Indian Institute of Technology (IIT-Kharagpur)

AUTUMN Semester, 2021
COMPUTER SCIENCE AND ENGINEERING

Computer Organization and Architecture Laboratory
MIPS Assignment 3

September 21, 2021

AIM: To get proficient in writing recursive functions in MIPS along with handling arrays, allocating variables dynamically, writing function subroutine and passing parameters to functions. **No credit will be given for an iterative (linear) implementation**. Your program must have **recursive function** as specified in the questions.

INSTRUCTIONS: Make one submission per group in the form of a single zipped folder containing your source code(s). Name your submitted zipped folder as Assgn_3_Grp_GroupNo.zip and (e.g. Assgn_3_Grp_25.zip). Inside each submitted source files, there should be a clear header describing the assignment no., problem no., semester, group no., and names of group members. The file name should be of the format QuestionNo_Grp_GroupNo.s (e.g. Q1_Grp_25.s). Liberally comment your code to improve its comprehensibility.

Question 1

Write a complete MIPS-32 program that -

1. Prompts the user for four positive integers n, a r, m as "Enter three positive integers (n, a, r and m) : ".
2. Allocates space for an $n \times n$ square matrix in integer array A. Populate the array A in a row major fashion using a Geometric Progression (GP) series with initial value a and common ratio r such that the i^{th} element $A[i] = (ar^i) \bmod m$.
3. Print the elements of matrix A.
4. Recursively computes the determinant of the matrix A. The value of determinant of a matrix can be calculated by following Laplace expansion.

Laplace expansion expresses the determinant of a matrix A in terms of determinants of smaller matrices, known as its minors. The minor $M_{i,j}$ is defined to be the determinant of the $(n-1) \times (n-1)$ matrix that results from A by removing the i^{th} row and the j^{th} column. The expression $(-1)^{i+j} M_{i,j}$ is known as a cofactor. For every i, one has the equality given in Equation 1 which is called the Laplace expansion along the i^{th} row. The computation of minor is recursive in nature.

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} M_{i,j} \cdot A[i][j] \tag{1}$$

The above expression reduces the matrix dimension considering any i-th row. It can similarly be done w.r.t. any j-th column.

5. Prints the final determinant with suitable message as "Final determinant of the matrix A is ".

Follow these implementation-level constraints while writing your code. Write the following functions:

1. "initStack" : Initialise the stack pointer ($\$sp$) and frame pointer ($\$fp$).
2. "pushToStack" : This function takes one argument as input (in $\$a0$) and push it to the stack.
3. "popFromStack" : This function does not take any argument and returns the first element in the stack.
4. "printMatrix" : This function takes two parameters- the positive integers n (in $\$a0$) and the address of the two-dimensional $n \times n$ integer array A (in $\$a1$). It prints the elements of A in a row-major fashion.
5. Write a recursive subroutine *recursive_Det* that is passed the following parameters- a positive integer n' and the address of any intermediate matrix A' stored in the two-dimensional $n' \times n'$ integer array. It returns the determinant of the matrix A'.

If required, you can write additional functions as well, but with proper comments and descriptions.

Question 2

Write a complete MIPS-32 program that -

1. Reads an array of ten integers from the user (can also be negative). These numbers are collected from the input console using a loop and stored in the memory in an array called 'array'. Do not store the numbers as scalars in ten different non-contiguous locations or in ten different registers.

2. Write a recursive function named *recursive_sort* that takes the start address, start index and end index of an array in order to sort the array recursively. You have to implement your code following Algorithm 1 as given below.

3. After sorting, print the sorted array on the console with a proper message as "Sorted array :" .

Follow these implementation-level constraints while writing your code. Write the following helper functions:

1. "initStack" : Initialise the stack pointer ($\$sp$).

2. "pushToStack" : This function takes one argument as input and push it to the stack.

3. "SWAP" : The function takes two array elements as inputs and perform swap operation.

4. "printArray" : This function takes the array address and array size and prints the elements of A.

If required, you can write additional functions as well, but with proper comments and descriptions.

Algorithm 1 recursive_sort($A, left, right$**)**

$l \leftarrow left, r \leftarrow right, p \leftarrow left$;
while $l < r$
    while $A[l] \leq A[p]$ **and** $l < right$
        $l++$;
    while $A[r] \geq A[p]$ **and** $r > left$
        $r--$;
    if $l \geq r$ **then**
        SWAP(A[p], A[r]); // Swap the array elements
        recursive_sort(A, left, r-1);
        recursive_sort(A, r+1, right);
        return;
    SWAP(A[l], A[r]);

Question 3

Write a complete MIPS-32 program that -

1. Reads an array of ten integers from the user (can also be negative). Read an integer (n) from the user to be searched in the array.

2. Sort the 1-D array in ascending order using the *recursive_sort* function implemented in the previous question, and print the sorted array with the message – "Sorted array: ".

3. Write a recursive function *recursive_search* to search the array for the presence of the value *key* in the array following the Algorithm 2 given below. The address of the sorted array and *key* are passed as argument to implement the *recursive_search* function. The function returns the index where key is found, or return -1 if not found.

4. If the search is successful, the program will print an appropriate success message with the array index (i) where the value was found, such as- "$< n >$ is FOUND in the array at index $< i >$.".

5. If the search is unsuccessful, the program will print a failure message, such as "$< n >$ NOT FOUND in the array.".

Follow these implementation-level constraints while writing your code. Write the following helper functions:

1. "initStack" : Initialise the stack pointer ($\$sp$).

2. "pushToStack" : This function takes one argument as input and push it to the stack.

3. "printArray" : This function takes the array address and array size and prints the elements of A.

If required, you can write additional functions as well, but with proper comments and descriptions.

Algorithm 2 recursive_search($A, start, end, key$**)**

while $start \leq end$
    $mid1 \leftarrow start + (end - start)/3;$
    $mid2 \leftarrow end - (end - start)/3;$
    if $key == A[mid1]$ **then**
        return $mid1;$
    else if $key == A[mid2]$ **then**
        return $mid2;$
    else if $key < A[mid1]$ **then**
        return $recursive\_search(A, start, mid1 - 1, key);$
    else if $key > A[mid2]$ **then**
        return $recursive\_search(A, mid2 + 1, end, key);$
    else
        return $recursive\_search(A, mid1 + 1, mid2 - 1, key);$
return -1