

Group ID - 65

Member #1 : Gautam Choudhary [15114027]

Member #2 : Sandeep Pal [15114063]

B. Tech. in Comp. Sci & Enginn.

IIT, Roorkee

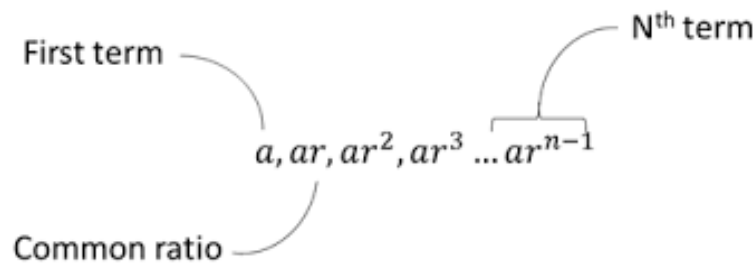
Project : Sum of 'N' consecutive terms of a GP with given 'F' and 'R'.

Submitted to : Dr. Sudip Roy

CSN-221: Computer Architecture and Microprocessors

Department of Computer Science and Engineering

Indian Institute of Technology Roorkee



A geometric progression is represented as above in most general form.

Problem Number : 9

Problem Statement : Calculate the sum of N consecutive terms of a geometric progression (GP) series with first term F and common ratio R. It should read the values "N", "F" and "R" from the user.

About Project

The project contains a

- gp_sum.c file - The file contains the c code for summation of 'n' terms of a GP.
- gp_sum.s file - The file contains the assembly code of gp_sum.c file.
- gp_sum.asm file - The file contains the assembly code.

Basic Idea

The *first term* of geometric progression is **F** and *common ratio* is **R**. Any 'k' th term of the geometric progression can be represented as-

$$T(K) = F * (R^{K-1})$$

Initialize sum to 0 and set temp variable to F.

In each iteration add temp to sum and multiply R to temp variable to generate next term of the geometric progression.

The sum obtained at the end of 'N' iterations is the sum of given GP.

C code (gp_sum.c)

This is the C code for generating the sum of 'N' terms of a GP -

```
1  // GroupID-65 (15114063_15114027) - Sandeep Pal & Gautam Choudhary
2  // Date: November 2, 2016
3  // gp_sum.c The program takes input the first term, common ratio and
4  //          number of terms of GP and prints the SUM of GP
5  #include <stdio.h>
6
7  int main(){
8      // Defining the Variables
9      /** n - Number of terms
10     ** f - First term of GP
11     ** r - Common Ratio of GP
12     */
13     int f, r, n;
14
15     // Taking input from the User (in order)
16     printf("Enter 'First' term of GP, f:  ");
17     scanf("%d", &f);
18
19     printf("Enter 'Common Ratio' of GP, r: ");
20     scanf("%d", &r);
21
22     printf("Enter the number of terms, n:  ");
23     scanf("%d", &n);
24
25
26     // Temporary Variables
27     int term = f;
28     int sum = 0;
29     int i;
30
31     // Algorithm to calculate SUM of N terms of a GP
32     for(i=0; i<n; i++){
33         sum = sum + term;
34         term = term*r ;
35     }
36
37     // Print the OUTPUT i.e. SUM
38     printf("The sum of GP is: %d\n", sum);
39
40
41     return 0;
42 }
```

Assembly Language Program (gp_sum.s file)

Using the command "**gcc -S gp_sum.c**" in command prompt terminal, we got the following assembly language program file as **gp_sum.s** –

```
1  .file "gp_sum.c"
2  .section .rodata
3  .align 8
4  .LC0:
5  .string "Enter 'First' term of GP, f:  "
6  .LC1:
7  .string "%d"
8  .align 8
9  .LC2:
10 .string "Enter 'Common Ratio' of GP, r: "
11 .align 8
12 .LC3:
13 .string "Enter the number of terms, n:  "
14 .LC4:
15 .string "The sum of GP is: %d\n"
16 .text
17 .globl main
18 .type main, @function
19 main:
20 .LFB0:
21 .cfi_startproc
22 pushq %rbp
23 .cfi_def_cfa_offset 16
24 .cfi_offset 6, -16
25 movq %rsp, %rbp
26 .cfi_def_cfa_register 6
27 subq $32, %rsp
28 movl $.LC0, %edi
29 movl $0, %eax
30 call printf
31 leaq -24(%rbp), %rax
32 movq %rax, %rsi
33 movl $.LC1, %edi
34 movl $0, %eax
35 call __isoc99_scanf
```

```

36     movl    $.LC2, %edi
37     movl    $0, %eax
38     call    printf
39     leaq    -20(%rbp), %rax
40     movq    %rax, %rsi
41     movl    $.LC1, %edi
42     movl    $0, %eax
43     call    __isoc99_scanf
44     movl    $.LC3, %edi
45     movl    $0, %eax
46     call    printf
47     leaq    -16(%rbp), %rax
48     movq    %rax, %rsi
49     movl    $.LC1, %edi
50     movl    $0, %eax
51     call    __isoc99_scanf
52     movl    -24(%rbp), %eax
53     movl    %eax, -12(%rbp)
54     movl    $0, -8(%rbp)
55     movl    $0, -4(%rbp)
56     jmp     .L2
57 .L3:
58     movl    -12(%rbp), %eax
59     addl    %eax, -8(%rbp)
60     movl    -20(%rbp), %eax
61     movl    -12(%rbp), %edx
62     imull   %edx, %eax
63     movl    %eax, -12(%rbp)
64     addl    $1, -4(%rbp)
65 .L2:
66     movl    -16(%rbp), %eax
67     cmpl    %eax, -4(%rbp)
68     jl      .L3
69     movl    -8(%rbp), %eax
70     movl    %eax, %esi
71     movl    $.LC4, %edi
72     movl    $0, %eax
73     call    printf
74     movl    $0, %eax
75     leave
76     .cfi_def_cfa 7, 8
77     ret
78     .cfi_endproc
79 .LFE0:
80     .size    main, .-main
81     .ident    "GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4"
82     .section    .note.GNU-stack,"",@progbits
83

```

The Assembly Language Program (.asm file)

- This is a description of the file showing the registers and variables used in program.

```
1  |# GroupID-65 (15114063_15114027) - Sandeep Pal & Gautam Choudhary
2  |# Date: November 2, 2016
3  |# gp_sum.asm The program takes input the first term, common ratio and number of terms of GP
4  |# and prints the SUM of GP
5  |#
6  |# Registers used:
7  |# $a0 - syscall parameter.
8  |# $v0 - syscall parameter and return value.
9  |# $t0 - used to hold current 'term' of GP
10 |# $t1 - used to hold current 'sum' of terms of GP
11 |# $t2 - used to hold value of counter variable 'i'
12 |# $t3 - used to hold number of terms of GP
13 |# $t4 - used as hold Common Ratio of GP
14 |#
15 |# Memory Variables used:
16 |# f - used to store First term of GP
17 |# r - used to store Common Ratio of GP
18 |# n - used to store Number of terms of GP
19 |# sum - used to store SUM of terms of GP
20 |#
```

- **`.data`** contains all the variable declarations. **`.ascii`** is a kind of datatype for *string* and **`.word`** is kind of datatype of *integers* with size 4bytes i.e. 32bits.

```
22 |.data
23 |    ask_input_str1: .ascii "Enter 'First' term of GP, f: "
24 |    ask_input_str2: .ascii "Enter 'Common Ratio' of GP, r: "
25 |    ask_input_str3: .ascii "Enter the number of terms, n: "
26 |    tell_output_str: .ascii "The sum of GP is: "
27 |
28 |    f: .word 0      #First term of GP
29 |    r: .word 0      #Common Ratio of GP
30 |    n: .word 0      #Number of terms of GP
31 |    sum: .word 0     #Initialize integer 'sum' with value 'zero'
32 |
```

-
- **'.text'** section includes the actual code of your program. As the comments are very descriptive itself. First of all, all the *inputs* are taken from the user and stored into *memory*.

```
34 .text
35 main:
36     #Prompts user to enter 'First' term of GP*****
37     la $a0, ask_input_str1                #loads the address of string to be printed in $a0
38     li $v0, 4                            #system call code for printing string
39     syscall                               #system call
40     #Takes input the 'First' term of GP
41     li $v0, 5    # $v0 = <user input>
42     syscall
43     #Store to memory
44     sw $v0, f;    # f = $v0
45
46
47
48
49     #Prompts user to enter 'Common Ratio' of GP*****
50     la $a0, ask_input_str2                #loads the address of string to be printed in $a0
51     li $v0, 4                            #system call code for printing string
52     syscall                               #system call
53     #Takes input the 'Common Ratio' of GP
54     li $v0, 5    # $v0 = <user input>
55     syscall                               #system call code for integer input
56     #Store to memory
57     sw $v0, r;    # r = $v0
58
59
60
61
62     #Prompts user to enter 'Number' of terms of GP*****
63     la $a0, ask_input_str3                #loads the address of string to be printed in $a0
64     li $v0, 4                            #system call code for printing string
65     syscall                               #system call
66     #Takes input the 'Number' of terms of GP
67     li $v0, 5    # $v0 = <user input>
68     syscall
69     #Store to memory
70     sw $v0, n;    # n = $v0
71
```

-
- Now the *temporary* variables are declared. Then, a **loop** is executed in which the **term** is generated and it is added to the **sum** each time. Also, the *counter variable* checks the number of iterations are no more than the '**number**' of terms.

```
75      # Temporary variables
76      lw $t0, f      # ($t0 == term) = f
77      li $t1, 0      # ($t1 == sum) = 0
78      li $t2, 0      # ($t2 == i) = 0
79      lw $t3, n      # ($t3 == n)
80      lw $t4, r      # ($t4 == r)
81
82
83
84      # LOOP Statement
85      loop: beq $t2, $t3, end # if(i == n) jump to label 'end'
86          add $t1, $t1, $t0      # (sum = sum + term)
87          mult $t0, $t4          # (<product> = term * r)
88          mflo $t0              # (term = <product>)
89          addi $t2, $t2, 1      # (i++)
90          j loop                # jump to label 'loop'
91      end:
92          #Store result back to memory
93          sw $t1, sum          # (sum = $t1)
94
95          #Prints the 'output_str' to Console
96          li $v0, 4
97          la $a0, tell_output_str
98          syscall
99          #Prints 'GP - Sum' to Console
100         li $v0, 1
101         move $a0, $t1
102         syscall
103
104         # Program exits
105         li $v0, 10
106         syscall
```


Test Case #1

- First term, $f = 3$
- Common Ratio, $r = 2$
- Number of terms, $n = 5$
- @ SUM = 93

The screenshot displays the QtSpim MIPS simulator interface. On the left, the 'Console' window shows the user input for the test case: 'Enter 'First' term of GP, f: 3', 'Enter 'Common Ratio' of GP, r: 2', 'Enter the number of terms, n: 5', and the output 'The sum of GP is: 93'. The main window shows the assembly code for a program that calculates the sum of a geometric progression. The code includes instructions for loading values into registers, performing arithmetic operations, and branching. The 'Int Regs [10]' window shows the current state of the registers: PC = 4194528, EPC = 0, Cause = 0, BadVAddr = 0, Status = 805371664, HI = 0, LO = 96, R0 [r0] = 0, R1 [at] = 268500992, R2 [v0] = 10, R3 [v1] = 0, R4 [a0] = 93, R5 [a1] = 21474810, R6 [a2] = 21474810, R7 [a3] = 0, R8 [t0] = 96, R9 [t1] = 93, R10 [t2] = 5, R11 [t3] = 5, R12 [t4] = 2, R13 [t5] = 0, R14 [t6] = 0, R15 [t7] = 0, and R16 [s0] = 0. The assembly code is as follows:

```
[00400070] 34020003 ori $2, $0, 3 ; 67: ori $v0, 3 = $v0 = 3
[00400074] 0000000c syscall ; 68: syscall
[00400078] 3c011001 lui $1, 4097 ; 70: sw $v0, n; # n = $v0
[0040007c] ac22007c sw $2, 124($1) ; 76: lw $t0, f # ($t0 == term) = f
[00400080] 3c011001 lui $1, 4097 ; 77: li $t1, 0 # ($t1 == sum) = 0
[00400084] 8c280074 lw $8, 116($1) ; 78: li $t2, 0 # ($t2 == i) = 0
[00400088] 34090000 ori $9, $0, 0 ; 79: lw $t3, n # ($t3 == n)
[0040008c] 340a0000 ori $10, $0, 0
[00400090] 3c011001 lui $1, 4097
[00400094] 8c2b007c lw $11, 124($1) ; 80: lw $t4, r # ($t4 == r)
[00400098] 3c011001 lui $1, 4097
[0040009c] 8c2c0078 lw $12, 120($1)
[004000a0] 114b0006 beq $10, $11, 24 [end-0x004000a0]
[004000a4] 01284820 add $9, $9, $8 ; 86: add $t1, $t1, $t0 # (sum = sum + term)
[004000a8] 010c0018 mult $8, $12 ; 87: mult $t0, $t4 # (= term * r)
[004000ac] 00004012 mflo $8 ; 88: mflo $t0 # (term = )
[004000b0] 214a0001 addi $10, $10, 1 ; 89: addi $t2, $t2, 1 # (i++)
[004000b4] 08100028 j 0x004000a0 [loop] ; 90: j loop # jump to label 'loop'
[004000b8] 3c011001 lui $1, 4097 ; 93: sw $t1, sum # (sum = $t1)
[004000bc] ac290080 sw $9, 128($1)
[004000c0] 34020004 ori $2, $0, 4 ; 96: li $v0, 4
[004000c4] 3c011001 lui $1, 4097 [tell_output_str]; 97: la $a0, tell_output_str
[004000c8] 34240060 ori $4, $1, 96 [tell_output_str]
[004000cc] 0000000c syscall ; 98: syscall
[004000d0] 34020001 ori $2, $0, 1 ; 100: li $v0, 1
[004000d4] 00092021 addu $4, $0, $9 ; 101: move $a0, $t1
[004000d8] 0000000c syscall ; 102: syscall
[004000dc] 3402000a ori $2, $0, 10 ; 105: li $v0, 10
[004000e0] 0000000c syscall ; 106: syscall
```

The bottom of the window shows the 'Memory and registers cleared' message and the SPIM version information: 'SPIM Version 9.1.17 of January 1, 2016', 'Copyright 1990-2016, James R. Larus.', 'All Rights Reserved.', 'SPIM is distributed under a BSD license.', 'See the file README for a full copyright notice.', 'QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.'

Test Case #2

- First term, $f = 5$
- Common Ratio, $r = 3$
- Number of terms, $n = 4$
- @ SUM = 200

The screenshot shows a MIPS simulator interface. On the left, a 'Console' window displays the following text:

```
Enter 'First' term of GP, f: 5
Enter 'Common Ratio' of GP, r: 3
Enter the number of terms, n: 4
The sum of GP is: 200
```

On the right, the 'Text' segment window shows assembly code with comments. The code implements a loop to calculate the sum of a geometric progression. Key instructions include `syscall`, `lui`, `sw`, `lw`, `ori`, `beq`, `add`, `mult`, `mflo`, `addi`, `j`, `sw`, `li`, `la`, `ori`, `addu`, and `add`. Comments explain the logic, such as `; 68: syscall`, `; 70: sw $v0, n; # n = $v0`, and `; 86: add $t1, $t1, $t0 # (sum = sum + term)`.

At the bottom, a Windows taskbar is visible with the search bar and various application icons. The system clock shows 12:12 AM on 11/3/2016.