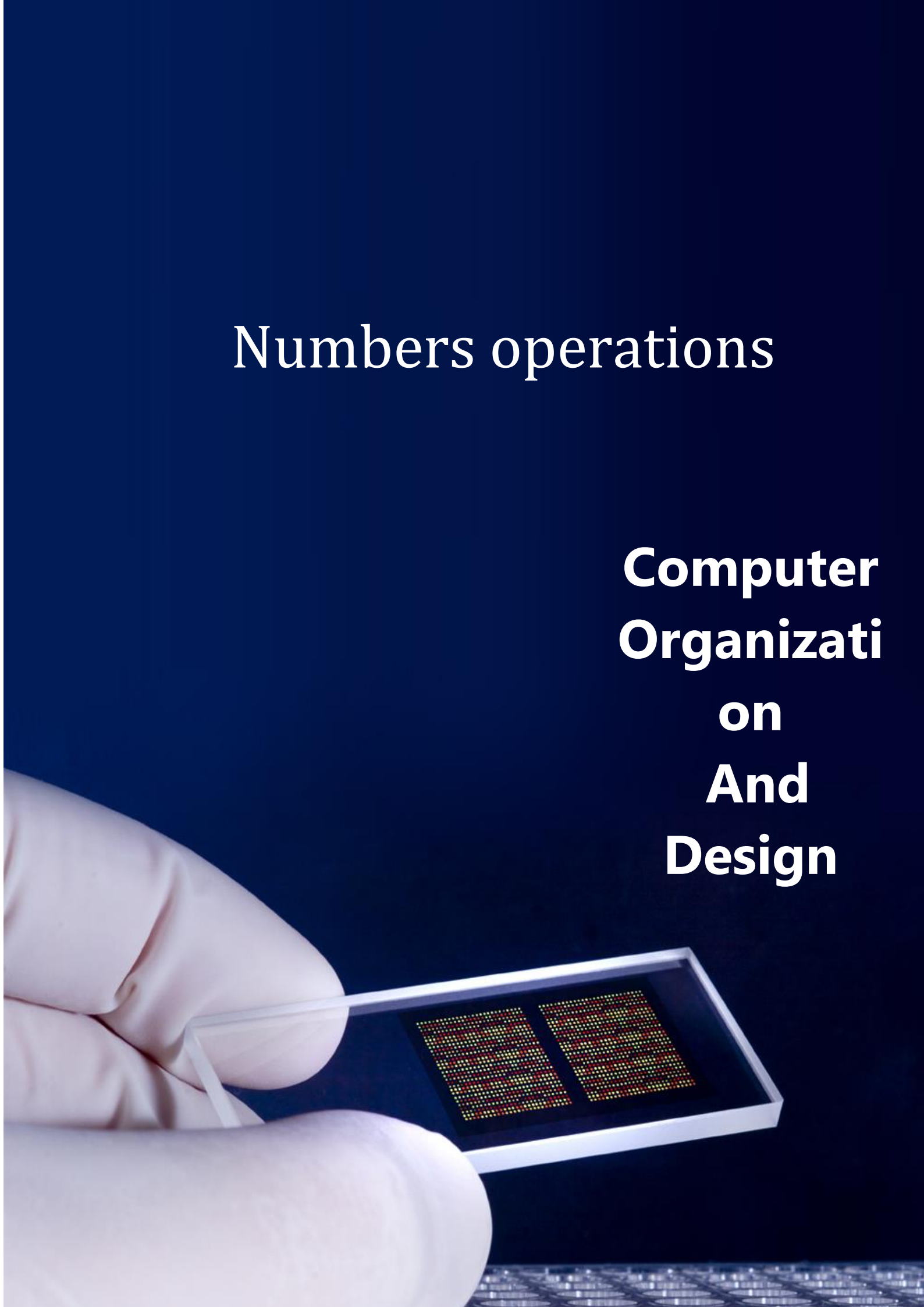
Numbers operations

Computer Organization And Design



Faculty of Engineering at Shoubra
Electrical Engineering department-CCEP

CCE 307 – COURSE PROJECT (TERM 232)

Course Project cover page

Student Name	Edu Email	Student ID	Marks		
			Report & Slides (30)	Implementation (50)	Presentation (20)
Mohamed Khaled	Mohammed20089@feng.bu.edu.eg	221902935			
Omar Nour Eldin	omar20687@feng.bu.edu.eg	221902921			
Mohamed Hatem	Mohmed20080@feng.bu.edu.eg	221902934			
Mohamed Ahmed	muhammad20779@feng.bu.edu.eg	221902931			
Kirlos Sameh	Kyrollos20740@feng.bu.edu.eg	221902929			

Date handed in: / 12 / 2023

section	page
Data Section	4
Main	6
Min, Max, Sort, Average & Sum	8
nCr	18
Decimal to Binary	25
Facorial	27
Fibonacci	30
Quit	34
Task assignment	35

1) Data Section:

Description: In the data section we added the string messages that will be prompted to the user as well as the global variables and constants needed in the program.

Assembly Code:

```
.data
#variables of Min,Max,sum,average,sort
	array:		.space	40	#(up to 10 items) * (4 bytes)
	arraySize:	.word	0	#array length

	#Answers
	theAverage:	.word	0
	theSum:		.word	0
	theMax:		.word	0
	theMin:		.word	0

	#Messages
	newline:	.asciiz	"\n"
	point:		.asciiz	"."
	spacing:	.asciiz	","
	colon:		.asciiz	" :"

	Message:	.asciiz	"\nEnter size of the array "
	promptMessage:	.asciiz	"\nEnter numbers "
	Input:		.asciiz	"\nThe numbers in descending order are "
	Average:	.asciiz	"\nThe Average is  "
	Sum:		.asciiz	"\nThe Sum is  "
	Min:		.asciiz	"\nThe Min is  "
	Max:		.asciiz	"\nThe Max is  "



 mess1:	.asciiz	"\nplease Enter either\nNumber (1): for min,max,sum of digits of an integer \nNumber (2): for calculating nCr \nNumber (3): for caluclating the factorial of
#main
valid1:	.asciiz "\n**********Please enter a valid input.**********\n"
line:	.asciiz "\n********************************************************************************\n"
```

```
#nCr
 promptMessage1: .asciiz "\nEnter n: "        #.asciiz for string
 promptMessage2: .asciiz "Enter r: "
 errorMsg:       .asciiz "out of range"
 errorMsg2:      .asciiz "invaild input"
 resultMessage:  .asciiz "C"
 resultmessage2: .asciiz " = "
 n:              .word   0                    #.word for integer
 r:              .word   0
 theAnswer:      .word   0
 max1:           .word   10000000000  #must  be less than 4294967295(2^32 - 1)

#-------------------------------------------------------------------------------------------------------------------

 #Fibonacci sequence
 prompt1: .asciiz "Enter the sequence index/n "
 prompt2: .asciiz "The Fibonacci value is: "
 nl:         .asciiz     "\n"


#-------------------------------------------------------------------------------------------------------------------


 # Factorial
 popMessage: .asciiz "Enter a anumber to find its factorial:  "
 resultMassege: .ascii "\n The factorial of number is "
 theNumber: .word 0

#-------------------------------------------------------------------------------------------------------------------

 #Binary
 mess7: .asciiz "\nenter a decimal number\n"

#-------------------------------------------------------------------------------------------------------------------
```

2) Main Function:

Description: The main function is used to display a menu to the user to choose the required function to run or to choose to close the program

Assembly Code:

```
.text
  main:

    li $v0, 4            # Load immediate value 4 into register $v0 (system call code for printing a string
  la $a0, line          # Load address of the string "line" into register $a0
  syscall               # Execute the system call to print the string

  li $v0, 4             # Load immediate value 4 into register $v0 (system call code for printing a string)
  la $a0, mess1         # Load address of the string "mess1" into register $a0
  syscall               # Execute the system call to print the string

  # Take a number [1,6] from the user.
  li $v0, 5             # Load immediate value 5 into register $v0 (system call code for reading an integer)
  syscall               # Execute the system call to read an integer

  beq $v0, 1, func3     # Branch to "func3" if the value in register $v0 is equal to 1
  beq $v0, 2, func1     # Branch to "func1" if the value in register $v0 is equal to 2
  beq $v0, 3, func4     # Branch to "func4" if the value in register $v0 is equal to 3
  beq $v0, 4, func2     # Branch to "func2" if the value in register $v0 is equal to 4
  beq $v0, 5, func5     # Branch to "func5" if the value in register $v0 is equal to 5
  beq $v0, 6, quit      # Branch to "quit" if the value in register $v0 is equal to 6

invalid:               # Label for the invalid option
  li $v0, 4             # Load immediate value 4 into register $v0 (system call code for printing a string)
  la $a0, valid1        # Load address of the string "valid1" into register $a0
  syscall               # Execute the system call to print the string
  j main                # Unconditional jump to the "main" label (to start the main part of the program)
```

Output:

```
please Enter either
Number (1): for min,max,sum,average,sort of digits of an integer
Number (2): for calculating nCr
Number (3): for caluclating the factorial of intger number
Number (4): for calcuating Fibonacci sequence
Number (5): for decimal to binary conversion
Number (6): to quit
```

Code Description:

li $v0,4 \ we give it no. 4 to print a string
la $a0,line \ print a separator line(*******)
syscall \ to execute the code
la $a0, mess1 put the message (The menu message) you want to print in $a0
li $v0,5 \ give it 5 to take an integer from user (the choice from menu)
****

beq $v0,1, func1
beq $v0, 2, func2
beq $v0, 3, func3
beq $v0, 4, func4
beq $v0, 5, func5
beq $v0, 6, quit \Take the choice of the user and compare it to numbers from (1:6) then if it is equal with any number will enter its function
****

la $a0, valid1 \print to the user that it's a valid number because the number taken from the user doesn't match any of the functions' numbers.
j main \ after finishing it will repeat the main to choose from the menu again

Min, Max, Sum of Digits function:

Description:

The function takes a number of integers as an input from the user and prints the maximum digit, minimum digit, descending order, sum of integers and the average of the numbers.

Assembly Code:

```
#Min,Max,sum,average,sort
func3:#print prompt message
            li $v0, 4
            la $a0, Message
            syscall
            li $v0, 4
            la $a0, newline
            syscall

            #read int from the keyboard
            li $v0, 5
            syscall
            ########################
            sw $v0, arraySize # store given int in the $v0
            mul $v1, $v0, 4
            ########################
            add  $t0, $t0, $v1  #array size const
            addi $t4, $t4, 90   #loop counter
            addi $t9, $t9, 1    #input counter
            add  $s5, $s5, $v0  #division  s5 = no of elements

            #print prompt message
            li $v0, 4
            la $a0, promptMessage
            syscall
            li $v0, 4
            la $a0, newline
            syscall

            #read int from the keyboard
            input:
                    beq $t1, $t0, continue

                    move $a0, $t9 # display num from 1 to 3 to input number
```

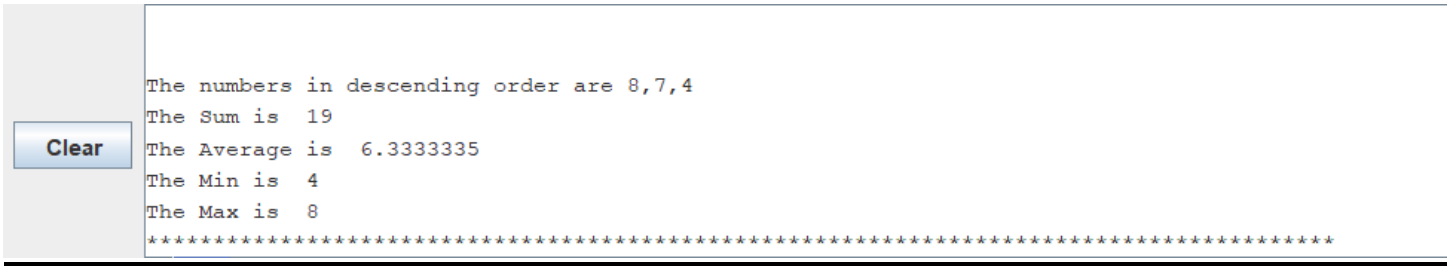
```
                li $v0, 1
                syscall
                li $v0, 4
                la $a0, colon
                syscall

                #read int from the keyboard
                li $v0, 5
                syscall
                move $t2, $v0

                sw   $t2, array($t1)
                addi $t1, $t1, 4      # next position in the array
                addi $t9, $t9, 1      # update input counter

                j input #loop 3 times

continue:
        #reinitialize register
        move $t1, $zero #for array[x]

        move $t2, $zero
        addi $t2, $t2, 4 #for array[x+1]

        move $s0, $zero
        addi $s0, $s0, 1 #condition check


sorting:
        beq $t3, $t4, calculation  #finish loop if($t3 == $t4) t3 loop counter $t4 = go to ca
        beq $t2, $t0, continue     #reinitialize array offset for looping

        lw $t5, array($t1) # $t5 = array[x]
        lw $t6, array($t2) # $t6 = array[x+1]
```

```
        addi $t3, $t3, 1

        slt $t7, $t5, $t6 # if ($t5 < $t6) $t7 = 1
        beq $t7, $s0, rearrange

        #increament
        addi $t1, $t1, 4
        addi $t2, $t2, 4

        j sorting

rearrange:
        sw $t5, array($t2) # $t5 = array[x+1]
        sw $t6, array($t1) # $t6 = array[x]

        addi $t1, $t1, 4
        addi $t2, $t2, 4

        j sorting

calculation:
        #reinitialize register
        move $t1, $zero  #array element
        move $t2, $zero  #temp holder
        move $t3, $zero  #first array element
        move $t4, $zero
        addi $v1,$v1,-4 #mario
        add  $t4, $t4, $v1 #last array element

        move $t5, $zero   # min
        move $t6, $zero   # max
        move $t7, $zero
```

```
total:
        beq  $t1, $t0, average  #if (t1 == t2 ) go to avg after gettting the sum    t1 offset    t2 = 
        lw   $t2, array($t1)
        addi $t1, $t1, 4        #update array increament
        add  $s1, $s1, $t2      #t2 new array element s2

        j total


average:
        #div  $s1, $s5 #$s1 / $s5 = array size     divide summation by arraysize
        #mfhi $s2      #s2 = remainder
        #mflo $s3      #s3 = quotient
        mtc1 $s1, $f0          #move to coprocessor
        cvt.s.w $f0, $f0       #convert to float
        #addi $t5, $zero, 3
        mtc1 $s5, $f2          #move to coprocessor
        cvt.s.w $f2, $f2       #convert to float
        div.s $f12, $f0, $f2

#PRINTING
        output:
        li $v0, 4
        la $a0, newline
        syscall
        # 0 4 8 12... after sorting
        lw $t5, array($t3) # $t5 = array[0] max
        lw $t6, array($t4) # $t6 = array[8] min

        li $v0, 4
        la $a0, Input
        syscall
```

```
    #print elements of an array in descending order
    element:
            beq  $t7, $t0, next
            lw   $t8, array($t7)
            addi $t7, $t7, 4

            #print the value
            move $a0, $t8
            li   $v0, 1
            syscall

            beq $t7, $t0, element # if t0 != 12

            #print a comma
            li $v0, 4
            la $a0, spacing
            syscall
            j element
    next:
    #print sum

            #display result message for user
            li $v0, 4
            la $a0, Sum
            syscall

            move $a0, $s1 # display sum num
            li   $v0, 1
            syscall




   #print max
          #display result message for user
          li $v0, 4
          la $a0, Max
          syscall

          # display max num
          move $a0, $t5
          li   $v0, 1
          syscall
j main
```

Output:



Code description:

```
func3:              ( Function label)

   # Print prompt message
   li $v0, 4          (Load immediate 4 into register $v0 (syscall code for printing a string))
   la $a0, Message    (Load the address of the string message into register $a0)
   syscall            (Perform a system call to print the message)
   li $v0, 4          (Load immediate 4 into register $v0 again (syscall code for printing a string))
   la $a0, newline    (Load the address of a newline character into register $a0)
   syscall            (Perform a system call to print the newline)

   (Read an integer from the keyboard)
   li $v0, 5          (Load immediate 5 into register $v0 (syscall code for reading an integer))
   syscall            (Perform a system call to read an integer)
   ########################

   (Store the inputted integer in memory)
   sw $v0, arraySize   (Store the value in register $v0 (inputted integer) at the memory address specified by the label 'arraySize')

   # Multiply the inputted integer by 4
   mul $v1, $v0, 4     (Multiply the value in register $v0 (inputted integer) by 4 and store the result in register $v1)

   ########################

  (Update counters and variables)
   add $t0, $t0, $v1   (Add the values in registers $t0 and $v1 and store the result in $t0)
                ((presumably, $t0 represents the array size constant))

   addi $t4, $t4, 90  (Add the immediate value 90 to the value in register $t4)
                ((presumably, $t4 is a loop counter))

   addi $t9, $t9, 1   ( Add the immediate value 1 to the value in register $t9)
                ( (presumably, $t9 is an input counter))
```

```
   add $s5, $s5, $v0  ( Add the values in registers $s5 and $v0 and store the result in $s5)
                ( (presumably, $s5 is used to keep track of the number of elements))
                ( s5 = no of elements)

   # Print prompt message
   li $v0, 4          (Load immediate 4 into register $v0 (syscall code for printing a string))
   la $a0, promptMessage   # Load the address of the prompt message into register $a0)
   syscall            ( Perform a system call to print the prompt message)
   li $v0, 4          ( Load immediate 4 into register $v0 again (syscall code for printing a string))
   la $a0, newline     ( Load the address of a newline character into register $a0)
   syscall            ( Perform a system call to print the newline)
input:
   beq $t1, $t0, continue    ( Branch to the label 'continue' if the values in registers $t1 and $t0 are equal)

   move $a0, $t9           ( Move the value in register $t9 to register $a0)
                   ( (presumably, $t9 holds a number from 1 to 3 to display as input number))

   li $v0, 1             ( Load immediate 1 into register $v0 (syscall code for printing an integer)
   syscall               ( Perform a system call to print the value in register $a0 as an integer)

   li $v0, 4             ( Load immediate 4 into register $v0 (syscall code for printing a string))
   la $a0, colon          ( Load the address of the string "colon" into register $a0)
   syscall               ( Perform a system call to print the string)

   (Read an integer from the keyboard)
   li $v0, 5             ( Load immediate 5 into register $v0 (syscall code for reading an integer))
   syscall               ( Perform a system call to read an integer)
   move $t2, $v0          ( Move the value read from the keyboard to register $t2)

   sw $t2, array($t1)     ( Store the value in register $t2 at the memory address specified by the expression 'array($t1)')
                   ( (presumably, it stores the inputted value in an array at the corresponding position))

   addi $t1, $t1, 4       ( Add the immediate value 4 to the value in register $t1)
                   ( (presumably, it increments the position in the array for the next input))

   addi $t9, $t9, 1       ( Add the immediate value 1 to the value in register $t9)
                   ( (presumably, it increments the input counter))

   j input               ( Unconditionally jump to the label 'input')


continue:
   # Reinitialize registers
   move $t1, $zero        ( Move the value in register $zero to register $t1)
                   ((presumably, resetting $t1 to 0 for array[x]))
   move $t2, $zero         ((Move the value in register $zero to register $t2)
                   ((presumably, resetting $t2 to 0 for array[x+1]))
   addi $t2, $t2, 4        (Add the immediate value 4 to the value in register $t2)
                   ((presumably, setting $t2 to 4 for array[x+1]))

   move $s0, $zero         (Move the value in register $zero to register $s0)
                   ((presumably, resetting $s0 to 0 for condition check))
   addi $s0, $s0, 1        (Add the immediate value 1 to the value in register $s0)
                   ((presumably, setting $s0 to 1 for the condition check))

sorting:
```

```
   beq $t3, $t4, calculation    (Branch to the label 'calculation' if the values in registers $t3 and $t4 are equal)
                        ((presumably, finishing the loop if $t3 == $t4, where $t3 is a loop counter and $t4 is a limit))

   beq $t2, $t0, continue      (Branch to the label 'continue' if the values in registers $t2 and $t0 are equal)
                        ((presumably, reinitializing the array offset for looping))

   lw $t5, array($t1)          (Load the word at the memory address specified by the expression 'array($t1)' into register $t5)
                        ((presumably, $t5 is assigned the value of array[x]))

   lw $t6, array($t2)          (Load the word at the memory address specified by the expression 'array($t2)' into register $t6)
                        ((presumably, $t6 is assigned the value of array[x+1]))

   addi $t3, $t3, 1            (Add the immediate value 1 to the value in register $t3)
                        ((presumably, incrementing the loop counter $t3))

   slt $t7, $t5, $t6           (Set register $t7 to 1 if the value in $t5 is less than the value in $t6, otherwise 0)
                        ((presumably, comparing $t5 and $t6))

   beq $t7, $s0, rearrange     (Branch to the label 'rearrange' if the values in registers $t7 and $s0 are equal)
                        ((presumably, if $t5 < $t6, $t7 will be 1 and the branch will be taken))

   addi $t1, $t1, 4            (Add the immediate value 4 to the value in register $t1)
                        ((presumably, incrementing $t1 for the next element in the array))

   addi $t2, $t2, 4            (Add the immediate value 4 to the value in register $t2)
                        ((presumably, incrementing $t2 for the next element in the array))

   j sorting                   (Unconditionally jump to the label 'sorting')
total:
   beq $t1, $t0, average   ( Branch to the label 'average' if the values in registers $t1 and $t0 are equal)
                        ( (presumably, if $t1 == $t0, go to average after getting the sum))
   lw $t2, array($t1)      ( Load the word at the memory address specified by the expression 'array($t1)' into register $t2)
                        ( (presumably, $t2 is assigned the value of array[offset]))
   addi $t1, $t1, 4        ( Add the immediate value 4 to the value in register $t1)
                        ( (presumably, incrementing $t1 for the next element in the array))
   add $s1, $s1, $t2       ( Add the value in register $t2 to the value in register $s1 and store the result in $s1)
                        ( (presumably, updating the sum by adding the new array element))

   j total                 ( Unconditionally jump to the label 'total')

average:
   mtc1 $s1, $f0           ( Move the value in register $s1 to coprocessor register $f0)
                        ( (presumably, moving the sum to a floating-point register))
   cvt.s.w $f0, $f0        ( Convert the value in coprocessor register $f0 from integer to floating-point)
                        ( (presumably, converting the sum to a floating-point representation))
   mtc1 $s5, $f2           ( Move the value in register $s5 to coprocessor register $f2)
                        ( (presumably, moving the array size to a floating-point register))
   cvt.s.w $f2, $f2        ( Convert the value in coprocessor register $f2 from integer to floating-point)
                        ( (presumably, converting the array size to a floating-point representation))
   div.s $f12, $f0, $f2    ( Divide the value in coprocessor register $f0 by the value in coprocessor register $f2)
                        ( and store the quotient in coprocessor register $f12)
                        ( (presumably, calculating the average by dividing the sum by the array size))
output:
   li $v0, 4               ( Load immediate value 4 into register $v0 (print_string syscall))
   la $a0, newline         ( Load the address of the string "newline" into register $a0)
   syscall                 ( Print the newline character)
```

```
   lw $t5, array($t3)   ( Load the word at the memory address specified by the expression 'array($t3)' into register $t5)
                ( (presumably, $t5 is assigned the maximum value from the array))
   lw $t6, array($t4)   ( Load the word at the memory address specified by the expression 'array($t4)' into register $t6)
                ( (presumably, $t6 is assigned the minimum value from the array))

   li $v0, 4          ( Load immediate value 4 into register $v0 (print_string syscall))
   la $a0, Input       ( Load the address of the string "Input" into register $a0)
   syscall            ( Print the string "Input")

element:
   beq $t7, $t0, next   ( Branch to the label 'next' if the values in registers $t7 and $t0 are equal)
                ( (presumably, if $t7 == $t0, exit the loop and go to the next section))

   lw $t8, array($t7)   ( Load the word at the memory address specified by the expression 'array($t7)' into register $t8)
                ( (presumably, $t8 is assigned the current array element to be printed))

   addi $t7, $t7, 4     ( Add the immediate value 4 to the value in register $t7)
                ( (presumably, incrementing $t7 to point to the next element in the array))

   move $a0, $t8       ( Move the value in register $t8 to register $a0)
                ( (presumably, setting $a0 to the current array element to be printed))

   li $v0, 1          ( Load immediate value 1 into register $v0 (print_int syscall))
   syscall            ( Print the value in register $a0 as an integer)

   beq $t7, $t0, element   ( Branch to the label 'element' if the values in registers $t7 and $t0 are equal)
                  ( (presumably, if $t7 != $t0, continue printing the remaining elements))

   li $v0, 4          ( Load immediate value 4 into register $v0 (print_string syscall))
   la $a0, spacing     ( Load the address of the string "spacing" into register $a0)
   syscall            ( Print a comma and a space)

   j element          ( Unconditionally jump back to the label 'element')
next:
   # Print sum
   li $v0, 4          ( Load immediate value 4 into register $v0 (print_string syscall))
   la $a0, Sum          ( Load the address of the string "Sum" into register $a0)
   syscall            ( Print the string "Sum")

   move $a0, $s1         ( Move the value in register $s1 (the sum) to register $a0)
   li $v0, 1          ( Load immediate value 1 into register $v0 (print_int syscall))
   syscall            ( Print the value in register $a0 as an integer)

   # Print average
   li $v0, 4          ( Load immediate value 4 into register $v0 (print_string syscall))
   la $a0, Average       ( Load the address of the string "Average" into register $a0)
   syscall            ( Print the string "Average")

   li $v0, 2          ( Load immediate value 2 into register $v0 (print_float syscall))
   syscall            ( Print the floating-point value in coprocessor register $f0 as a decimal number)

   # Print minimum
   li $v0, 4          ( Load immediate value 4 into register $v0 (print_string syscall))
   la $a0, Min          ( Load the address of the string "Min" into register $a0)
   syscall            ( Print the string "Min")

   move $a0, $t6         ( Move the value in register $t6 (the minimum) to register $a0)
```

```
    li $v0, 1           ( Load immediate value 1 into register $v0 (print_int syscall))
    syscall             ( Print the value in register $a0 as an integer)

    # Print maximum
    li $v0, 4           ( Load immediate value 4 into register $v0 (print_string syscall))
    la $a0, Max         ( Load the address of the string "Max" into register $a0)
    syscall             ( Print the string "Max")

    move $a0, $t5       ( Move the value in register $t5 (the maximum) to register $a0)
    li $v0, 1           ( Load immediate value 1 into register $v0 (print_int syscall))
    syscall             ( Print the value in register $a0 as an integer)

j main                  ( Unconditionally jump back to the label 'main')
```

nCr function:

Description:

The function takes two integers as an input from the user and the first integer must be greater than the second one the it prints the nCr of these two numbers where :

nCr = npr/ rpr (n!/r!(n-r)!)

Assembly Code:

```
func1:
	# call input function
		jal input
	# call the nCr function
		jal nCr
	# call the Display function
		jal Display
	# end the program
		jal endProgram
#-------------------------------------------------------------------------------------------------------------------------------
	input:
		addi $sp, $sp, -4		# allocate one more free space
		sw   $ra, 0($sp)		# store the return address in a free location
	#display the prompt message
		li $v0, 4			# 4 to tell the system to print a string text that located in $a0
		la $a0, promptMessage1		#load the message from the golbal vriable "prommptMessage1" from the ram into $a0 in order to
		syscall				# execute the system call specified by the value in $v0
	# read the number from the user
		li $v0,5			# 5 to tell the system to read an integer value from the user that would be saved in $v0
		syscall				# execute the system call specified by the value in $v0
		sw $v0, n			# store the value from the user in the gobal variable n
	#display the prompt message
		li $v0, 4			# 4 to tell the system to print a string text that located in $a0
		la $a0, promptMessage2		# load the message from the golbal vriable "prommptMessage2" from the ram into $a0 in order to
		syscall				# execute the system call specified by the value in $v0
	# read the number from the user
		li $v0,5			# 5 to tell the system to read an integer value from the user that would be saved in $v0
		syscall				# execute the system call specified by the value in $v0
		sw $v0, r			#store the value from the user in the gobal variable r
	 # check for invalid input
		jal invalid1
		lw   $ra, 0($sp)		# get back the value $ra to return back to  the caller function
		addi $sp, $sp, 4		# free the  space locations
	#return to the caller function
		jr $ra
```

```
invalid1:
   lw $a0,n                  # load n from the ram into the argument regiser $a0 to pass it to the function
   lw $a1,r                  # load r from the ram into the argument regiser $a1 to pass it to the function
#check for invaild input
   slt  $t0,$a0,$a1          # check if  n < r               --> $t0 =1
   slt  $t1,$a0,$zero        # check if  n is a negative value --> $t1 =1
   slt  $t2,$a1,$zero        # check if  r is a negative value --> $t2 =1
   or   $t3,$t0,$t1          # check if  n < r or n is a negative value  --> $t3 =1
   or   $t1,$t2,$t3          # check if  (n < r or n is a negative vaule) or (r is a negative vaule) --> $t1 =1
   beq  $t1,$zero,pass       # check for invaild input ($t1 = 1) or to pass and continue the function
   li   $v0,4                # 4 to print a string text that is located in $ a0
   la   $a0,errorMsg2        # to print the rest of the result message
   syscall
   j func1                   # execute the system call specified by the value in $v0
    pass:                    # end of the program
    jr $ra                          # return to the caller function

#-------------------------------------------------------------------------------------------------------------------------------

 overflow:
            lw      $t3,max1                    # load the maximum value in the $t3
            slt     $t1,$v0,$zero         # check if  $v0 is a negative value
            slt     $t2,$t3,$v0           # check if $v0 is greater than the maximum value
            or      $t3,$t1,$t2           # check of $v0 is a negative value or greater than the maximum value
            beq     $t3,$zero,valid       # if $v0 is postive value and less than the maximum value then would jump to valid lab
            li      $v0,4                 # 4 to print a string text that is located in $ a0
            la      $a0,errorMsg          # to print the string located in the golbal variable $a0
            syscall                       # execute the system call specified by the value in $v0
            j       func1          # return to the calelr fucntion
            valid:
            jr $ra                        # return to the caller function
```

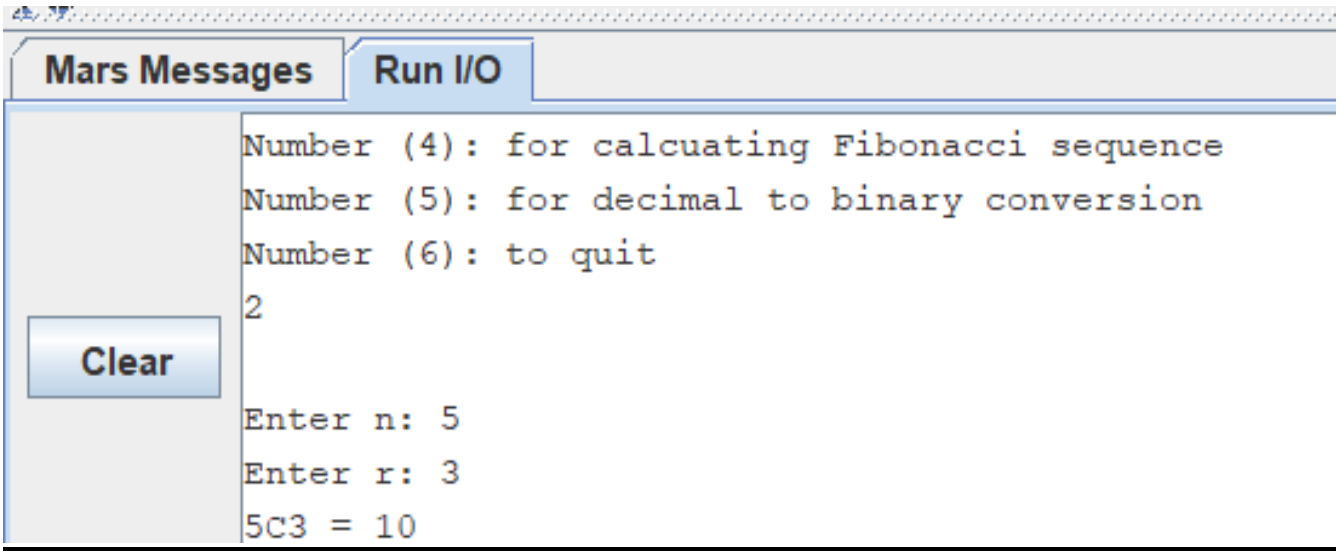
```
nCr:                        # n = $a0   r = $a1
        addi $sp, $sp, -12      # allocate three more free space
        sw   $ra, 0($sp)        # store the return address in a free location
        sw   $s0, 4($sp)        # store the value of  $s0 in a free location
        sw   $s1, 8($sp)        # store the value of  $s1 in a free location

        lw $a0,n                # load n from the ram into the argument regiser $a0 to pass it to the function
        lw $a1,r                # load r from the ram into the argument regiser $a1 to pass it to the function

        #get nPr
        jal check               # to check if r > n/2  if so then it would be set to n-r to avoid over flow in bigger r
        jal nPr                 # call the function
        add $s0,$v0,$zero       # save the rsult in $s0

        # get rPr
        move $a0,$a1            # load n in argument register $a0 to pass it to function and set it to the value of r
        jal nPr                 # call the function
        add $s1,$v0,$zero       # save the resutl in $s1

        #get the answer [nCr = (nPr)/(rPr)]
        div $v0,$s0,$s1

        sw $v0,theAnswer        # store the value in theAnswer

        # ending the function
        lw   $ra, 0($sp)         # get back the value $ra to return back to  the caller function
        lw   $s0, 4($sp)         # get back the value of $s0
        lw   $s1, 8($sp)         # get back the value of $s1
        addi $sp, $sp, 12       # free the  space locations

        #return to the caller fucntion
        jr $ra
```

```
check:                          # $a0 = n , $a1 = r

        #check for overflow:
        div   $t2,$a0,2         # $t2 = n/2
        slt   $t1,$a1,$t2       # check if the r is biger than n/2
        bne   $t1,$zero end     # if r < n/2 then  to the caller function
        sub   $t3,$a0,$a1       # nCr = (n-r)Cr then if r > n/2 would be set to n-r to avoid overflow in large numbe
        add   $a1,$t3,$zero     # r = n-r
        end:                    # end of the program
        jr    $ra               # return back to the caller function

#-------------------------------------------------------------------------------------------------------------------

nPr:                            # $a0 = n    ,  $a1 = r     $t0= counter   $v0 = result
        addi $sp, $sp, -4       # allocate one more free space
        sw   $ra, 0($sp)        # store the return address in a free location

        addi  $v0,$zero,1       # give inital value to $v0 = 1
        add   $t0,$zero,$zero   # use $t0 as counter in the loop
        loop:                   # the begining of the loop --> for(int c = 0; c<r; c++)
        beq   $t0,$a1,endloop   # if (counter == r) end
        mul   $v0,$v0,$a0       # $v0 = $v0 * $a0
        jal   overflow          # check if there is an over flow
        addi  $a0,$a0,-1        # decrease the value on n by 1
        addi  $t0,$t0,1         # increase the counter by one
        j     loop              # return back to the loop
        endloop:                # the end of the loop

        lw   $ra, 0($sp)        # get back the value $ra to return back to  the caller function
        addi $sp, $sp, 4        # free the  space locations
        # end of the loop
        jr    $ra               # retun to the caller funtion
```

```
Display:
# Display the result

        li      $v0,1                   # 1 to print an integer value that is located in $ a0
        lw      $a0,n                   # to print the number
        syscall                         # execute the system call specified by the value in $v0

        li      $v0,4                   # 4 to print a string text that is located in $ a0
        la      $a0,resultMessage       # to print the string located in the golbal variable $a0
        syscall                         # execute the system call specified by the value in $v0

        li      $v0,1                   # 1 to print an integer value that is located in $ a0
        lw      $a0,r                   # to print the value of global vriable "theAnswer"
        syscall                         # execute the system call specified by the value in $v0

        li      $v0,4                   # 4 to print a string text that is located in $ a0
        la      $a0,resultmessage2      # to print the rest of the result message
        syscall                         # execute the system call specified by the value in $v0

        li      $v0,1                   # 1 to print an integer value that is located in $ a0
        lw      $a0,theAnswer           # to print the value of global vriable "theAnswer"
        syscall                         # execute the system call specified by the value in $v0

        jr $ra                          #return to the caller function
```

```
endProgram:
        j main
```

Output:



Code Description:

1.store the data: We store that we need in the memory by (.data) and give them values. then, in main we call the functions which we used in the program.
2. we take the input: 2.1. We make a non-leaf function which do: 1.allocate more free space and store the return address because we will jump to the label which called invalid 2.we make the value of ($v0) equal 4 to make syscall print a string which in ($a0) and make the value of ($v0) equal 5 to take the integer value of (n) then we do this steps again to get the value of (r)
3. we call the function of invalid to check it validation .
4. we return to the main function. 2.2. invalid function: Its check the validation of input which must be 1.n>r 2.n and r are positive integers When one of the two condition is false, we print a error message Then return to the input function.
3. calculate the ncr combination :
3.1. We make a non-leaf function which do:
1.allocate more free space and store the return address because we will jump to the labels which called npr and check and save the value of $s0 and $s1 because we will use them.
2.we load the value of n and r from the memory by (lw)
3. check the value of r if larger than n/2 make r=n-r because ncr= ncn-r to avoid overflow
4. jump to npr to calculate npr and save the return value in $s0
5. jump to npr again to calculate rpr and save the return value in $s1
6.get the value of ncr and store the value then return the old values of $s0 and $s1 from memory and return to the main
3.2. npr function: We make a non-leaf function which do:
1.allocate more free space and store the return address because we will jump to the label which called overflow
2. we give initial value to $v0 and make $t0 counter for the loop
3. we make the loop to get the value of n by npr= n*(n-1)*(n-2)*…… to n=r and between this check if the value v0 is overflow or not 4. then return to ncr
……
4.Display: We make a leaf function which do:
1.make value of $vo=1 to print the value of n
2.then make the value of $v0=4 to print string (c)
3. make value of $vo=1 to print the value of r
4. then make the value of $v0=4 to print string (=)
5.then make the value of $v0=1 to print the answer End program: endProgram: li $v0,10 # 10 for ending the program syscall # execute the system call specified by the value in $v0

C code:

```c
#include <limits.h>
int overflow(int val){
    int MAX = INT_MAX;
    return(val > MAX || val < 1);
}
void check(int n , int *r){
    if(*r > n/2)
        *r = n - *r;
}
int nPr(int n, int r){
    int npr = 1;
    for(int i = 0; i < r; i++){
        npr *= n;
        if(overflow(npr))
            return -1;
        n--;
    }
    return npr;
}
int nCr(int n, int r){
    check(n, &r);
    int npr = nPr(n, r);
    int rpr = nPr(r, r);
    if(npr == -1 || rpr == -1)
        return -1;
    return npr / rpr;
}
int invalid(int n, int r){
    return (n < r) || (n < 0) || (r < 0);
}
int main() {
    int n, r;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    printf("Enter the value of r: ");
    scanf("%d", &r);

    if(invalid(n, r)){
        printf("Invalid input\n");
        return 1;
    }
    int ncr = nCr(n, r);
    if(ncr == -1){
        printf("Out of range\n");
        return 0;
    }
    printf("nCr(%d, %d) = %d\n", n, r, ncr);
    return 0;
}
```

Decimal to Binary function:

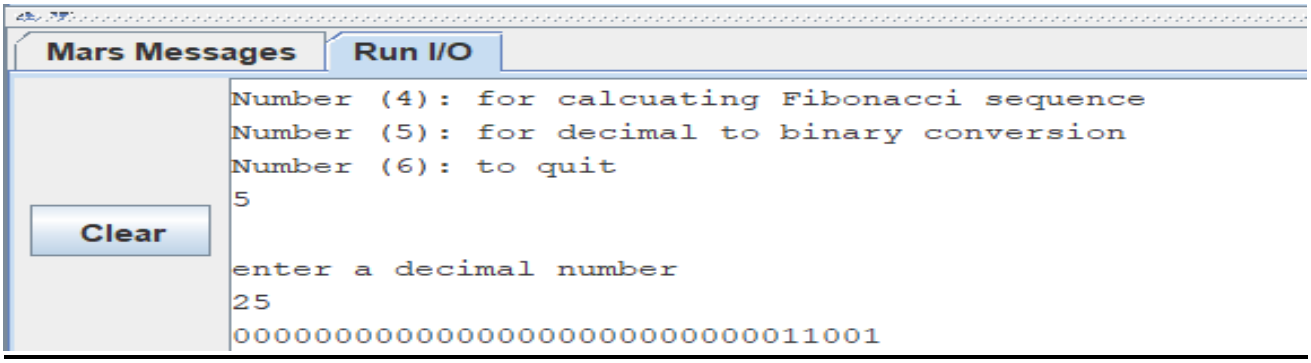
Description:

The function is used to take a decimal integer number from the user and print the binary number of this input.

Assembly Code:

```
#conversion from decimal to binary
   func5:
    jal binary          # calling the function
    j main


    binary:
   #print message eneter the number
    li $v0,4            # print_string
    la $a0,mess7        # load address of the string (printing enter a decimal number)
    syscall
    # to take the num from the user
    li $v0,5            #load data from user( input INTEGER)
    syscall             # Read int input
    move $a1, $v0       # Store user input in $a1
   addiu $t0,$zero,31   #set value 31 to register t0
   addiu $t1,$zero,0    #set value 0 to register t1

  loop1:
    srlv $t2,$a1,$t0    #place  n/2^i in $t2 (n>>i)
    and $t3,$t2, 1      #do and operator of each bit with 1
     li $v0,1           #print an integer number($t3)
    add $a0,$t3,$zero
    syscall
    addi $t0,$t0,-1      #decrease the number by one
    bge $t0,$zero,loop1  #if (i) greater than or equal 0 repeat loop1
    jr $ra               #return to the caller function

#-------------------------------------------------------------------------------------------------------------------------------

#quit
   quit:
    li $v0,10
    syscall
```

Output:



Code description:

In This code we first print a message (“enter a decimal number”), Then getting the input from the user and store it in $a1, Then we make a loop its counter is defined as $t0 ,and we initialize this counter by 31 in this loop we make right shift to the number that the user input by the counter (decimal number>>counter) and store the result in register($t2), Then we do and operator of each bit with 1 and store the result in register($t3), Then we print an integer number (0 or 1),Then we decrease the counter by one, Then we check if the counter is greater than or equal 0 if this condition is true we repeat the loop until the condition become false we will return to the caller function Then we jump to main function.

C code:

```
void decimaltobinary(int n){
    for(int i = 31 ; i >= 0 ; i--){
        int k = n >> i;
        if(k & 1)
            printf("1");
        else
            printf("0");
    }
}
```

Factorial of a number function:

Description:

This function is used to take an integer and prints the factorial of this number.

Assembly Code:

```
#Factorial

 func4:
.text
     li $v0, 4            # Load immediate value 4 into register $v0 (system call code for printing a string)
     la $a0, popMessage   # Load address of the string "popMessage" into register $a0
     syscall              # Execute the system call to print the string

     li $v0, 5            # Load immediate value 5 into register $v0 (system call code for reading an integer)
     syscall              # Execute the system call to read an integer
     sw $v0, theNumber    # Store the read integer value into the memory location "theNumber"

    # Call the factorial function.
    lw $a0, theNumber     # Load the value from memory location "theNumber" into register $a0 (function argument)
    jal findFactorial     # Jump and link to the "findFactorial" function
    sw $v0, theAnswer     # Store the result of the factorial function in the memory location "theAnswer"

   # Display the results.
   li $v0, 4             # Load immediate value 4 into register $v0 (system call code for printing a string)
   la $a0, resultMassege # Load address of the string "resultMessage" into register $a0
   syscall               # Execute the system call to print the string
   li $v0, 1             # Load immediate value 1 into register $v0 (system call code for printing an integer)
   lw $a0, theAnswer     # Load the value from memory location "theAnswer" into register $a0 (function argument)
   syscall               # Execute the system call to print the integer

# Tell the OS that this is the end of the program.
  li $v0, 10            # Load immediate value 10 into register $v0 (system call code for program termination)
  syscall               # Execute the system call to terminate the program

.globl findFactorial
 findFactorial:
 subu $sp, $sp, 8      # Subtract 8 from the stack pointer to allocate space for the return address and saved registers
 sw $ra, ($sp)         # Save the return address on the stack
 sw $s0, 4($sp)        # Save the value of register $s0 on the stack
```

```
# Base case.
  li $v0, 1            # Load immediate value 1 into register $v0 (base case for factorial)
  beq $a0, 0, factorialDone   # Branch to "factorialDone" if the value in register $a0 is equal to 0

  move $s0, $a0        # Move the value in register $a0 to register $s0 (current factorial value)
  sub $a0, $a0, 1      # Subtract 1 from the value in register $a0 (decrement for recursive call)
  jal findFactorial    # Jump and link to the "findFactorial" function (recursive call)

  mul $v0, $s0, $v0    # Multiply the value in register $s0 (current factorial value) with the return value ($v0) of the recursive call
factorialDone:
  lw $ra, ($sp)        # Load the return address from the stack into register $ra
  lw $s0, 4($sp)       # Load the value of register $s0 from the stack
  addu $sp, $sp, 8     # Add 8 to the stack pointer
# Return from factorial.
  jr $ra              # Jump to the address stored in register $ra (return from the function)

j main              # Unconditional jump to the "main" label (to start the main part of the program)
#################
```

Output:

```
Number (5): for decimal to binary conversion
Number (6): to quit
3
Enter a anumber to find its factorial:  5

 The factorial of number is 120
-- program is finished running --
```

C code:

```
#include <stdio.h>

int factorial(int n);

int main() {
    int n;

    printf("Enter a positive integer: ");
    scanf("%d", &n);

    printf("Factorial of %d = %d", n, factorial(n));

    return 0;
}

int factorial(int n) {
    if(n > 1)
        return n * factorial(n - 1);
    else
        return 1;
}
```

Fibonacci function:

Description:

The Fibonacci sequence is a sequence in which each number is the sum of the two preceding ones.

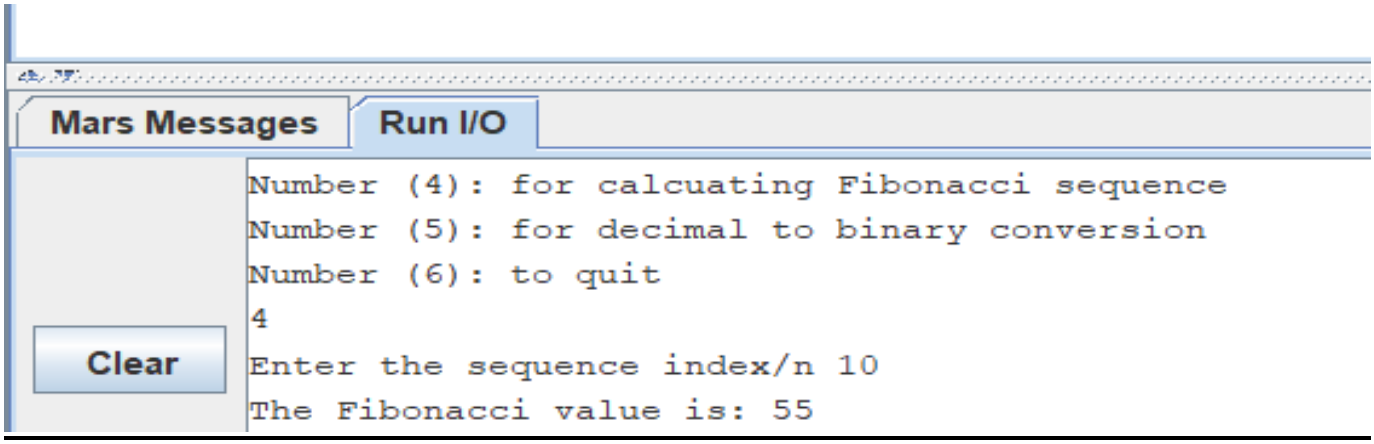
Assembly Code:

```
# Fibonicciaa sequance

 func2:
 # Message: "Enter the index"
    li $v0, 4            # Load immediate value 4 into register $v0 (system call code for printing a string)
    la $a0, prompt1      # Load address of the string "prompt1" into register $a0
    syscall              # Execute the system call to print the string

 # Read input from the user
    li $v0, 5            # Load immediate value 5 into register $v0 (system call code for reading an integer)
    syscall              # Execute the system call to read an integer

 # Call the fibonacci function
    move $a0, $v0        # Move the user input value to register $a0 (function argument)
    jal fib              # Jump and link to the "fib" function
    move $a1, $v0        # Move the return value of fibonacci to register $a1

# Print "prompt2"
    li $v0, 4            # Load immediate value 4 into register $v0 (system call code for printing a string)
    la $a0, prompt2      # Load address of the string "prompt2" into register $a0
    syscall              # Execute the system call to print the string

# Print the result
    li $v0, 1            # Load immediate value 1 into register $v0 (system call code for printing an integer)
    move $a0, $a1        # Move the value in register $a1 (fibonacci result) to register $a0 (function argument)
    syscall              # Execute the system call to print the integer

# Exit the program
    li $v0, 10           # Load immediate value 10 into register $v0 (system call code for program termination)
    syscall              # Execute the system call to terminate the program
```

```
# Function: int fibonacci(int n)
fib:
  # Save registers
  addi $sp, $sp, -16    # Allocate space on the stack for saving registers
  sw $ra, 0($sp)        # Save the return address on the stack
  sw $s0, 4($sp)        # Save register $s0 on the stack
  sw $s1, 8($sp)        # Save register $s1 on the stack
  sw $s2, 12($sp)       # Save register $s2 on the stack

  # Check base cases
  beq $a0, 0, fib_return_0     # Branch to "fib_return_0" if the value in register $a0 is equal to 0
  bne $a0, 1, fib_check_2      # Branch to "fib_check_2" if the value in register $a0 is not equal to 1
  li $v0, 1                   # Load immediate value 1 into register $v0 (base case: fibonacci(1) = 1)
  j fib_return                # Jump to "fib_return"

fib_check_2:
    bne $a0, 2, fib_recursive  # Branch to "fib_recursive" if the value in register $a0 is not equal to 2
    li $v0, 1                 # Load immediate value 1 into register $v0 (base case: fibonacci(2) = 1)
    j fib_return              # Jump to "fib_return"

fib_recursive:
    addi $sp, $sp, -4    # Allocate space on the stack for saving register
    sw $a0, 0($sp)       # Save the value of register $a0 on the stack
    addi $a0, $a0, -1    # Decrement the value in register $a0 by 1 (fibonacci(n-1))
    jal fib              # Jump and link to the "fib" function (recursive call)
    move $s0, $v0        # Move the return value of the recursive call to register $s0

    lw $a0, 0($sp)       # Load the original value of register $a0 from the stack
    addi $a0, $a0, -2    # Decrement the value in register $a0 by 2 (fibonacci(n-2))
    jal fib              # Jump and link to the "fib" function (recursive call)
    move $s1, $v0        # Move the return value of the recursive call to register $s1

    add $v0, $s0, $s1    # Add the values in $s0 and $s1 and store the result in $v0 (fibonacci(n) = fibonacci(n-1) + fibonacci(n-2))

    addi $sp, $sp, 4     # Deallocate the space on the stack used for saving register $a0
```

```
fib_return:
    # Restore registers and return
    lw $s2, 12($sp)      # Restore the value of register $s2 from the stack
    lw $s1, 8($sp)       # Restore the value of register $s1 from the stack
    lw $s0, 4($sp)       # Restore the value of register $s0 from the stack
    lw $ra, 0($sp)       # Restore the return address from the stack
    addi $sp, $sp, 16    # Deallocate the space on the stack used for saving registers
    jr $ra               # Jump to the return address

fib_return_0:
    li $v0, 0            # Load immediate value 0 into register $v0 (base case: fibonacci(0) = 0)
    j fib_return         # Jump to "fib_return"

# Print a new line
    li $v0, 4            # Load immediate value 4 into register $v0 (system call code for printing a string)
    la $a0, nl           # Load address of the string "nl" (newline) into register $a0
    syscall              # Execute the system call to print the newline

j main                # Unconditional jump to the "main" label (to start the main part of the program)
     # Exit the program
```

Output:



C code:

```c
#include <stdio.h>
#include <stdlib.h>

#include <stdio.h>

int main() {
  int n, n3 ;
  int n1 = 0;
  int n2 = 1 ;

  printf("Enter the index of the Fibonacci number: ");
  scanf("%d", &n1);

  for (int i = 0; i < n1; i++) {
    if (i <= 1) {
      n3 = i;
    } else {
      n3 = n1 + n2;
      n1 = n2;
      n2 = n3;
    }
  }

  printf("%d\n", n3);

  return 0;
}
```

Quit function:

Description:

The function is used to terminate the program.

Assembly Code:

```
#quit
    quit:
     li $v0,10
     syscall
```

Output:

```
please Enter either
Number (1): for min,max,sum,average,sort of digits of an integer
Number (2): for calculating nCr
Number (3): for caluclating the factorial of intger number
Number (4): for calcuating Fibonacci sequence
Number (5): for decimal to binary conversion
Number (6): to quit
```

TASK ASSIGNMENT:

Each teammate has contributed roughly the same efforts in implementing the assembly functions where:

- **nCr**: Mohamed Khaled Ahmed
- **Factorial**: Omar Nour Eldin Mohamed
- **Decimal To Binary**: Mohamed Hatem Abdelmenem
- **Max, Min, Sort, Average, Sum:** Mohamed Ahmed Mohamed Asar
- **Fibonacci:** Kirlos Sameh Samy

Each teammate has contributed roughly the same efforts in writing the report and the presentation.