Implementation of Bucket sort using RISC-V assembly

RISC-V

UE21EC352A

Team Composition

Achyuth S.S	PES1UG21EC010
Archanaa A Chandaragi	PES1UG21EC056

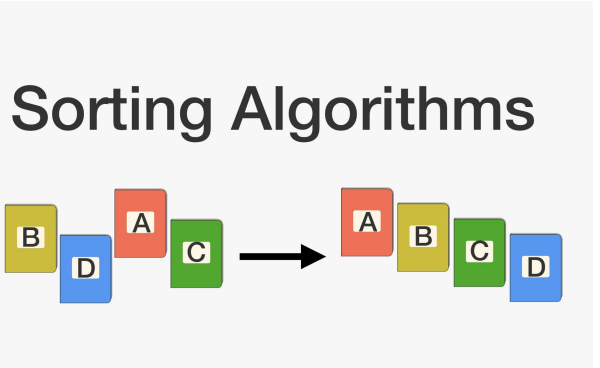
Guide : Prof. Vanamala H R

Contents

- Introduction
- Problem Statement
- Algorithm of Bucket sort
- Time and Space complexity analysis
- Hardware /Software used
- Code
- Result
- Applications
- References

Introduction

- Efficient sorting algorithms for large size data.
- In order to tackle the high time complexity and space complexity, we have Bucket sort algorithm that can help us.



Problem Statement

- The design of Sorting algorithms has always been carried out since many years.
- But, multiple sorting algorithms do not satisfy time and space complexities in some critical cases.
- In order to alleviate this issue, we have Bucket sort algorithm.

Algorithm - Bucket sort

- Bucket sort, or bin sort, is a sorting algorithm that works by distributing the elements of an array into a number of buckets.
- The computational complexity depends on the algorithm used to sort each bucket, the number of buckets to use, and whether the input is uniformly distributed.

Algorithm - Bucket sort

Consider the following array:

Create buckets with a range from 0 to 25. The buckets range are 0-5, 5-10, 10-15, 15-20, 20-25

Now, sort each bucket individually. The elements of each bucket can be sorted by using any of the stable sorting algorithms.

Algorithm - Bucket sort

Finally, gather the sorted elements from each bucket in order

Time complexity analysis

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

Space complexity analysis

Space complexity	$O(n * k)$

Software

- RIPES Simulator

Code

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

.text

start:
    la x6,array  #x6 contai ns the a
    lw x7,n  # x7 = 8
    #but we need to start from only
    addi x7,x7,-1 #x7=7
    li x8,4   # a common multiplier
    mul x9,x7,x8 # we will start che
    add x10, x6,x9 # we are traversi
    #initialising loop variable with
    addi x14,x7,0 #x14 => loop varia
    addi x19,x0,7 # another loop vri
    lw x11,0(x10) #loading the last
    li x23,0  #initialising a check
```

```
smallest:  # loop to find the smallest element in a certain array itera

    beq x14,x0,check  #loop variable
    addi x10,x10,-4 # to traverse through array backwards
    # we had loaded x11 with the last value in the array, now we consid
    #that to be the smallest and then traverse backwards by comparing i
    lw x12,0(x10)
    addi x14,x14,-1
    blt x12,x11,small_assign # if we find an element smaller than x11 t
    beq x0,x1,smallest
```

```
small_assign: #assigns x11 with smallest value
#note: this loop will be traversed only if the smallest elemnt is not
# to account for the case when the smallest element is at the end , we
    addi x11,x12,0  # put in the new small number value( it will be ir
    sub x20,x7,x19 # x20 will give the array position where we are sup
    add x16,x14,x20 # x16 value will give the array position that we a
    addi x23,x23,1 #x23 is the checking variable
    beq x0,x1,smallest
```

Code

```
check:
    bne x23,x0,next
do:
    sub x20,x7,x19 # enter this loop if smal
    add x16,x19,x20

next:

    addi x19,x19,-1 # x19 was initialised to
    la x15,array

    mul x17,x16,x8 # to get array location o
    # x17 = x16*4
    add x15,x15,x17 # x15 is array variable
    lw x18,0(x15) # we are temporarily stori


    mul x21,x20,x8    # similarly x21 will h
    la x20,array
    add x20,x20,x21
```

```
    lw x22,0(x20) # temporatily storing that value

    #swapping them both
    sw x22,0(x15)
    sw x18,0(x20)


#step2: Next time, x14 value decreases to 6, so the
# till the 2nd element of the array ie,"smallest" t
# 1st element is already in correct place


    add x14,x0,x19



# and the 5,4,3,2,1 => this would check these many

    la x10,array

    mul x9,x7,x8
    add x10, x6,x9 # once again traversing to end o
```

Code

```
lw x11,0(x10)  # storing that value in x11
addi x23,x0,0 # resetting this check vairable
bne x19,x0,smallest # going back to smallest
```

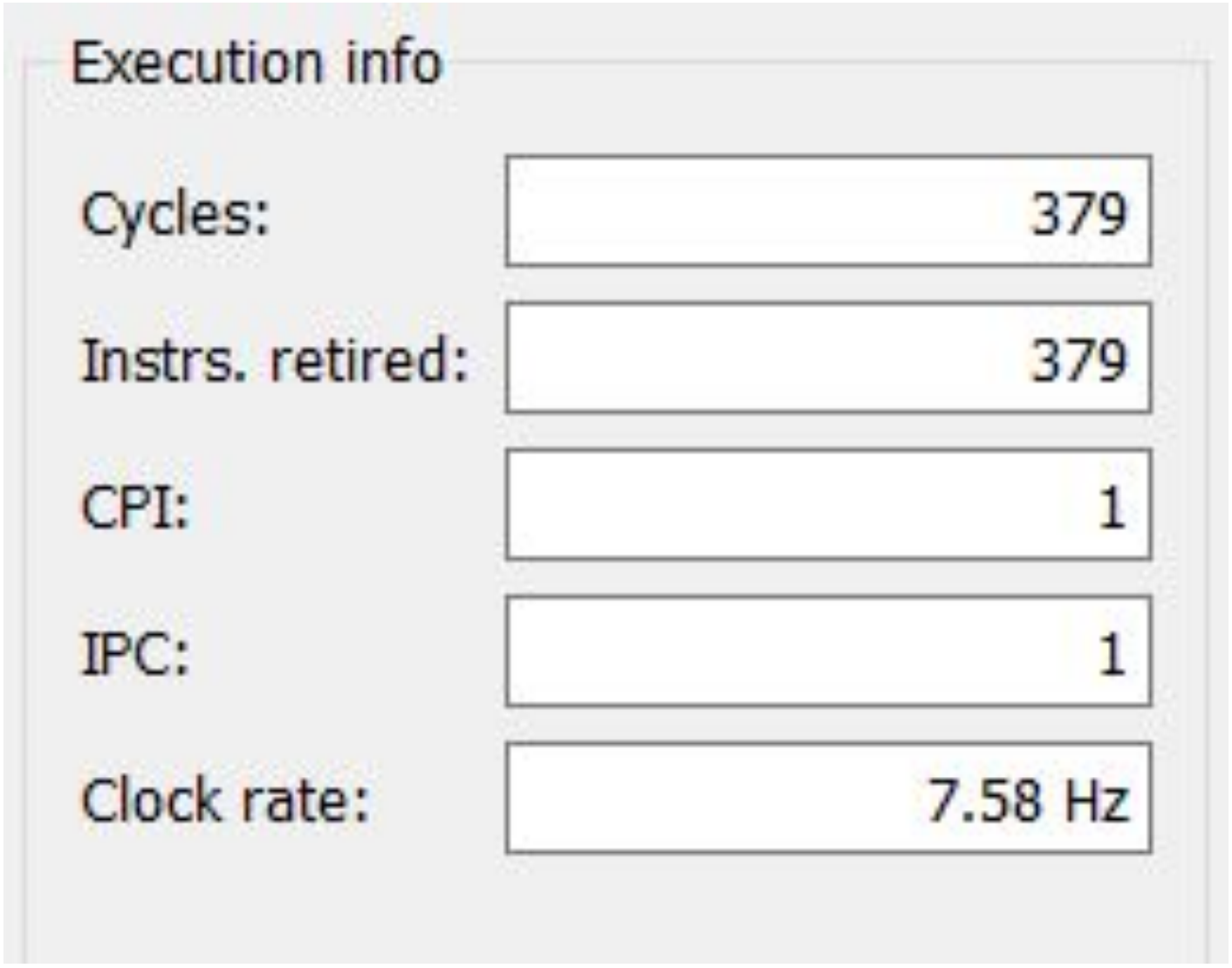
Results

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

Applications

- Sorts floating-point numbers efficiently within range.
- External sorting for large datasets in chunks.
- Efficient for sorting integers within limited range.
- Used in graphics for histogram sorting.

References

1.Z. Zhao and C. Min, "An Innovative Bucket Sorting Algorithm Based on Probability Distribution," 2009 WRI World Congress on Computer Science and Information Engineering, Los Angeles, CA, USA, 2009, pp. 846-850, doi: 10.1109/CSIE.2009.376.

2. RISC-V ISA specifications : https://riscv.org/technical/specifications/

THANK YOU