1 Multi-dimensional array in MIPS

1.1 MIPS program to find the sum of diagonal elements of a matrix

CODE

Here is the MIPS code to perform diagonal sum of a matrix.

```
.data
    myMatrix: .word 2, 5
                .word 10, 7

    size: .word 2

    .eqv DATA_SIZE 4

    sum: .word 0

    sumIs: .asciiz "Sum of the diagonal elements of the matrix is: "

.text
    main:
        la $a0, myMatrix

        lw $a1, size

        jal sumDiag

        li $v0, 4
        la $a0, sumIs
        syscall

        li $v0, 1
        lw $a0, sum
        syscall

        li $v0, 10
        syscall

    sumDiag:
        li $t0, 0
        lw $t1, sum

        diagLoop:
            mul $t2, $t0, $a1
```

```
        add $t2, $t2, $t0
        mul $t2, $t2, DATA_SIZE
        add $t3, $t2, $a0

        lw $t2, ($t3)
        add $t1, $t1, $t2
        addi $t0, $t0, 1

        blt $t0, $a1, diagLoop

    jr $ra
```

You can get the code from here

1.1.1 Creating the matrix

Matrix in MIPS

Creating a 2D array in MIPS is simple(as can be seen from the code). The matrix that we created in the code above is a square matrix.

Defining constants

In MIPS, defining a constant can be done using *.eqv*. It's like the **#define** preprocessor in C.

sumDiag

This function performs the diagonal sum of the matrix. Here, we have used the **row-major** approach. **$t0** is the index. It's both the row index and also the column index because we are working with only diagonal elements and not with all the elements.

$t1 represents the sum of the diagonal elements of the matrix.

diagLoop

size basically represents the dimensions of the matrix. Here **size** is 2 which means that the dimensions of the matrix is 2×2

The formula that we have used is

$$addr = baseAddr + (rowIndex * colSize + colIndex) * dataSize$$

1.1.2 Which portion represents what

mul $t2, $t0, $a1

This instruction represents $rowIndex * colSize$ because **$t0** represents the index and **$a1** represents the size of the matrix and the result is being stored

into **$t2**.

add $t2, $t2, $t0

This instruction represents $rowIndex * colSize + colIndex$. **$t2** now stores the product of **$t0** and **$a1** so we are adding the column index which is **$t0** and storing the sum into **$t2**.

mul $t2, $t2, DATA_SIZE

This instruction represents $(rowIndex * colSize + colIndex) * dataSize$. We are multiplying **$t2** with **DATA_SIZE** and storing the result into **$t2**.

add $t3, $t2, $a0

This instruction represents $baseAddr + (rowIndex * colSize + colIndex) * dataSize$. We are adding the base address(stored in **$a0**) to the result, that we had previously obtained, stored in **$t2** and storing that sum into **$t3**. We are not storing the result into **$a0** because we donot want to lose the base address of the matrix.

lw $t2, ($t3)

We are loading the value stored in the address in **$t3** and storing it into **$t2**.

add $t1, $t1, $t2

We are adding the value in **$t2** to **$t1** which represents the sum of the diagonal elements of the matrix.

addi $t0, $t0, 1

We are incrementing the index by 1 here.

blt $t0, $a1, diagLoop

We are basically checking whether **$t0** is smaller than the size of the matrix or not. If it is, then the loop will continue to sum up the diagonal elements. But if it is equal to or greater than the size of the matrix then the function **sumDiag** will terminate.