(arg1, arg2) {
  // Define the chart to be drawn.
  var data = new google.visualization.DataTable();
  data.addColumn('number', '');
  data.addColumn('number', 'correct');
  if (arg2.length == 3){
    // numbers do not divide evenly (Inexpressible)
    data.addColumn('number', arg2[0]);
    // divide by zero
    data.addColumn('number', arg2[1]);
    // Mixed
    data.addColumn('number', arg2[2]);
  } else{
    // Overflow or Underflow
    data.addColumn('number', arg2[0]);
    // Mixed
    data.addColumn('number', arg2[1]);
  }

    data.addRows(arg1);
    var options = {
      chart: {
        title: ' Plotting the simulation of floating point with Basic Arithmetics'
      },
      width: 1150,
      height: 500,
    };
    var chart = new google.charts.Scatter(document.getElementById('chart_div1'));

    chart.draw(data, google.charts.Scatter.convertOptions(options));


  // Listen for the 'select' event, and call my function selectHandler() when
  // the user selects something on the chart.

    google.visualization.events.addListener(chart, 'select', selectHandler);

    // This function to get the index of the selected element of chart.
  function selectHandler() {
```

```
40
41      mobile += 1;
42
43      var selectedItemRow = chart.getSelection()[0].row;
44      var selectedItemCol = chart.getSelection()[0].column;
45
46      if (selectedItemRow != null){
47
48        xValue = data.getValue(selectedItemRow, 0);
49        yValue = data.getValue(selectedItemRow, selectedItemCol);
50
51        var valid = checkIndex();
52
53        if(valid){
54          clickIndex += 1;
55          getRequest();//xValue, yValue);
56        }
57      }
58    };
59  }
```

Listing 11: check if the mobile variable has not run out of range to precArr length.

```
1  /* checkIndex function is to check if the mobile variable
2   * has not run out of range to precArr length.
3   */
4  function checkIndex(){
5    if(mobile < precArr.length){return true;}else{return false;}
6  }
```

Listing 12: sub function

```
1  // function to create the simulation of Arithmetics operation ( subtraction )
2  function sub(){//xValue, yValue){
3    var prec = parseInt(document.getElementById("bit-width").value);
4    // range to iteration (loop)
5    var range = Math.pow(2, prec);
6
7    var height= precArr[mobile];
8    var width = precArr[mobile];
9
10   var index = 0;
11   var rows = [];
12
13   var f1; // axis data format
14   var f2; // axis data format
15
16   var getPixel = function(x, y, height, width, dictX, dictY) {
17     var correct = 0;
18     var underflow = 0;
19
20     var check = function(x,y) {
21       if ((x - y) >=0 && (x - y) < range) { correct += 1; } else { underflow += 1; }
22     }
23     check(x, y);
24     check(x, y + height);
25     check(x + width, y);
26       check(x + width, y + height);
27         console.log(x,y,correct, underflow, dictX, dictY, index);
```

```
      if (correct > 0 && underflow == 0) {
        // Correct data
      rows[index] = [dictX, dictY, null, null];

    } else if(underflow > 0 && correct == 0) {
      // Overflow data
      rows[index] = [dictX, null, dictY, null];

    } else {
      // Mix data
      rows[index] = [dictX, null, null, dictY];
    }
  }
  var xV;
  var yV;
  if(xValue == null && yValue == null){xV=0; yV=0;}else{xV = xValue; yV = yValue;}
  if(clickIndex > 0){clickedX.push(xV);clickedY.push(yV);}

  var getFormat = function(param){f1 = decBin(param, prec); f2 = decBin(param+height, prec);}
  //var x = 0
  for(var x = xV, i = 0; i < 16; i++, x += width+1){
    getFormat(x);
    var dictX = {v:x, f:'X: '+f1+' - '+f2};
    //var y = 0
    for(var y = yV, j = 0; j < 16; j++, y += height+1){
      getFormat(y);
      var dictY = {v:y, f:'Y: '+f1+' - '+f2};
      getPixel(x, y, height, width, dictX, dictY);
      index += 1;
    }
  }
  drawChart(rows, ['Underflow', 'Mix']);
}
```

Listing 13: mult function

```
// function to create the simulation of Arithmetics operation ( multiplication )
function mult(){//xValue, yValue){
  var prec = parseInt(document.getElementById("bit-width").value);
  // range to iteration (loop)
  var range = Math.pow(2, prec);

  var height= precArr[mobile];
  var width = precArr[mobile];

  var index = 0;
  var rows = [];

  var f1; // axis data format
  var f2; // axis data format

  var getPixel = function(x, y, height, width, dictX, dictY) {
    var correct = 0;
    var overflow = 0;

    var check = function(x,y) {
      if (x * y < range) { correct += 1; } else { overflow += 1; }
    }
    check(x, y);
```

```
    check(x, y + height);
    check(x + width, y);
    check(x + width, y + height);
    if (correct > 0 && overflow == 0) {
        // Correct data
      rows[index] = [dictX, dictY, null, null];

    } else if(overflow > 0 && correct == 0) {
      // Overflow data
      rows[index] = [dictX, null, dictY, null];

    } else {
      // Mix data
      rows[index] = [dictX, null, null, dictY];
    }
  }
  var xV;
  var yV;
  if(xValue == null && yValue == null){xV=0; yV=0;}else{xV = xValue; yV = yValue;}
  if(clickIndex > 0){clickedX.push(xV);clickedY.push(yV);}

  var getFormat = function(param){f1 = decBin(param, prec); f2 = decBin(param+height, prec);}

  for(var x = xV, i = 0; i < 16; i++, x += width+1){
    getFormat(x);
    var dictX = {v:x, f:'X: '+f1+' to '+f2};
    //var y = 0
    for(var y = yV, j = 0; j < 16; j++, y += height+1){
      getFormat(y);
      var dictY = {v:y, f:'Y: '+f1+' to '+f2};
      getPixel(x, y, height, width, dictX, dictY);
      index += 1;
    }
  }
  drawChart(rows, ['Overflow', 'Mix']);
}
```

Listing 14: divi function

```
// function to create the simulation of Arithmetics operation ( division )
function divi(){//xValue, yValue){
  var prec = parseInt(document.getElementById("bit-width").value);
  // range to iteration (loop)
  var range = Math.pow(2, prec);

  var height= precArr[mobile];
  var width = precArr[mobile];

  var index = 0;
  var rows = [];

  var f1; // axis data format
  var f2; // axis data format

  var getPixel = function(x, y, height, width, dictX, dictY) {
    var correct = 0;
    var infi = 0;
    var Inexpressible = 0;

```

```
    var check = function(x,y) {
      if ((x/y) % 1 == 0) { correct += 1; }
      // Check if the result if NAN (e.g. 0/0) or the result is Infinity (e.g. 1/0, 2/0, ...)
      else if(isNaN(x/y) || !(isFinite(x/y))) { infi += 1;}
      else{ Inexpressible += 1; }
    }
    check(x, y);
    check(x, y + height);
    check(x + width, y);
    check(x + width, y + height);
    if (correct > 0 && Inexpressible == 0 && infi == 0) {
        // Correct data: does (1/3) which is 0.33.. mod 1 equal to zero
      rows[index] = [dictX, dictY, null, null, null];

    } else if(Inexpressible > 0 && correct == 0 && infi == 0) {
      // Inexpressible data
      rows[index] = [dictX, null, dictY, null, null];

    }else if(infi > 0 && correct == 0 && Inexpressible == 0) {
      // check if is NAN eg. 0/0 or is Infinity(not finite) eg. 1/0 , 1000/0
      rows[index] = [dictX, null, null, dictY, null];

    } else {
      // Mix data
      rows[index] = [dictX, null, null, null, dictY];
    }
  }
  var xV;
  var yV;
  if(xValue == null && yValue == null){xV=0; yV=0;}else{xV = xValue; yV = yValue;}
  if(clickIndex > 0){clickedX.push(xV);clickedY.push(yV);}

  var getFormat = function(param){f1 = decBin(param, prec); f2 = decBin(param+height, prec);}

  for(var x = xV, i = 0; i < 16; i++, x += width+1){
    getFormat(x);
    var dictX = {v:x, f:'X: '+f1+' to '+f2};

    for(var y = yV, j = 0; j < 16; j++, y += height+1){
      getFormat(y);
      var dictY = {v:y, f:'Y: '+f1+' to '+f2};
      getPixel(x, y, height, width, dictX, dictY);
      index += 1;
    }
  }
  drawChart(rows, ['Inexpressible', 'infinity','Mix']);
}
```

Listing 15: reset function is to reset all the interactive action from user with chart, and get back to the first stage

```
/* reset function is to reset all the interactive action from user with chart.
 * set the mobile variable to zero index ( the start of precArr )
 */
function reset(){
  mobile = 0;
  xValue = null;
  yValue = null;
  clickIndex = 0;
```

```
9    getRequest();
10 }
```

Listing 16: getBack function provide to the user the previous stage of chart

```
1  /* getBack function provide to the user the previous event of chart.
2   * The function provide event points to the appropriate function (current )
3   */
4  function getBack(){
5    if(mobile != 0){ mobile -= 1;}else{ mobile = 0;}
6
7    if(clickIndex == 0 || clickIndex == 1){
8      xValue= null;
9      yValue=null;
10     clickIndex = 0;
11   }else{
12     clickIndex -= 1;
13     xValue=clickedX[clickIndex-1];
14     yValue=clickedY[clickIndex-1];
15
16   }
17   getRequest();
18 }
```