Project 1 Report

Samwel, Youssef

[yo800238@ucf.edu](mailto:yo800238@ucf.edu)

EEL3801: Computer Organization

Due Date: 6/28/2023

# Project Description

The objective of this project was performing multiplication, division, and modular arithmetic in MIP’s assembly. The students were tasked to compute a provided group of equations using repeated addition and repeated subtraction for multiplication and division, respectively. The inputs for part A were 4 numbers and the output was to compute f=(5\*B\*D+A) and g=(D\*D-C\*A). For part B, the input were 2 numbers, and the output was to compute h=(f\_ten+2)/(g\_ten-2) where g\_ten-2>0 and i = (f\_ten-25) mod h\_remainder.

# 2.0 Program Design

The program was based on concentrations that are the following,

* No multiplication or division instruction could be used (mul/div)
* Subroutines were not allowed to perform divisions or multiplication.

## Part A:

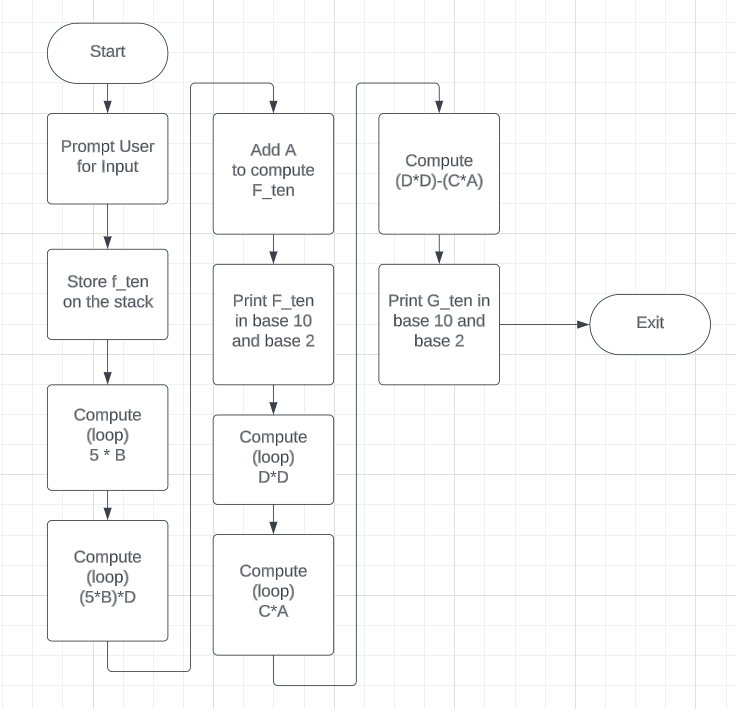
The program starts at the "start" label and prompts the user to enter four integers (A, B, C, D) using the "Prompt" message. The program reads the user input for each integer using the "read integer" syscall (system call) and stores them in memory locations A, B, C, and D. Next, the program computes the value of f, which is obtained by multiplying 5 with B and D, and then adding A. The multiplications are performed using loops. The program prints the result of f in both decimal and binary formats. Then, the program computes the value of g, which is obtained by subtracting the product of C and A from the square of D. Again, loops are used for the necessary calculations. Finally, the program prints the result of g in both decimal and binary formats. The program terminates by executing the "exit" syscall. The program uses a combination of arithmetic instructions, memory operations (loading and storing values), syscalls for input/output, and loops for repetitive computations. It also utilizes data segment (.data) to store variables and string constants. The program does not use subroutine’s for competition but to print new lines. To use a subroutine, the program uses the jal instruction to jump and return from a subroutine. When the jal instruction is executed the return address is stored in the stack pointer. Use of subroutines was not required but made the code more maintainable. Overall, the program follows a structured flow to read user input, perform computations, and print the results in both decimal and binary formats.

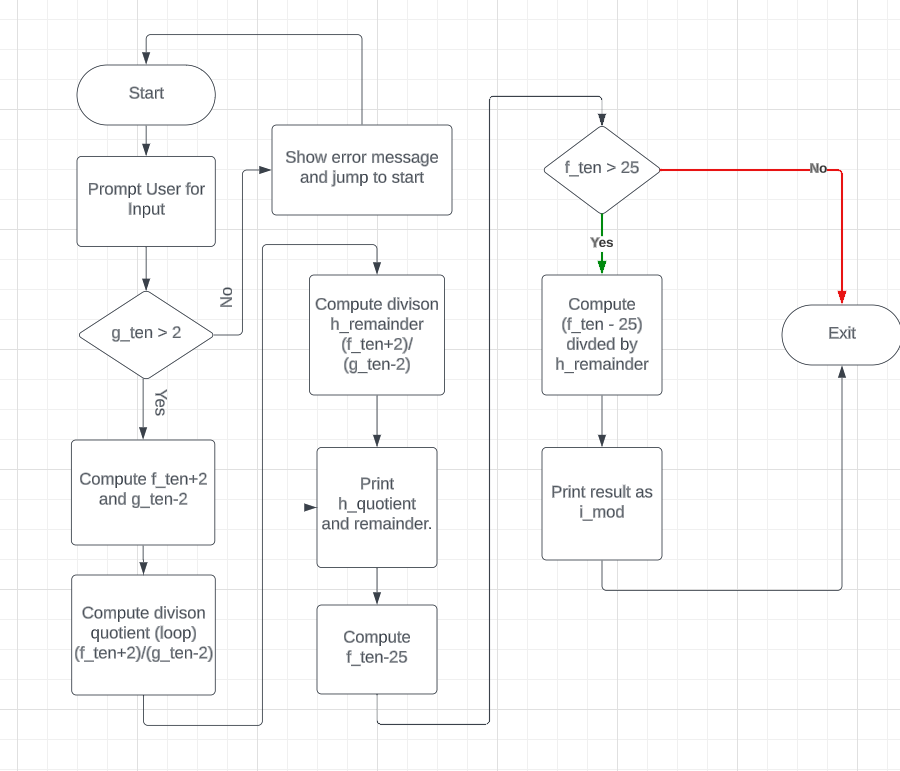
## Part B:

The program starts by prompting the user to enter two integers, "f\_ten" and "g\_ten," using the system calls for printing strings and reading integers. The entered values are temporarily stored in registers s0 (f\_ten) and s1 (g\_ten). The program checks if g\_ten - 2 is less than or equal to 0. If it is, an error message is displayed, and the program returns to the start to ask the user for valid inputs. The program performs the computation using a loop. Registers s0 and s1 are manipulated to compute the division quotient and remainder. The quotient is stored in register s3, and the remainder is stored in register s4. The loop continues until s0 is less than or equal to 0. If s0 is equal to 0, the program proceeds to print the output. The program prints the string "h\_quotient = " followed by the value stored in register s3, representing the division quotient. It then prints a new line. Next, it prints the string "h\_remainder = " followed by the value stored in register s4, representing the division remainder. Two new lines are printed for formatting. Computing Division to Determine Remainder, the program performs a similar computation as before but uses registers t0, t1, and t2 this time. The computation is based on the value stored in register s4, which is the remainder obtained in the previous step. The division quotient is stored in register t1, and the remainder is stored in register t2. If s4 is equal to 0, the program skips the computation and proceeds to print the output. The program prints the string "i\_mod = " followed by the value stored in register t2, representing the division remainder obtained in the second computation. Two new lines are printed for formatting. The program includes an exit routine that terminates the execution using the system call for program exit.

Error Handling:

If the user enters an invalid value for g\_ten, the program displays an error message indicating that g\_ten must be greater than 2. The program then returns to the start to prompt the user for valid inputs. Overall, the program follows a sequential execution flow, involving user input, computation, and display of results. It incorporates validation and error handling to ensure the program's correctness and user-friendly behavior.





# 3.0 Symbol Table

|  |  |
| --- | --- |
| Register | Purpose |
| $t0 | Loop counter and temporary storage |
| $t1 | Temporary storage |
| $t2 | Accumulator for multiplication results |
| $t4 | Storage for the value of D \* D |
| $a0 | Argument register for syscall |
| $v0 | Syscall ID register |

|  |  |
| --- | --- |
| Data Label | Purpose |
| A | Variable to store user input for integer A |
| B | Variable to store user input for integer B |
| C | Variable to store user input for integer C |
| D | Variable to store user input for integer D |
| Prompt | String constant to display prompt for user input |
| F\_Ten | String constant to display "f\_ten = " |
| F\_Two | String constant to display "f\_two = " |
| G\_Ten | String constant to display "g\_ten = " |
| G\_Two | String constant to display "g\_two = " |

|  |  |
| --- | --- |
| Register | Purpose |
| $s0 | Temporary storage for f\_ten |
| $s1 | Temporary storage for g\_ten |
| $s3 | Quotient for division operation |
| $s4 | Remainder for division operation |
| $t0 | Temporary storage for computation |
| $t1 | Quotient for division operation |
| $t2 | Remainder for division operation |
| $v0 | Syscall ID register |
| $a0 | Syscall argument register |
| $ra | Return address register |

|  |  |
| --- | --- |
| Data Label | Purpose |
| s0 | Register used to store the value of f\_ten |
| s1 | Register used to store the value of g\_ten |
| v0 | Register used for syscall arguments and syscall id |
| a0 | Register used for syscall arguments |
| s3 | Register used to store the quotient of the division |
| s4 | Register used to store the remainder of the division |
| t0 | Register used to compute (f\_ten - 25) |
| t1 | Register used to store the quotient of the second division |
| t2 | Register used to store the remainder of the second division |
| f\_ten\_str | String constant to display "f\_ten = " |
| g\_ten\_str | String constant to display "g\_ten = " |
| error\_str | String constant to display the error message for g\_ten condition |
| h\_quotient\_str | String constant to display "h\_quotient = " |
| h\_remainder\_str | String constant to display "h\_remainder = " |
| i\_mod\_str | String constant to display "i\_mod = " |
| f\_ten\_var | Variable to store the value of f\_ten |

# 4.0 Learning Coverage

* Understand MIPS registers.
* Learn about MIP’s instructions.
* Understand program segments such as .text and .data.
* Learn about looping and branching.
* Learn about registers more effectively for optimal performance.

# 5. Prototype in C-language:

## Part A

#include <stdio.h>

void print\_binary(int v) {

    char str[32];

    for (int i = 0; i <= 31; i++) {

        str[(31 - i)] = v & (1 << i) ? '1' : '0';

    }

    printf("%.32s", str);

}

int main() {

    int A, B, C, D;

    printf("Enter 4 integers for A, B, C, D respectively:\n");

    putchar('\n');

    scanf("%d", &A);

    putchar('\n');

    scanf("%d", &B);

    putchar('\n');

    scanf("%d", &C);

    putchar('\n');

    scanf("%d", &D);

    putchar('\n');

    // Compute 5\*B

    int f1 = 0;

    for (int i = 5; i > 0; i--) {

        f1 = f1 + B;

    }

    // Compute 5\*B\*D

    int f2 = 0;

    for (int i = D; i > 0; i--) {

        f2 = f2 + f1;

    }

    // Compute 5\*B\*D+A

    int f = f2 + A;

    printf("f\_ten = %d\n\n", f);

    printf("f\_two = ");

    print\_binary(f);

    puts("\n");

    // D\*D

    int g1 = 0;

    for (int i = D; i > 0; i--) {

        g1 = g1 + D;

    }

    // Compute C\*A

    int g2 = 0;

    for (int i = A; i > 0; i--) {

        g2 = g2 + C;

    }

    // Compute D\*D-C\*A

    int g = g1 - g2;

    printf("g\_ten = %d\n\n", g);

    printf("g\_two = ");

    print\_binary(g);

    puts("\n\n");

    return 0;

}

## Part B

#include <stdio.h>

int main() {

    int f\_ten, g\_ten;

    // Prompt user

    printf("f\_ten = ");

    scanf("%d", &f\_ten);

    putchar('\n');

    printf("g\_ten = ");

    scanf("%d", &g\_ten);

    putchar('\n');

    int h1 = f\_ten + 2;

    int h2 = g\_ten - 2;

    int h\_quotient = 0;

    int h\_remainder = 0;

    while (h1 > 0) {

        h1 = h1 - h2;

        h\_quotient++;

    }

    if (h1 < 0) {

        h\_quotient = h\_quotient - 1;

        h\_remainder = h1 + h2;

    }

    printf("h\_quotient = %d\n\n", h\_quotient);

    printf("h\_remainder = %d\n\n", h\_remainder);

    // (f\_ten-25) MOD h\_remainder

    int i1 = f\_ten - 25;

    int i\_remainder = 0;

    while (i1 > 0) {

        i1 = i1 - h\_remainder;

    }

    if (i1 < 0) {

        i\_remainder = i1 + h\_remainder;

    }

    printf("i\_mod = %d", i\_remainder);

    return 0;

}

# 6.0 Test Plan