**Project 1 Report**

Christopher Badolato

[badolato.christopher@knights.ucf.edu](mailto:badolato.christopher@knights.ucf.edu)

EEL 3801-0011: Computer Organization

2/25/2019

1. Project Description

The objective of this assignment is to create the given function in Mips code, we will take user inputs and conduct some mathematics with them. We are limited in our use of instructions (No use of the mult or div functions), therefore we must create our multiplication and divisions using loops. We have combined project part A and Part B into one Mips program that will calculate F and G (part A) and use that output to solve for h\_quotient, h\_remainder, i\_quotient, i\_remainder, and finally j\_remainder.

F = (A x A) – (B x D)

G = (A x D) + (6 x C)

h = (F / G);

i = (F + G) / h\_quotient;

j = (F– G) % i\_quotient;

1. Program Design

In this program we will solve the above functions without using the mult or div function in Mips. This means for each function we need to create loops.

First off, we need to take input from the user for our values of A, B, C and D, using our $v0 register we will call syscall to take integers from the user and store them in registers $r0, $r1, $r2, $r3 respectively. To perform the first step of multiplication we must iterate through A($r0) by creating a temporary register ($t6) to act as an iterator throughout the program adding one each time to our temporary value to ensure the correct number of loops. Adding A($r0) to our temporary register ($t4) each loop will in turn multiply the results by A($r0) and store our value onto register ($t4). After we loop, we must to reset our iterator back to the value we are going to loop through next. Next, we will perform the same operation using B($r1) as our loop iterator to multiply it by D($r3) and store that value in ($t5). Finally, we can now subtract ($t4) from ($t5) and store in into F or register ($s0). Once we store our F value onto ($s0) we can change our temporary registers.

For G, we can perform the same type of loops as above starting using $r0 as our iterator and adding $r3 to our temporary register ($t4) each iteration. As well as, using C($r2) as our iterator and adding 6 to our $t5 register. Then we can sum our values at $r4 and $r5 and store them into our register G($s1)

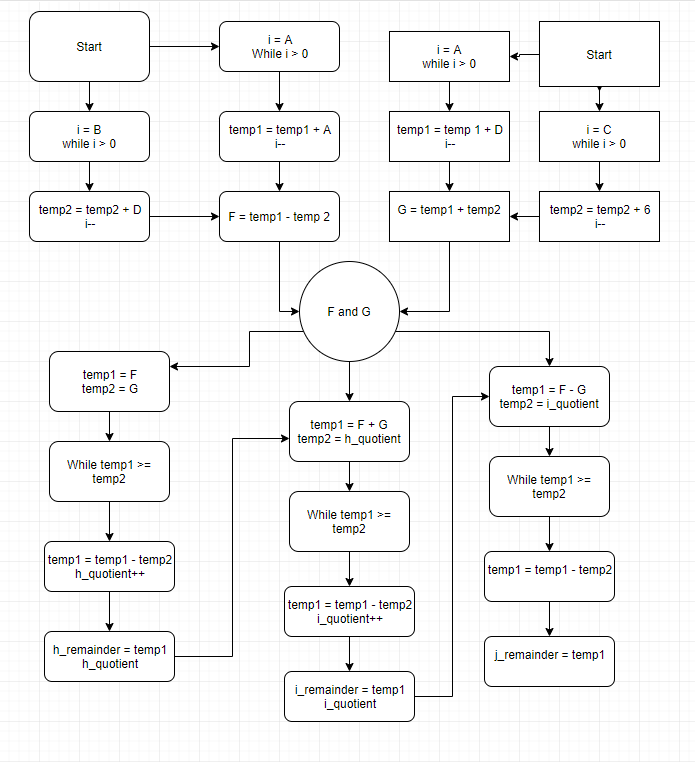
To perform the division, we must first initialize our register values for h\_quotient($s2), h\_remainder ($s3), i\_quotient ($s4), i\_remainder($s5), j\_remainder($s6) to zero. We then store our F and G as our temporary registers $t4 and $t5. To find h\_quotient($s2) and h\_remainder ($s3), we will subtract our value stored in register $t5 from our value store in $t4 until our value for $t4 is smaller than the value stored in $t5, in turn creating a while loop. Each iteration we will add one to h\_quotient($s2) giving us our quotient value. After the loop has finished, we can store our temporary value ($t4) into h\_remainder($s3) giving us our remainder.

As for i\_quotient ($s4) and i\_remainder($s5), we will apply similar loops as the division above using the sum of our values of $s0 and $s1 stored in our temporary loop value $t4. As well as, h\_quotient stored in our $t5 register. Our iterator summing one to register $s4 for i\_quotient, and storing our final value in the register $t4 into to $s5

As for our last calculation for j\_remainder($s6), we will set our temporary loop value ($t4) to the substitution of $s0 and $s1, as well as storing our value for i\_quotient($s4) to ($t5). After the loop has finished, we will store the value left in ($t4) to $s6.

Finally, we can print each value using syscall, storing each value into register $a0 to print.

We can follow the Flowchart below to see the flow of the program in C and in Mips.



1. Symbol and Label Tables

|  |  |
| --- | --- |
| Register | Assignment |
| $v0 | Tells Mars when to Syscall. |
| $t0-$t7 | Used to store temporary values throughout the program such as our inputs from the user, as well as out loop iterators. |
| $t0-$t3 | Stores input from user represents A, B, C and D in our C program (are overwritten later). |
| $t4 | Represent our temp1 in C code. Used to store results during division and multiplication during loops |
| $t5 | Represent our temp2 in C code. Used to store results during division and multiplication during loops |
| $t6 | Represent i in Code. Used as an iterator for our loops in multiplication |
| $s0 | Stores our value that represents F in our C program |
| $s1 | Stores our value that represents G in out C program |
| $s2 | Stores our h\_quotient Value |
| $s3 | Stores our h\_remainder Value |
| $s4 | Stores our i\_quotient Value |
| $s5 | Stores our i\_remainder Value |
| $s6 | Stores our j\_remainder Value |

Each loop Label has a corresponding exit condition.

|  |  |
| --- | --- |
| Label | Use |
| Results1-9 | Store the starting address of each result string |
| Loop1: Exit1: | Loops for (A x A) |
| Loop2: Exit2: | Loops for (B x D) |
| Loop3: Exit3: | Loops for (A x D) |
| Loop4: Exit4: | Loops for (6 x C) |
| Loop5: Exit5: | Represents the while loop for F/G in C |
| Loop6: Exit6: | Represents the while loop for (F + G)/ h\_quotient |
| Loop7: Exit7: | Represents the while loop for (F - G)/ h\_quotient |

1. Learning Coverage

* Looping to multiply, divide and modulus using Mips.
* Although temporary and saved registers are similar, this experiment teaches the different uses of the registers. We can see the difference between saved values and temporary ones that we can override later.
* Understand printing output and taking user input in Mips.

1. Prototype in C

Code can be easily copy and pasted into IDE.

|  |
| --- |
| // Christopher Badolato  // 3064088  // 2/25/2019  // EEL3801 0011  // this code will conduct the given functions from project 1.  // It represents the MIPS code we are writing, which involves solving  // multiplication, division, and mod, without using any of there designated functions  #include <stdio.h>  int main(){  //initializes variables, gets user input for calculations  int F, G, A, B, C, D, i = 0, temp1 = 0, temp2 = 0;  int h\_quotient = 0, h\_remainder = 0, i\_quotient = 0, i\_remainder = 0, j\_remainder = 0;  printf("please enter values for A,B,C,D \n");  scanf("%d", &A);  scanf("%d", &B);  scanf("%d", &C);  scanf("%d", &D);  //F = (A x A) – (B x D)  for(i = A; i > 0; i--){  temp1 = temp1 + A;  }  for(i = B; i > 0; i--){  temp2 = temp2 + D;  }  //subtract our two values for F and reset temps  F = temp1 - temp2;  temp1 = 0;  temp2 = 0;  for(i = A; i > 0; i--){  temp1 = temp1 + D;  }  for(i = C; i > 0; i--){  temp2 = temp2 + 6;  }  //Add our two values for G  //print f and g  G = temp1 + temp2;  printf("F\_ten: %d\n", F);  printf("G\_ten: %d\n", G);  //reset our temp values to equal F and G  //calculate h\_quotient and h\_remainder  temp1 = F;  temp2 = G;  while(temp1 >= temp2){  temp1 = temp1 - temp2;  h\_quotient++;  }  h\_remainder = temp1;  //reset our temporary values  //calculates i\_quotient and i\_remainder  temp1 = F + G;  temp2 = h\_quotient;  while(temp1 >= temp2){  temp1 = temp1 - temp2;  i\_quotient++;  }  i\_remainder = temp1;  //reset our temporary values  //calculate j\_remainder  temp1 = F - G;  temp2 = i\_quotient;  while(temp1 >= temp2){  temp1 = temp1 - temp2;  }  j\_remainder = temp1;  //output the final results.  printf("h\_quotient %d\n", h\_quotient);  printf("h\_reminder %d\n", h\_remainder);  printf("i\_quotient %d\n", i\_quotient);  printf("i\_remainder %d\n", i\_remainder);  printf("j\_remainder %d\n", j\_remainder);  return 0;  } |

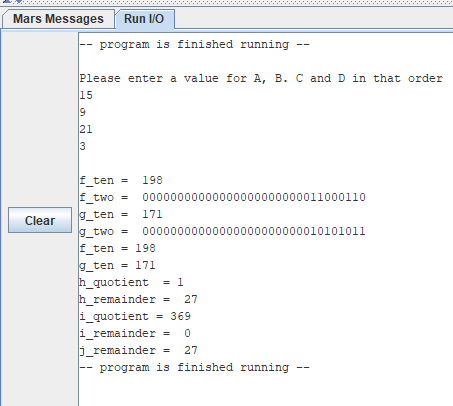
1. Test Plan

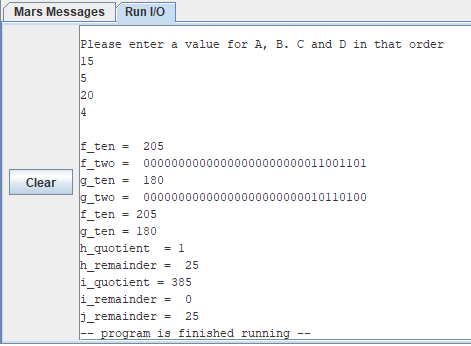
To test our Mips code, we have applied various test cases to ensure our output is correct. Each case corresponds to the results shown below in the images. We have tested and verified with our C code that these outputs correspond to the output of our Mips run version

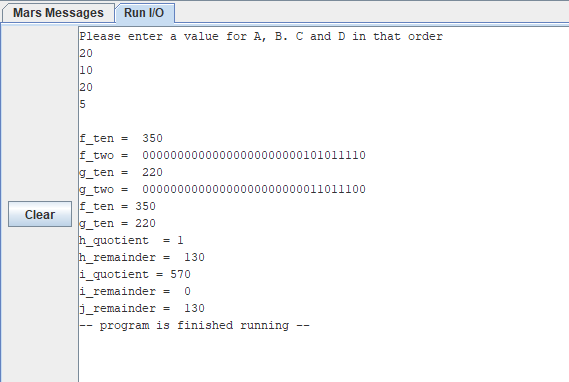
|  |  |
| --- | --- |
| Inputs | Outputs |
| A = 15  B = 9  C = 21  D = 3 | F = 198  G = 171  h\_quotient = 1  h\_remainder = 27  i\_quotient = 369  i\_remainder = 0  j\_remainder = 27 |
| A = 15  B = 5  C = 20  D = 4 | F = 236  G = 186  h\_quotient = 1  h\_remainder = 52  i\_quotient = 420  i\_remainder = 0  j\_remainder = 52 |
| A = 20  B = 10  C = 20  D = 5 | F = 350  G = 220  h\_quotient = 1  h\_remainder = 130  i\_quotient = 570  i\_remainder = 0  j\_remainder = 130 |

1. Test Results

The results of my test cases are below. I was having some issues with numbers that would change the values to all 0’s. Any instance where we would end up with an undefined fraction, or a case such that G > F, would result in the program crashing when finding i\_remainder. So, I just tested it with a few cases. Testing each of these in our C code also resulted in the below outputs.







1. References

* EEL 3801 00219 DeMara Slides, Module 03-MIPS-ISA.pdf.
* Mars 4.5 Mips Simulator
* Recitation files
* Testing reference Sheet