COMPUTER ARCHITECTURE

Assignment 2

~Kanav Bhardwaj, Ivan Bhargava, Lesin

IMT2023024, IMT2023022, IMT2023565

We have used 3 programs :- Matrix 2x2 multiplication, Fibonacci series and array insertion

1. Matrix 2x2

   First, it loads the values of both matrices into registers. Then, it calculates each element of the resulting matrix by performing multiplication and addition operations according to the matrix multiplication algorithm. Finally, it stores the result back into memory.

   Here's a summary of the steps:

   - Load the addresses of matrix_X and matrix_Y into registers $a0 and $a1.
   - Load the values of matrix_X into registers $t0, $t1, $t2, and $t3.
   - Load the values of matrix_Y into registers $t4, $t5, $t6, and $t7.
   - Perform multiplication and addition operations to calculate each element of the resulting matrix.
   - Store the result back into memory at the address specified by the result label.
   - Exit the program.

   C Code

```c
#include <stdio.h>

int main() {
    int matrix_X[2][2] = {{1, 2}, {3, 4}};
    int matrix_Y[2][2] = {{5, 6}, {7, 8}};
    int result[2][2];

    // Perform matrix multiplication
    result[0][0] = matrix_X[0][0] * matrix_Y[0][0] + matrix_X[0][1] * matrix_Y[1][0];
    result[0][1] = matrix_X[0][0] * matrix_Y[0][1] + matrix_X[0][1] * matrix_Y[1][1];
    result[1][0] = matrix_X[1][0] * matrix_Y[0][0] + matrix_X[1][1] * matrix_Y[1][0];
    result[1][1] = matrix_X[1][0] * matrix_Y[0][1] + matrix_X[1][1] * matrix_Y[1][1];

    // Display the result
    printf("Result:\n");
    printf("%d %d\n", result[0][0], result[0][1]);
    printf("%d %d\n", result[1][0], result[1][1]);

    return 0;
}
```

   Machine Code

```
00111100000000010001000000000001
00110100001001000000000000000000
00111100000000010001000000000001
00110100001001010000000000010000
10001100100010000000000000000000
10001100100010010000000000000100
10001100100010100000000000001000
10001100100010110000000000001100
10001100101011000000000000000000
10001100101011010000000000000100
10001100101011100000000000001000
10001100101011110000000000001100
01110001000011001000000000000010
01110001001011101000100000000010
00000010001100001000000000100000
01110001000011011001000000000010
01110001001011111001100000000010
00000010011100101001000000100000
01110001010011001010000000000010
01110001011011101010100000000010
00000010101101001010000000100000
01110001010011011011000000000010
01110001011011111011100000000010
00000010111110110101100000100000
00111100000000010001000000000001
10101100001100000000000000100000
00111100000000010001000000000001
10101100001100100000000000100100
00111100000000010001000000000001
10101100001101000000000000101000
00111100000000010001000000000001
10101100001101100000000000101100
00100100000000100000000000001010
00000000000000000000000000001100
```

Data Memory

```
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000000001000
00000000000000000000000000000000
```

Result

```
PS E:\IIITB\Sem 2\Comp Arch\PROJECT 2\MIPS Non-pipelined Project\MIPS Non-pipelined Project> python -u "e:\IIITB\Sem 2\Comp Arch\PROJECT 2\MIPS Non-pipelin
oject\MIPS Non-pipelined Project\non-pipelined-processor.py"
Enter choice from Matrix Multiplication :1 , Fibbonacci :2 , Array Copying :3 ::  1
Enter 8 elements , 4 per matrix with spaces : 1 2 3 4 5 6 7 8
19
22
43
50
```

2. Fibonacci

Initialization:

- $t0 is loaded with the value 8, representing the total number of Fibonacci numbers to calculate.
- $t1 is initialized to 0, representing the 0th Fibonacci number (Fib(0)).
- $t2 is initialized to 1, representing the 1st Fibonacci number (Fib(1)).
- $t3 is initialized to 0, representing a counter for the loop.

Fibonacci Calculation Loop:

- Inside the loop labeled fibonacci_loop, the next Fibonacci number is calculated iteratively.
- $t4 is set to the sum of the current Fibonacci numbers $t1 and $t2.
- $t1 is then updated to hold the value of the previous $t2 (Fib(n-1)).
- $t2 is updated to hold the value of the calculated Fibonacci number $t4 (Fib(n)).
- The loop counter $t3 is incremented by 1.
- The loop continues until the counter $t3 reaches the value stored in $t0.
- Exit:

Once the loop finishes executing, the program exits by loading the exit syscall code into $v0 and executing the syscall.

C Code

```c
#include <stdio.h>

int main() {
    int n = 8; // Number of Fibonacci numbers to generate
    int x = 0, y = 1, t4, counter = 0;

    // Output the first Fibonacci number (0)
    printf("%d ", x);

    // Output the second Fibonacci number (1)
    printf("%d ", y);

    counter += 2; // Count the first two Fibonacci numbers

    // Calculate and output the remaining Fibonacci numbers
    while (counter < n) {
        t4 = x + y;
        x = y;
        y = t4;
        printf("%d ", y);
        counter++;
    }

    return 0;
}
```

Machine Code

```
00100100000010000000000000001000
00100100000010010000000000000000
00100100000010100000000000000001
00100100000010110000000000000000
00000001001010100110000000100000
00000000000010100100100000100000
00000000000011000101000000100000
00100001011010110000000000000001
00010101011010001111111111111011
00100100000000100000000000001010
00000000000000000000000000001100
```

Result

```
PS E:\IIITB\Sem 2\Comp Arch\PROJECT 2\MIPS Non-pipelined Project\MIPS Non-pipelined Project> python -u "e:\IIITB\Sem 2\Comp Arch\PROJECT 2\MIPS Non-pipelined Pr
oject\MIPS Non-pipelined Project\non-pipelined-processor.py"
Enter choice from Matrix Multiplication :1 , Fibbonacci :2 , Array Copying :3 ::  2
Enter which digit fibbonacci: 8
21
```

3. Array Insertion
   - It loads the base address of the source array (source) and the target array (target) into registers $t2 and $t3, respectively.

- It loads the size of the array (size) into register $t0.
- It initializes a loop counter ($t1) to 0.
- Inside the loop:
- It loads an element from the source array into register $t4.
- It stores the loaded element into the target array.
- It increments the loop counter.
- It moves to the next element in both the source and target arrays by incrementing the respective base addresses.
- It jumps back to the start of the loop.
- When the loop counter reaches the size of the array, the program jumps to the end and exits.

C Code

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int source[] = {1, 2, 3, 4, 5};
    int size = 5;
    int target[5][4]; // 5x4 target array

    // Copy elements from source to target
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < 4; j++) {
            target[i][j] = source[i];
        }
    }

    // Output the target array
    printf("Target array:\n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%d ", target[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Machine Code

```
00111100000000001000100000000001
00110100001010100000000000000000
00111100000000001000100000000001
00110100001010110000000000010100
00111100000000001000100000000001
10001100001010000000000000101000
00100000000010010000000000000000
00010001001010000000000000000110
10001101010011000000000000000000
10101101011011000000000000000000
00100001001010010000000000000001
00100001010010100000000000000100
00100001011010110000000000000100
00001000000100000000000000000111
```

Data Memory

```
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000101
```

Result

```
PS E:\IIITB\Sem 2\Comp Arch\PROJECT 2\MIPS Non-pipelined Project\MIPS Non-pipelined Project> python -u "e:\IIITB\Sem 2\Comp Arch\PROJECT 2\MIPS Non-pipelined Pr
oject\MIPS Non-pipelined Project\non-pipelined-processor.py"
Enter choice from Matrix Multiplication :1 , Fibbonacci :2 , Array Copying :3 ::  3
1
2
3
4
5
```