COMPUTER ARCHITECTURE PROJECT

MIPS PROCESSOR

NIPUN VERMA (IMT2023591),
AYUSH MISHRA (IMT2023129), HARSH SINHA(IMT2023571)

1. Fibonacci-

The following MIPS assembly code calculates Fibonacci of x using an iterative method. To access the value of x we will use lw and sw to store the answer in memory, in between various registers are used to help in the functioning of the program. The time complexity of the code is O(n) and the space complexity is O(1).

C code-

```c
#include<stdio.h>
int main(){
    int n=15;
    int a=0;
    int b=1;
    int i=2;
    int ans=0;
    if(n==0){
        printf("%d\n",a);
    }
    else if(n==1){
        printf("%d\n",b);
    }
    else{
        while(i<n+1){
            ans=a+b;
            a=b;
            b=ans;
            i++;
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

MIPS Assembly-

```
.data
        x: .word 15
.text
        addi $t4,$t4,0
        lw $s0,0($t4)
        addi $t0,$zero,0
        addi $t1,$zero,1
        addi $t2,$zero,0
        addi $t3,$s0,-1
        beq $s0,$t0,loop1
        beq $s0,$t1,loop2
        bne $t3,$t2,loop

        loop1:
                sw $t0,4($t4)
                li $v0,10
                syscall
        loop2:
                addi $t0,$t1,0
                sw $t0,4($t4)
                li $v0,10
                syscall
        loop:
                add $s1,$t0,$t1
                addi $t0,$t1,0
                addi $t1,$s1,0
                addi $t2,$t2,1
                bne $t3,$t2,loop
                addi $t0,$s1,0
                sw $t0,4($t4)
                li $v0,10
                syscall
```

OUTPUT-

```
final val 610
```

2. Factorial-

This program calculates the factorial of a given integer using an iterative method. It initializes the data segment with a variable x initialized to 7. It then runs a loop accumulating the value of factorial in every iteration. At the end of the loop, sw stores the answer in the memory. The syscall command with the li command helps to terminate the program. The following code has a time complexity of O(n) and a space complexity of O(1).

C code-

```c
#include<stdio.h>
int main(){
    int n=7;
    int ans=1;
    if(n==0){
        printf("%d\n",1);
    }
    else if(n==1){
        printf("%d\n",1);
    }
    else{
        while(n>1){
            ans=ans*n;
            n--;
        }
        printf("%d\n",ans);
    }

    return 0;
}
```

MIPS Assembly Code-

```
.data
        x: .word 7
.text
        addi $t3,$t3,0
        lw $s0,0($t3)
        addi $t0,$zero,1
        addi $t1,$zero,1
        addi $t2,$zero,0
        beq $t2,$s0 ,exit

        bne $t2,$s0,loop
        loop:
                mul $t1,$t1,$t0
                addi $t0,$t0,1
                addi $t2,$t2,1
                bne $t2,$s0,loop
                sw $t1 4($t3)
                li $v0,10
                syscall
        exit:
                sw $t1 4($t3)
                li $v0,10
                syscall
```

OUTPUT-

```
final val 5040
```

3. Power

This function raises the power of the base i.e. x to n and stores the result in memory. This function iteratively raises x to the power n. The code works only if the power to which the base should be raised is non-negative.

Otherwise, it will give floating point value.

C Code-

```c
#include<stdio.h>
int main(){
    int x=2;
    int y=0;
    int ans=1;
    if(y==0){
        printf("%d\n",1);
    }
    else{
    while(y){
        ans=ans*x;
        y--;
    }
    printf("%d\n",ans);
    }
    return 0;
}
```

MIPS Assembly-

```
.data
        x: .word -2
        n: .word 3
.text
        addi $t0,$t0,0
        lw $s1,0($t0)
        lw $s2,4($t0)
        addi $s0,$zero,1
        addi $t1,$zero,0
        bne $t1,$s2,loop
        sw $s0,8($t0)
        li $v0,10
        syscall
        loop:
        mul $s0,$s0,$s1
        addi $t1,$t1,1
        bne $t1,$s2,loop
        sw $s0,8($t0)
        li $v0,10
        syscall
```

OUTPUT-

```
final val -8
```

Processor-

1. **Data Memory –**

   The memory file is created by using the .data file generated by the MARS assembler. The binary text is read line by line and the contents are stored in a dictionary where the key is the memory address and the value is the content. This resonates with MIPS as the dictionary also stores keeping in mind the byte-addressable memory feature of the MIPS processor.

2. **Instruction Memory-**

   The PC of instruction starts from the value 12288. The Instruction memory using the concept of byte addressable memory stores the instruction in a dictionary which has its keys as PC and the value of the dictionary as the instruction.

3. **Register File**-

   The register file stores the addresses and their values in a dictionary in which the key of dictionary is the 5 bit address of the register and the value of the dictionary is the value in register. The values of all registers are initialized to zero.

4. **Converter functions-**

   There are four converting functions two of which follow 2's complement form while other two do not follow 2's complement form. The functions include two decimal to binary converters and two binary to decimal converters.

5. **Splitter function-**

   This function takes in 32 bit instruction and divides the instruction into four 8 byte instructions and returns the list of these 4 instructions to simulate byte addressable memory in MIPS.

6. **Op generator-**
   This is a dictionary consisting of opcodes of the instructions as keys and instruction name as the value.

7. **Fetch Function-**

   This function takes the value of PC and calls the decode function after accessing instruction from the instruction memory.

8. **Decode Function-**

   This function when provoked calls the control function with the appropriate opcode to generate the control lines. It also performs the sign extension. It then calls the execute function with appropriate arguments.

9. **Execute Function-**
   This function using the control signals (like ALU OP) calls the appropriate function to execute the instruction.

10. Control Function-

This function sets all the control signals appropriately for different types of instruction formats. This function is always provoked by the decode function.

11. Memory Function-

This function accesses the memory files only if required.

12. Writeback Function-

This function updates the contents of the register file if required.