



iFYP

**Lab Manual # 03 – RISC-V Procedures and Instruction
Formats**

(Sorting using RISC-V Assembly language)

Release: 1.1

Date: 09-Sep-2024

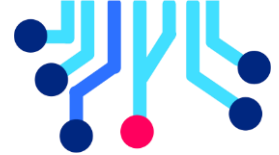


NUST Chip Design Centre (NCDC), Islamabad, Pakistan

Copyrights ©, NUST Chip Design Centre (NCDC). All Rights Reserved. This document is prepared by NCDC and is for intended recipients only. It is not allowed to copy, modify, distribute or share, in part or full, without the consent of NCDC officials.

Revision History

Revision Number	Revision Date	Revision By	Nature of Revision	Approved By
1.0	04/03/2024	Hira Sohail, Muhammad Bilal	Complete manual	Dr. Waqar
1.1	21/05/2024	Ali Aqdas	Additional Tasks	Dr. Waqar



Contents

Objectives	2
Tools	2
A Quick Summary	2
Lab Task 1: Find The Greatest Common Divisor	4
Lab Task 2: Bubble Sort	4
Recursion in RISC-V Assembly	4
Quicksort:	6
Lab Task 3: Quick Sort	8
Reference	8

Objectives

The objectives of this lab are to:

- Understand basic programs in assembly language covering following o Branches
o Loops o
Recursion
s
- Find the greatest common divisor in assembly language
- Perform sorting using assembly language

Tools

- Venus

A Quick Summary

Example 1 – If-ElseIf-Else Statement

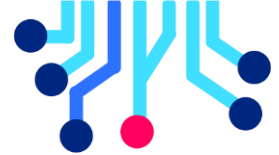
```

_start:

andi t0, t0, 0      # clear register t0
andi t1, t1, 0      # clear register t1
andi t2, t2, 0      # clear register t2
andi t3, t3, 0      # clear register t3
andi t4, t4, 0      # clear register t4
andi t5, t5, 0      # clear register t5
li t0, 2            # t0 = 2

li t3, -2           # t3 = -2
slt t1, t0, zero    # t1 = t0 < 0 ? 1 : 0
beq t1, zero, ElseIf # go to ElseIf if t1 = 0
j EndIf             # end If statement

```



```
ElseIf:
    sgt t4, t3, zero    # t4 = t3 > 0 ? 1 : 0
    beq t4, zero, Else  # go to Else if t4 = 0 j
EndIf                  # end Else statement

Else:
    seqz t5, t4, zero   # t5 = t4 == 0 ? 1 : 0

EndIf:
    j EndIf             # end If-ElseIf-Else statement
```

Example 2 – While Loop I

```
_start:
andi t0, t0, 0        # clear register t0
andi t1, t1, 0        # clear register t1
andi t2, t2, 0        # clear register t2
li t1, 100             # t1 = 100

loop:

    add t2, t2, t0     # t2 = t2 + t0
    addi t0, t0, 1     # ++t0
    blt t0, t1, loop   # iterate if t0 < t1

end:
    j end              # end of While loop
```

Example 3 – For Loop I

```
_start:

    andi t0, t0, 0     # clear register t0
    andi t1, t1, 0     # clear register t1

loop:

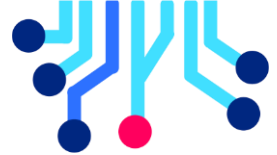
    andi t2, t2, 0     # clear t2 before starting the loop
    add t1, t1, t0     # t1 = t1 + t0

    addi t0, t0, 1     # ++t0

    slti t2, t0, 100   # t2 = t0 < 100 ? 1 : 0
    bne t2, zero, loop # go to loop if t2 != 0

end:

    j end              # end of For loop
```



Lab Task 1: Find The Greatest Common Divisor

Find the greatest common divisor of two numbers a and b , according to the Euclidean

Algorithm. The values of a and b should be statically defined variables. **(Hint: Remember the**

Concept of Different Assembler Directives)

Here is some additional information about the Euclidean Algorithm [The Euclidean Algorithm \(article\) | Khan Academy](#).

Lab Task 2: Bubble Sort

Implement the bubble sort algorithm. This algorithm sorts the components of a vector in ascending order by means of the following procedure.

1. Traverse the vector repeatedly until done.
2. Interchanging any pair of adjacent components if $V(i) > V(i+1)$.
3. The algorithm stops when every pair of consecutive components is in order.

Use 12-element arrays to test your program. Name the program **BubbleSort.S**.

Recursion in RISC-V Assembly

A procedure for calculating factorial.

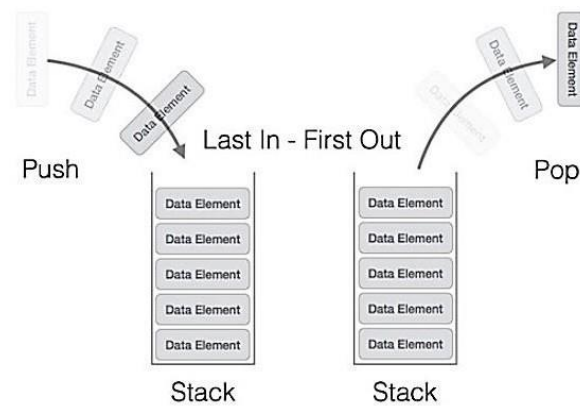
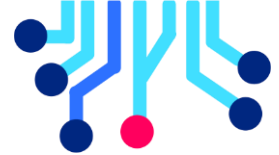
```
int fact (int n) {
    if (n < 1) return 1;
    else return (n * fact (n-1));
}
```

- A recursive procedure (one that calls itself!)

fact (0) = 1 fact (1) = 1 * 1 = 1
 fact (2) = 2 * 1 * 1 = 2 fact (3) = 3
 * 2 * 1 * 1 = 6 fact (4) = 4 * 3 * 2 *
 1 * 1 = 24 . . .

- Assume n is passed in $a0$; result returned in ra

Stack



Compiling a Recursive Program (cont.)

```
fact:
    addi sp, sp, -8 #   adjust the stack pointer

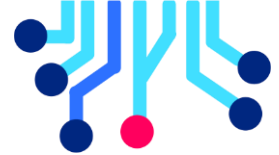
    sw ra, 4(sp)    # save the return address
    sw a0, 0(sp)    # save the argument n
    slti t0, a0, 1  # test for n < 1

    beq t0, zero, L1 # if n >=1, go to L1
    addi t1, zero, 1 # else return 1 in t1
    addi sp, sp, 8   # adjust stack pointer
    jr ra           #   return to caller
L1:
    addi a0, a0, -1  # n >=1, so decrease n
    jal fact         # call fact with (n-1) # this is where fact returns
bk_f:
    lw a0, 0(sp)     # restore argument n
    lw ra, 4(sp)     # restore return address

    addi sp, sp, 8 #   adjust stack pointer
    mul t1, a0, t1  #   t1 = n * fact(n-1)
    jr ra          #   return to caller
```

Example II

```
.globl _start
```



```
.text      #      recursive implementation of factorial
fact:      #      arg: n in a0, returns n! in a1
    addi sp, sp, -8 #      reserve our stack area

    sw ra, 0(sp)    #      save the return address

    li t0, 2        #      t0 = 2

    blt a0, t0, ret_one # go to ret_one if a0 < t0
    sw a0, 4(sp)      #      save our n

    addi a0, a0, -
1 jal fact          #      call fact (n-1), a1 <- fact(n-1)
    lw t0, 4(sp)      #      t0 <- n

    mul a1, t0, a1     #      a1 <- n * fact(n-1)

    j done
ret_one:
    li a1, 1

done:
    lw ra, 0(sp) # restore return address from stack
    addi sp, sp, 8 # free our stack frame
    jr ra # and return

_start:
    li a0, 5 # compute 5!
    jal fact # call 'fact'
    li a0, 1 # print it
    ecall
```

Quicksort

Quicksort is a divide and conquer algorithm. Quicksort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quicksort can then recursively sort the sub-arrays.

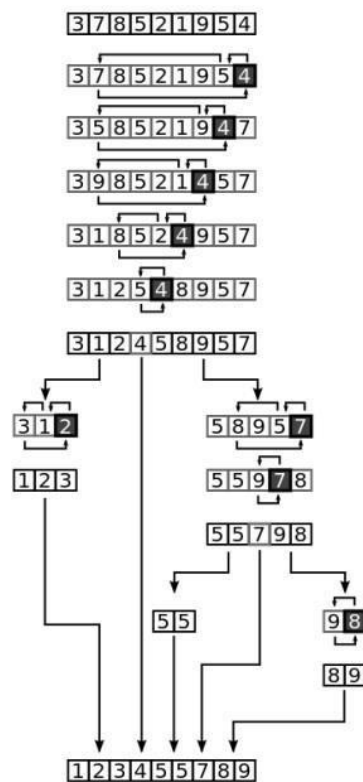
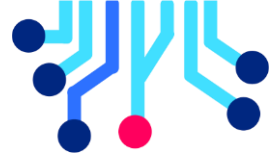


Fig: Quick sort algorithm

Quicksort:

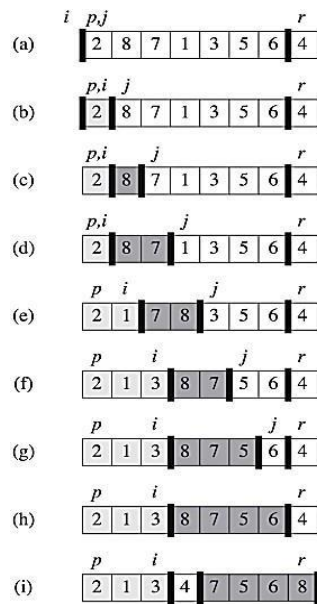
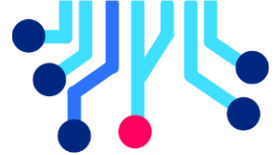
Pick an element, called a pivot, from the array.

- Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way).

```

1:  function PARTITION(A, lo, hi)
2:      pivot ← A[hi]
3:      i ← lo-1;
4:      for j = lo; j ≤ hi-1; j ← j+1 do
5:          if A[j] ≤ pivot then
6:              i ← i+1;
7:              swap A[i] with A[j];
8:          end if
9:      end for
10:     swap A[i+1] with A[hi];
11:     return i+1;
12: end function

```

In this example, $p = lo$ and $r = hi$.

Recursively apply the array partition to the sub-array of elements with smaller values and separately to elements with greater values.

```

1:  function QUICKSORT(A, lo, hi)
2:    if lo < hi then
3:      p ← partition(A, lo, hi);
4:      quicksort(A, lo, p - 1);
5:      quicksort(A, p + 1, hi);
6:    end if
7:  end function
  
```

Lab Task 3: Quick Sort

Implement Quicksort w.r.t. the following array in ascending order:

Sort the array for this assignment.

-1 22 8 35 5 4 11 2 1 78

Submit the source code along with the conclusion showing your understanding.

Reference

<https://www.cse.cuhk.edu.hk/~byu/CENG3420/2022Spring/slides/lab1-2.pdf>

[The Euclidean Algorithm \(article\) | Khan Academy](#)