MIPS Project:

Recursive Factorial and Sorting Algorithms using SPIM

CSCI 3301 - Computer Organization

Brandon Vo

Dr. Abdullah Nur

UNO Computer Science

March 9th, 2023

Objectives

The purpose of this project is to design two MIPS assembly programs that will perform a recursive factorial n algorithm and a signed integer array sorting algorithm of choice using the QtSPIM MIPS assembler software.

Theory

MIPS is the core assembly programming language used in the course. It is a RISC, reduced instruction set computer. We will be using QtSPIM software to run our MIPS assembly code.

For the factorial n algorithm, we will be using recursion. Recursion is a method to approach algorithm implementation, where a function calls itself until a base case is met. We can use recursive code by setting a function with arguments that the user will input. These inputs are used in the function, which will either branch to a recursive function call or end if the base case is already met, such as the input of 0 or 1.

$$n! = \prod_{k=1}^{n} k$$

Fig. 1) Factorial function

In our case, a recursive factorial function will run at average time of Θ(n), worst case of O(n), and best case of constant time Ω(1), rather than a factorial time of O(n!), which is extremely bad. For 13!, there are not enough bits to represent the value of 6 million. This means that we will throw an error message for any value greater than 12. For any value less than 0 (negative), we will also throw an error. These are done by using comparisons like branching if less than or greater than or equal to, which branches towards specific functions that output the errors. For example, we can branch to the function that outputs the negative message when the comparison of the register value is less than 0.

```c
#include<stdio.h>
long int multiplyNumbers(int n);
int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Factorial of %d = %ld", n, multiplyNumbers(n));
    return 0;
}

long int multiplyNumbers(int n) {
    if (n>=1)
        return n*multiplyNumbers(n-1);
    else
        return 1;
}
```

Fig. 2) Example of recursive factorial in C

For the sorting algorithm, any algorithm implementation for sorting works. Sorting algorithms that run in average case of $\Theta(n \log(n))$, worst case of $O(n \log(n))$, and best case of constant time $\Omega(n \log(n))$, are usually either the merge sort, quick sort, or heap sort. These are incredibly difficult to implement in assembly programming, in which it would be a really big waste of time to even attempt.

In our case, we use the bubble sort, which is rather inefficient, but incredibly quick to implement. The bubble sort works by iterating through the array and swapping values if the A[i]> A[i+1] condition is met. This means that on average we will have a $\Theta(n^2)$, worst case $O(n^2)$, and best case of $\Omega(n)$, since we either have to iterate through the array n*n times to swap all values, or we just have to iterate through the array once if it is already sorted.

For edge cases, we accept both positive and negative numbers in the array. For an empty array, the array will just return as an empty array.

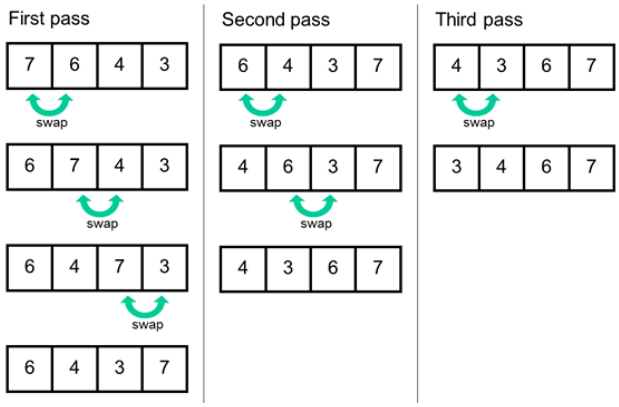

Fig. 3) Bubble Sort Algorithm

In both functions, we prompt the user with .ASCIIZ data, which are string declarations in MIPS. These strings are sent to the console through system calls through the LA instruction, which loads the message address into an argument in preparation for the syscall. Loading the integer 4 into $v0 tells the MIPS ISA that we want to print a string. Loading the integer 5 into $v0 indicates the scanning of a user-input integer. Using the system call will then prompt the user input and place the number in $v0. For edge cases, we simply output an empty message when the user enters a 0 for the array size. For positive and negative combined in arrays, our algorithm works so that both are valid, since the sorting algorithm uses memory addressing.

```cpp
#include <bits/stdc++.h>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}

// Function to print an array
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = { 5, 1, 4, 2, 8};
    int N = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, N);
    cout << "Sorted array: \n";
    printArray(arr, N);
    return 0;
}
```

```cpp
#include <stdio.h>

void swap(int* xp, int* yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already in p
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = { 5, 1, 4, 2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Fig. 4) Examples of Bubble Sort in C/C++

With this being known, we can create the final code for the project, which allows users to get input and output through the QtSPIM terminal.

Code

Code can be found in the appendix. Comments explain each function

Results (Factorial)

Console

```
Factorial Calculator
Enter an inclusive INTEGER between 0 and 12: 6
720
```

Console

```
Factorial Calculator
Enter an inclusive INTEGER between 0 and 12: 7
5040
```

Edge Casing (Factorial)

Console

```
Factorial Calculator
Enter an inclusive INTEGER between 0 and 12: 13
ERROR: Number greater than 12 not in range.
```

Console

```
Factorial Calculator
Enter an inclusive INTEGER between 0 and 12: 1
1
```

Console

```
Factorial Calculator
Enter an inclusive INTEGER between 0 and 12: -2
ERROR: Negative numbers not in range.
```

Results (Sort)

Console

```
MIPS Array Sorting Program with Bubble Sort.
Enter the number of elements of the integer array: 3
Enter the integers in the array, seperated by lines:
2
-9
-200
You have entered the array: 2 -9 -200
Here is the sorted array in ascending order: -200 -9 2
```

Console

```
MIPS Array Sorting Program with Bubble Sort.
Enter the number of elements of the integer array: 8
Enter the integers in the array, seperated by lines:
1
2
4
3
5
7
6
8
You have entered the array: 1 2 4 3 5 7 6 8
Here is the sorted array in ascending order: 1 2 3 4 5 6 7 8
```

Console

```
MIPS Array Sorting Program with Bubble Sort.
Enter the number of elements of the integer array: 5
Enter the integers in the array, seperated by lines:
-1
-2
-3
-4
-5
You have entered the array: -1 -2 -3 -4 -5
Here is the sorted array in ascending order: -5 -4 -3 -2 -1
```

Edge Casing (Sort)

Console

```
MIPS Array Sorting Program with Bubble Sort.
Enter the number of elements of the integer array:
The array is empty.
```

Discussion and Conclusion

Based off the results of our testing, our programs works clearly following the guidelines of the project. All listed edge cases were properly handled with errors or with assigned values. While doing this project, I was able to understand data structures and algorithms in more depth, due to being forced to do it in a lower level language compared to C or C++. I also learned more about the applications of assembly, system calls, and general systems level design as well.

Appendix

CSCI 3301 – MIPS Programming Using SPIM

Programming Assignment

Due Date: May 11, 2023, 11:59 PM

Instructions:

- The assignment must be done by you alone.
- Your programming code must be well commented.
- You need to write a report. You need to include your output screenshots in the report and briefly explain.
- You need to test your code for edge cases. Explain how you checked and solved the edge cases in the report.
- For the two MIPS-programs, you will generate 2 text files, named Facto.s and Sort.s. Put them in a folder.
- Put your report as a PDF file, compress the folder and submit the compressed folder.
- If you don't write the report, your assignment will not be graded.

Total Marks: 100.

Part 1 [50 points]:

The factorial function is mathematically defined as:

$$n! = \prod_{k=1}^{n} k$$

For example, factorial of 6, that is 6! = 720, which is computer as: $6 \times 5 \times 4 \times 3 \times 2 \times 1$.
Write a ***recursive*** MIPS assembly program using SPIM simulator (such as QtSpim), where for a given number (n) as an input (taken from the console), the program will recursively compute and return the corresponding factorial number. For retuning the result, use the standard console (screen) and print the output. You program must be able to take positive as well as negative integer numbers as input and must be able to respond appropriately.

Edge cases:

- 0! = 1
- Negative factorial is not defined. Return an error message.
- After 12 factorial, the output is larger than 32 bits. Therefore, return out of boundary error message.
- Also show some regular outputs, e.g. 6! = 720

Part 2 [50 points]:
Develop a MIPS program, "SORT", to sort a set of given integers (note: both positive and negative integers are allowed) by the user. Note that, you can use any sorting algorithm you want including the Bumble sort which we covered in the class.

> First, your program will ask the user, how many numbers the user wants to enter, and then ask the user to provide those numbers.
> Your program will read in those numbers and
> will place those numbers in an array.
> Sort the array elements in ascending order.
> Now, print the set of numbers that the user entered saying, "You have entered: …".
> Then, print the sorted list saying, "Here is the sorted list in ascending order: …".

Edge cases:
- Empty array
- All negative integers in the array
- All positive integers in the array
- Positive and negative integers together in the array

Factio.s

```
.data

        initString: .asciiz "Factorial Calculator\n"

        promptUser:   .asciiz "Enter an inclusive INTEGER between 0 and 12: "

        resultString: .asciiz "Result of factorial: "

        errorNumberNegative: .asciiz "ERROR: Negative numbers not in range.\n"

        errorNumberTooBig: .asciiz "ERROR: Number greater than 12 not in range.\n"


.text

.globl main

main:

        #Print Info about Program and Prompts User for Integer

        li $v0, 4

        la $a0, initString
```

```
        syscall
    la $a0, promptUser
    syscall

        #Read Input and Save to $a0
        li $v0, 5
        syscall
        add $a0, $v0, $zero

        #Check if Number is Too Big
        addi $t0, $zero, 12
        bgt $a0, $t0, numTooBig

        #Check if Number is Negative
        addi $t0, $zero, 0
        blt $a0, $t0, negativeNumber

        #Check Special Case When Num = 0
        addi $t0, $zero, 0
        beq $a0, $t0, numIsZero

        #Get Factorial When Previous Cases False
        add $t0, $a0, $zero #Move Inpout to $t0 (for recursion calls)
        add $t2, $a0, $zero #Move Inpout to $t2 (for recursion jump backs)
        addi $s0, $zero, 1
        j factorialLoop

#Call Recursion Until Num = 1
```

```
factorialLoop:
        #t0 = Current Iteration Index
        beq $t0, $zero, factorialUnloop #Stop recursion calls and start 'hopping' back out when num = 1
        addi $sp, $sp, -4 #Allocate stack space for current Iteration Index
        sw $t0, ($sp) #Save current Iteration Index to stack
        addi $t0, $t0, -1 #Decrement iteration
        j factorialLoop

#'Hop Out' of the Recursion call stack
factorialUnloop:
        #s0 = Current Result of Factorial (starts at 1)
        beq $t2, $zero, Exit #Exit program when iteration index = 0
        addi $t2, $t2, -1 #Decrement iteration idnex
        lw $t5, ($sp) #Load recursive call items from stack to $t5
        addi $sp, $sp, 4 #Move down stack when item retrieved
        mul $s0, $s0, $t5 #Multiply current result of factorial by current iteration
        j factorialUnloop

numTooBig:
        #Print Too Big Error Message and Exit Program
        li $v0, 4
        la $a0, errorNumberTooBig
        syscall
        li $v0, 10
   syscall

negativeNumber:
        #Print Negative Number Error Message and Exit Program
```

```
        li $v0, 4
        la $a0, errorNumberNegative
        syscall
        li $v0, 10
    syscall

numIsZero:
        #Set Result of Factorial to 0 and jump to Exit
        li $v0, 4
        la $a0, resultString
        syscall
        addi $s0, $zero, 1
        j Exit

Exit:
        #Print Number
        li $v0, 1
        add $a0, $zero, $s0
        syscall

        #Terminate Program
    li $v0, 10
    syscall
```

Sort.s

```
    .data
    initString: .asciiz "MIPS Array Sorting Program with Bubble Sort.\n"
```

```
    promptUserArraySize:   .asciiz "Enter the number of elements of the integer array: "
    promptUserArrayItems:   .asciiz "Enter the integers in the array, seperated by lines: \n"
    emptyArraySize: .asciiz "The array is empty.\n"
    preresultString: .asciiz "You have entered the array: "
            resultString: .asciiz "Here is the sorted array in ascending order: "
    integerArray: .word 24

    .text
    .globl main
main:
    #Print Info about Program and Prompts User to Enter Array Size
            li $v0, 4
            la $a0, initString
            syscall
    la $a0, promptUserArraySize
    syscall

    #Read Input of Array Size and Save to $a0
            li $v0, 5
            syscall
            add $a0, $v0, $zero #Save Array Size to $a0
    add $s2, $zero, $a0 #Save Array Size to $s2
    sll $s1, $s2, 2 #Save Memory Offset of Array to $s1

    #Exit if Array Length = 0
    beq $s2, $zero, emptyArray

    #Prompt Users for Items
```

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

#Initialize Array
la $s0, integerArray #Save Memory Location of intArray to $s0
add $t0, $zero, $s2 #Set Amount of Iterations to $t0
jal createArray

#Print Unsorted Array
li $v0, 4
	la $a0, preresultString
syscall
la $s0, integerArray
add $t0, $zero, $s2
jal printArray

#Sort Array
add $t1, $zero, $s2 #Set Amount of Iterations of Outer Loop to $t1
jal sortArray

#SAVE FIRST 6 ARRAY ITEMS TO t1-t6
#FOR DEBUGGING PURPOSES
la $s0, integerArray #Save Memory Location of intArray to $s0
lw $t1, 0($s0)
lw $t2, 4($s0)
lw $t3, 8($s0)
lw $t4, 12($s0)
```

```
    lw $t5, 16($s0)
    lw $t6, 20($s0)

    #Print Sorted Array
    li $v0, 4
        la $a0, resultString
    syscall
    la $s0, integerArray
    add $t0, $zero, $s2
    jal printArray

    j Exit

createArray:
    #Initialize Array of User-defined Size with User's Inputs
    li $v0, 5 #Read Input
    syscall #Read Input
    add $t2, $zero, $v0 #Set $t2 to User's Input
    sw $t2, ($s0) #Save Input to Current List at Index
    addi $s0, $s0, 4 #Increment Stack for Next Word
    addi $t0, $t0, -1 #Decrement Iteration Index
    bne $t0, $zero, createArray
    sub $s0, $s0, $s1
    jr $ra

sortArray:
    #Bubble Sort Outer Loop
    la $s0, integerArray #Save Memory Location of intArray to $s0
```

```
    addi $s0, $s0, -4 #Decrement $s0 by one byte for Inner Loop Init
    sw $ra, ($sp) #Save Return to Main PC to Stack
    add $t0, $zero, $t1 #Set Amount of Iterations of Nested Loop to $t0
    jal swapLoop #Inner Loop
    addi $t1, $t1, -1
    lw $ra, ($sp) #Restore Main PC to $ra
    bne $t1, $zero, sortArray
    jr $ra #Jump back to Main when Sorting Done

swapLoop:
    #Bubble Sort Nested Loop
    addi $t0, $t0, -1 #Decrement Iteration Amount
    addi $s0, $s0, 4 #Update $s0 to next array index

    #Exit Loop When All Array Items Have Been 'Looked At'
    beq $t0, $zero, exitLoop

    #Swap if ($s0) > 4($s0) since it would not be in order
    lw $t4, ($s0) #Load $t4 with Array[Index]
    lw $t5, 4($s0) #Load $t5 with Array[Index + 1]
    ble $t4, $t5, swapLoop
    sw $t5, ($s0)
    sw $t4, 4($s0)


    j swapLoop

exitLoop:
```

```
    jr $ra

printArray:
    #Print array whose starting memory is in $s0 and size is in $t0

    #Print Item
    li $v0, 1
    lw $a0, ($s0)
        syscall

    #Print Space
    li $v0, 11
    addi $a0, $zero, 32
        syscall

    addi $t0, $t0, -1 #Decrement Iterations
    addi $s0, $s0, 4 #Move to next array item
    bne $t0, $zero, printArray #Repeat if Iteration =/= 0

    #Print New Line and Return
    addi $a0, $0, 10
    addi $v0, $0, 11
    syscall
    jr $ra

emptyArray:
    li $v0, 4
        la $a0, emptyArraySize
```

```
    syscall
    j Exit

Exit:
    li $v0, 10
    syscall
```