PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100 Feet Ring Road, BSK III Stage, Bengaluru-560 085

Department of Electronics and Communication Engineering

Course Title:
RISC-V Architecture

Course Code:
UE21EC352a

Teacher:
Prof. H.R.Vanamala

Project Title:
Implementation of Bubble sort algorithm in RISC-V assembly

Done By:
Achyuth S.S - PES1UG21EC010
Archanaa A Chandaragi -PES1UG21EC056

Index

Introduction

The implementation of efficient sorting algorithms is crucial in various computer science applications. One such algorithm, Bucket sort, stands out for its ability to quickly sort elements with a uniform distribution. In this project, we aim to explore the implementation of the Bucket sort algorithm specifically tailored for the RISC-V architecture. By leveraging the unique features and capabilities of RISC-V, we seek to optimize the performance and efficiency of this sorting algorithm, ultimately enhancing its applicability in real-world scenarios.

Bucket sort Algorithm

Bucket sort is a sorting algorithm that divides the elements into several groups called buckets. Once these elements are scattered into buckets, then sorting is done within each bucket. Finally, these sorted elements from each bucket are gathered, in the right order, to get the sorted output.

The number of buckets depends on the kind of data we are dealing with.

We can also claim that Bucket sort follows what is called the *Scatter-Gather method*.

The Bucket sort algorithm works as follows:

a. Let us consider an array of “n” elements
b. Now, we create B buckets, each initialized with zero values
c. We then assign the range of elements that each bucket can hold.
d. **Scatter**: Find the range each element belongs to and put each element into the bucket meant for its range
e. **Sort**: Iterate over all the buckets, and sort elements with buckets.
f. **Gather** the sorted elements from each bucket.

The optimal value of B, can be decided based on the following:

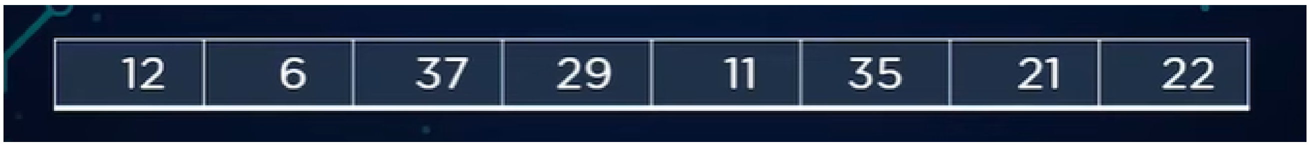
1. It should be easy to create a relationship between bucket index and numbers such that a range of numbers falls into each bucket.
2. If the nature of the distribution of input elements is known, the number of buckets and range should be such that there’s a roughly uniform distribution of elements in each range.

A pseudo code for Bucket sort is given below:

```
function bucketSort(array, k) is
    buckets ← new array of k empty lists
    M ← 1 + the maximum key value in the array
    for i = 0 to length(array) do
        insert array[i] into buckets[floor(k × array[i] / M)]
    for i = 0 to k do
        nextSort(buckets[i])
    return the concatenation of buckets[0], ...., buckets[k]
```

Considering an example to sort an array of elements using Bucket sort, it is as below:

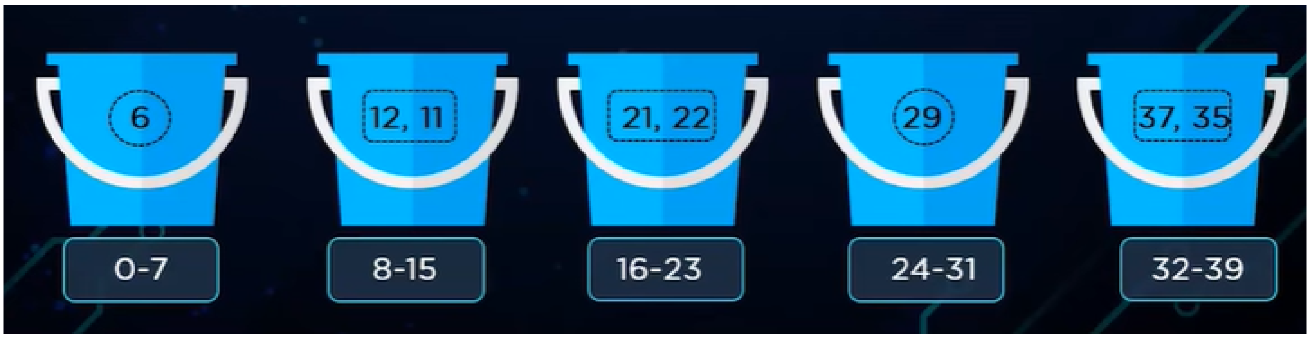
Consider the array as below



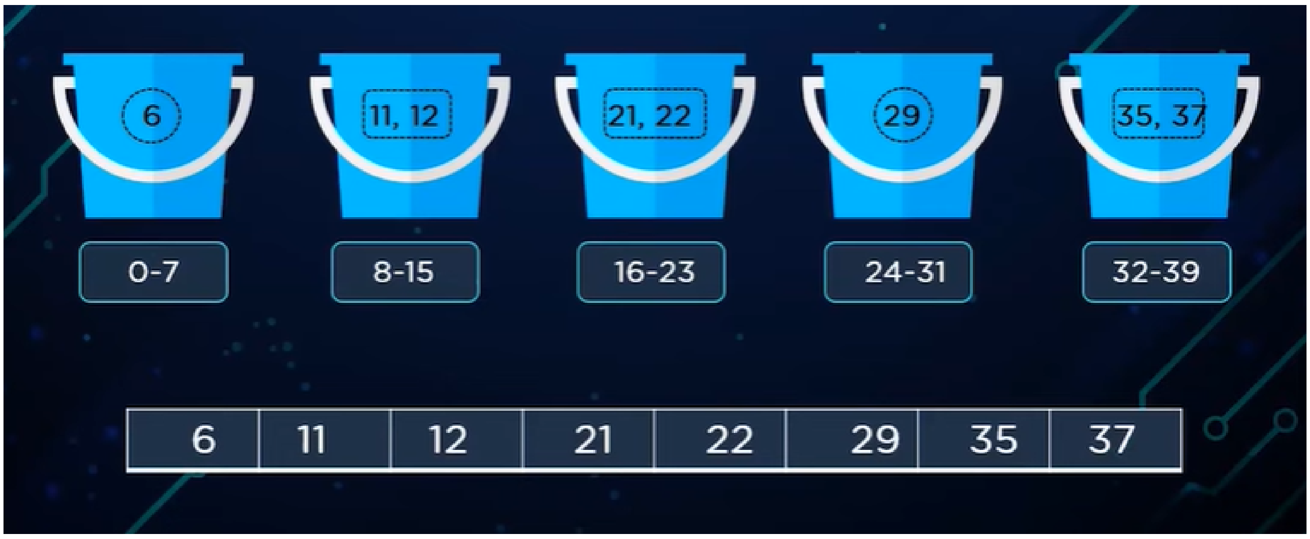
12	6	37	29	11	35	21	22

The range of a bucket is given as : (max - min) / n where n is the number of buckets.

For this example, range = (37 - 9) / 5 = 7.6 ~ 8



Collect all buckets,



Time and space complexity analysis

Case	Time
Best Case	O(n + k)
Average Case	O(n + k)
Worst Case	$O(n^2)$

Best Case Complexity - It occurs when there is no sorting required, i.e. the array is already sorted. In Bucket sort, best case occurs when the elements

are uniformly distributed in the buckets. The complexity will be better if the elements are already sorted in the buckets.
If we use the insertion sort to sort the bucket elements, the overall complexity will be linear, i.e., O(n + k), where O(n) is for making the buckets, and O(k) is for sorting the bucket elements using algorithms with linear time complexity at best case.
The best-case time complexity of bucket sort is **O(n + k)**.

Average Case Complexity - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. Bucket sort runs in linear time, even when the elements are uniformly distributed. The average case time complexity of bucket sort is **O(n + K)**.

Worst Case Complexity - In bucket sort, worst case occurs when the elements are of the close range in the array, because of that, they have to be placed in the same bucket. So, some buckets have more number of elements than others.
The complexity will get worse when the elements are in the reverse order.
The worst-case time complexity of bucket sort is **O(n^2)**.

Space Complexity	O(n*k)
Stable	YES

The space complexity of bucket sort is **O(n*k).**

Implementation of The Bucket sort

C Code:

```
#define N 10

#define MAX 113


int array[N] = {90, 80, 70, 60, 5, 4, 3, 2, 1, MAX};


int getMax(int a[], int n){

  int max = a[0];

  for (int i = 1; i < n; i++)

    if (a[i] > max)

      max = a[i];

  return max;

}


void bucket(int a[], int n){

  int max = getMax(a, n);

  int bucket[MAX];

  for (int i = 0; i <= max; i++){

      bucket[i] = 0;

  }

  for (int i = 0; i < n; i++){

      bucket[a[i]]++;
```

```
 }
  for (int i = 0, j = 0; i <= max; i++){
    while (bucket[i] > 0){
        a[j++] = i;
        bucket[i]--;
    }
  }
}

int main( ){
    bucket(array,N);
    return 0;
}
```

RISC-V alp Code :

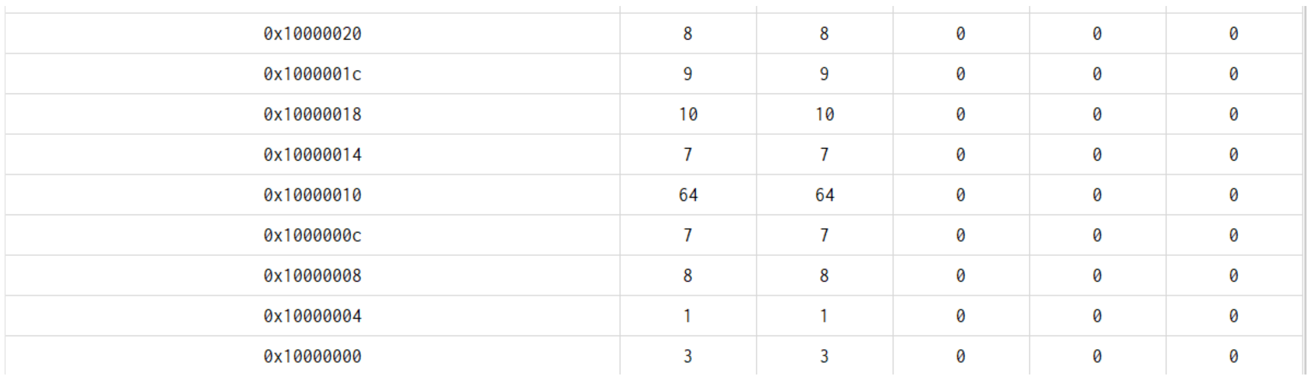
```
.data
array: .word 3,1,8,7,64,7,10,9
n: .word 8
.text

start:
la x6,array #x6 contains the array
lw x7,n # x7 = 8
addi x7,x7,-1 #x7=7
li x8,4 # a common multiplier to traverse through
array, so we are just keeping it for now
```

```
mul x9,x7,x8
add x10, x6,x9

addi x14,x7,0 #x14 => loop variable =7
addi x19,x0,7 # another loop variable for "next"
lw x11,0(x10)
li x23,0

lw x12,0(x10)
addi x14,x14,-1
blt x12,x11,small_assign # if we find an element
smaller than x11 then put its value into x11
beq x0,x1,smallest
small_assign:
beq x0,x1,smallest
check:
bne x23,x0,next
do:
sub x20,x7,x19 # enter this loop if small_assign
hasn't been entered
add x16,x19,x20
next:
addi x19,x19,-1
la x15,array
mul x17,x16,x8
# x17 = x16*4
add x15,x15,x17 # x15 is array variable now we are
traversing to the place
lw x18,0(x15) # we are temporarily storing the value
that we are gonna swap
mul x21,x20,x8 # similarly x21 will have the address
of the correct position
la x20,array
add x20,x20,x21
```

```
lw x22,0(x20)
sw x22,0(x15)
sw x18,0(x20)
add x14,x0,x19
la x10,array
mul x9,x7,x8
add x10, x6,x9
lw x11,0(x10)
addi x23,x0,0 # resetting this check variable
bne x19,x0,smallest
```

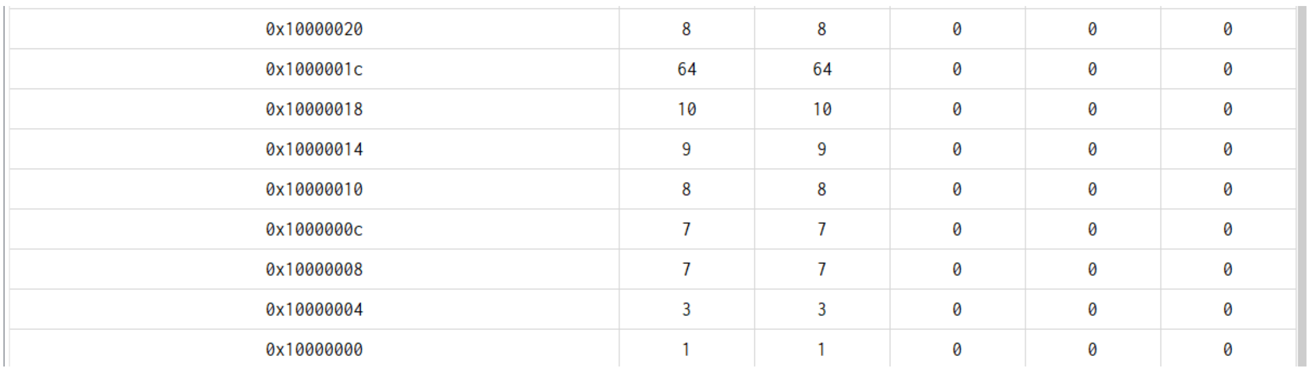
Results and Output

Before execution:



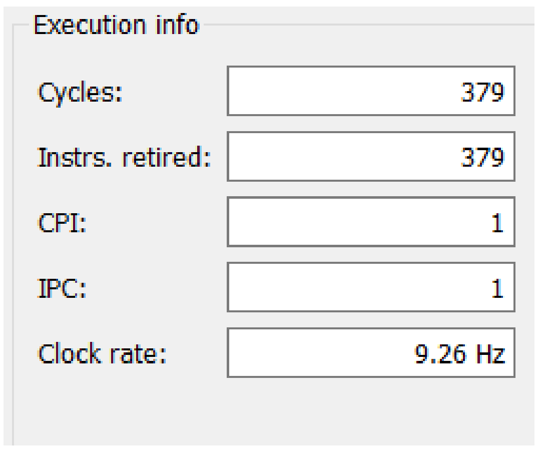
0x10000020	8	8	0	0	0
0x1000001c	9	9	0	0	0
0x10000018	10	10	0	0	0
0x10000014	7	7	0	0	0
0x10000010	64	64	0	0	0
0x1000000c	7	7	0	0	0
0x10000008	8	8	0	0	0
0x10000004	1	1	0	0	0
0x10000000	3	3	0	0	0

After execution:



0x10000020	8	8	0	0	0
0x1000001c	64	64	0	0	0
0x10000018	10	10	0	0	0
0x10000014	9	9	0	0	0
0x10000010	8	8	0	0	0
0x1000000c	7	7	0	0	0
0x10000008	7	7	0	0	0
0x10000004	3	3	0	0	0
0x10000000	1	1	0	0	0

Execution Info:



Execution info

Cycles:	379
Instrs. retired:	379
CPI:	1
IPC:	1
Clock rate:	9.26 Hz

Thus, we observe the execution and the array elements are sorted after execution.

So, APY told -
He needs the following documents before he hands over the “Financial report” [hard copy] to us:
a. Forwarding letter from QForest, through VK, to VC
b. Apology letter for delay in “Budget report”
c. Budget report + other expenses / billings