Lab 06 report: Integer Multiplication and Division

Objectives

- To learn how to preform integer multiplication using MIPS language.
- To learn how to preform integer division using MIPS language.
- To learn how the hi and lo registers are used in multiplication and division.

Introduction

In MIPS there are two methods to preform multiplication and division which are using the multiplication and division instructions or using the logical shift with addition. This lab was about using the multiplication and division in MIPS. For the multiplication part there are three main instructions have been taught in the lab:

- `mult Rs, Rt` `#Multiply Rs by Rt signed.`
- `multu Rs, Rt` `#Multiply Rs by Rt unsigned.`

In these two instructions the low-order 32-bit of the product is set in LO register and the high-order 32-bit of the product is set in the HI register.

- `mul Rd, Rs, Rt` `#Multiply Rs by Rt signed and setting Rd to low-order 32-bit of the product.`

While in this instruction, the low-order 32-bit of the product is set in Rd register and the high-order 32-bit of the product is set in the HI register.

In the division, there are two main instructions:

- `div $t0, $t1` `#Divide $t0 by $t1 signed.`
- `divu $t0, $t1` `#Divide $t0 by $t1 unsigned.`

In these two instructions the quotient of the division is set in LO register and the remainder of the division is set in the HI register.

In `mult`, `multu`, `div` and `divu` the results are stored in HI and LO registers, to access them there are four instructions which are:

- `mthi Rs` `#Set the value of lo register to Rs contents.`
- `mtlo Rs` `#Set the value of lo register to Rs contents.`
- `mfhi Rd` `#Set the value of Rd register to hi contents.`
- `mflo Rd` `#Set the value of Rd register to lo contents.`

Tasks

Task1:

1. **Requirement:** it is required to implement a MIPS program that calculates the minimum number of banknotes (500, 100, 50, 10, 5, 1) that are required to withdraw a specific amount of money that is requested by the user.
2. **Approach:** to implement this program it is required to prompt the user to enter a specific amount of money by using (syscall 4). Then the user should enter the amount of money using (syscall 5) then store it in a temporary register such as $s0, **but the number entered should be integer and more than zero**. To calculate and print the number of each banknote category there have to some steps that should be followed:
    1. Define a label for each banknote such as `find500:`.
    2. Initialize the banknote amount in a register such as `li $t1, 500.`
    3. Divide the total amount of money by the amount of the banknote by using `div` instruction, such as `div $s0, $t1.`
    4. Store the quotient (number of 500-banknote) which is stored in `lo` register in a temporary register by using `mflo` instruction, such as `mflo $t0.`
    5. Store the remainder (the remained amount of cash) which is stored in the `hi` register in a temporary register by using `mfhi` instruction, such as `mfhi $s0` it should be stored in the total amount register because the total amount is reduced.
    6. Check if there is a banknote available for the specific category by using `beqz` instruction such as `beqz $t0, find100`, it will skip the printing instructions for this banknote.
    7. Print the banknote message using (syscall 4).
    8. Print the banknote amount using (syscall 1)

    Now the 500-banknote amount is calculated. This method is implemented with all banknotes except 1-banknote it starts from 6, because it is already the remainder of the division of 5 banknote amount.

    Finally, terminate the program.

Tasks

Task2 - a:

1. **Question:** What is the maximum value of n such that n! can fit in a 32-bit register?
2. **Answer:** the maximum value of n such that n! can fit in a 32-bit register is 12. It is 12 because the maximum 32-number in integer represented in 32-bit binary form is 2^{32}-1 which is equal to 2,147,483,647 and 12! Is equal to 479,001,600. However, 13! Is equal to 6,227,020,800 which is greater than 2^{32}-1. Therefore, 12 is the maximum value of n such that n! can fit in a 32-bit register.

Task2 - b:

1. **Requirement:** it is required to implement a MIPS program to calculate the factorial (n!) of the number n that is entered by the user following the shown java code:

```
int fact(int n){
    int result = 1;
    for (int i=1; i<=n; i++){
        result = result * i;
   }
    return result;
       }
```

2. **Approach:** to implement this program it is required to prompt the user to enter a number n by using (syscall 4). The user will enter n using (syscall 5). The number entered by the user should be stored in a temporary register $s0 using move or add instructions. The program should check if the number is greater than or equal to zero to give a true value. To check if the number is zero or positive, we should use bgez $s0, next (if n >= 0 it will calculate the factorial. Else, it will inform the user about the error then will re-ask him to enter another number). To calculate the factorial the factorial should be initialized by 1 because it is a multiplication process such as li $t0, 1. Then to implement the loop we should do the following:
    1. initialize a number that will iterate the loop until it reaches the entered number n such as li $t1, 1.
    2. Define a label for the loop such as loop: .
    3. In the loop do the following:
        i. Check if the iterator $t1 is greater than or equal to n ($s0) by using bgt instruction, such as bgt $t0, $s0, endLoop (if it reached n it will exit the loop).
        ii. Multiply the iterator $t1 by the previous factorial value $t0 by mul instruction, such as mul $t0, $t0, $t1.
        iii. Increment the iterator $t0 by 1 using addiu instruction, such as addiu $t1, $t1, 1.
        iv. Jump to the label of the loop using j instruction, such as j loop.

    Now the factorial is calculated so we should print the result message using (syscall 4) then print the result using (syscall 1).

    Finally, terminate the program.

Conclusion

In conclusion, MIPS offers multiplication and division instructions without necessarily using shift instructions. also, MIPS division instructions calculates the division quotient and remainder using a single instruction instead of two steps as some languages.