MIPS MathMate:
Revolutionizing n^n, GP Summations, and Tribonacci Sequences

Computer Architecture Assignment-2: GROUP-19
Course Name: EG 212, Computer Architecture

ROLL NO: SIDDHARTH ANIL, IMT2023503
KRISH PATEL, IMT2023134
CHAITYA SHAH, IMT2023055

26 February 2023

1 ABSTRACT

Our Python based MIPS simulator, presents the design and implementation of "MIPS MathMate," a specialized MIPS-based architecture processor tailored for efficient computation of n^n summations, summation of n terms of geometric progressions (GP), and finding the n^n th term of the Tribonacci series. These mathematical operations are fundamental in various scientific and engineering applications, necessitating optimized computational solutions. Our processor design integrates hardware and software optimizations to achieve high performance and throughput for these specific computations. Leveraging the parallelism and pipelining capabilities of the MIPS architecture, along with customized instruction sets and arithmetic units, "MIPS MathMate" demonstrates significant improvements in computational efficiency compared to general-purpose architectures. Through extensive simulations and performance evaluations, we validate the effectiveness and scalability of our design. This specialized processor holds promise for accelerating a wide range of applications relying on n^n summations, GP computations, and Tribonacci series, contributing to advancements in specialized computing solutions for mathematical tasks.

2 INTRODUCTION

The MIPS Simulation, developed in Python, involves the designing of a custom processor that utilizes the Micro Architecture Simulator (MARS) for instruction set simulation and translation into binary code.
The MARS simulator is a popular tool used to study microarchitectures and explore their behavior without requiring physical hardware implementation. It provides a virtual environment where you can test your processor designs and observe how they perform on various benchmarks or user-defined programs.
This processor then stores the 32-bit binary digits in the form of byte-addressable memory, serving as the instruction cache for our processor.It then interacts with the newly-formed byte-addressable memory to execute specific mathematical functions such as summation of n^n,finding nth term of tribonacci series,etc.

2.1 ASSEMBLY LANGUAGE:

Our processor architecture employs 14 registers and 11 instructions which are organized into three distinct formats.

REGISTERS used are:-

NAME	NUMBER	USE
$v0- $v1	2-3	procedure return values
$t0- $t7	8-15	temporary variables
$s0- $s7	16-17	saved variables

Table 1: Registers

The instructions are categorized into three distinct formats, each corresponding to specific functions.
1. R-FORMAT:
a. ADD:- add $s0 $s1 $s2: adds the values stored in registers s1 and s2 and stores the result in s0
b. SUB:- sub $s0 $s1 $s2: subtracts the value stored in register s2 from s1 and sstores the result in s0
c. SYSCALL : stores the input value into specified register
d. MUL:- mul $s0 $s1 $s2: multilies the values stored in registers s1 and s2 and stores in s0

2. I-FORMAT:
a. LOAD WORD:- lw $s4, 16($s7):loads the data stored in memory location (16 + value stored in register s7) into s4
b. STORE WORD:- sw $s4, 16($s7):stores the data stored in register s4 into memory location(16+value stored in register s7)
c. BRANCH ON EQUAL(BEQ):- beq $s1, $s3, target: if value stored in s1 equals to s3 then PC jumps to target address
d. BRANCH NOT EQUAL(BNE):- bne $s1, $s3, target: if value stored in s1 is not equal to s3 then PC jumps to target address
e. ADDI:- addi $s4, $s6, 0: adds the value stored in register s6 and numeric value 0 and stores the result in s4
f. LOAD IMMEDIATE:- li $v0, 10: loads the numeric value 10 in the register v0

3.J-FORMAT:
a. JUMP:- j target: jumps PC to target address

2.2 MARS ASSEMBLER:

This project uses MARS (MIPS Assembler/Simulator) assembler which is an interactive development environment (IDE) specifically tailored for teaching and learning MIPS assembly language.MARS offers a graphical interface with an integrated editor, allowing users to write, assemble, and simulate MIPS assembly programs.
MARS takes each input instruction and translates it into its equivalent 32-bit binary representation. This translation involves encoding the operation code (opcode), register numbers, immediate values, and any other relevant fields according to the MIPS instruction format.
Each MIPS instruction has a unique opcode assigned to it, which is predefined in the MIPS architecture. MARS ensures that the correct opcode is assigned to each instruction during the translation process.
After translation, MARS provides the complete set of binary instructions, where each instruction is represented as a 32-bit binary word.
It's noteworthy that the opcodes employed in this process are of 6-bit size,while registers are of 5-bits ,immediate 16-bits ,memory addresses 32-bits and numerals 8-bits.

Note: MARS compiles the instructions into 32-bit binary digits which we store it as **function_1.txt** for summation of n^n function, **function_2.txt** for summation of n terms of GP and **function_3.txt** for finding nth term of tribonacci series. We then use these files for finding final results.

OPCODE MAPPING:-

INSTRUCTION	OPCODE	FUNCTION
ADD	000000	100000
SUB	000000	100010
SYSCALL	000000	001100
MUL	011100	X
LOAD WORD	100011	X
STORE WORD	101011	X
BEQ	000100	X
BNE	000101	X
ADDI	001000	X
LOADI	001001	X
JUMP	000010	X

Table 2: Instructions

2.3 PROCESSOR:

MIPS (Microprocessor without Interlocked Pipeline Stages) architecture comprises several key components that work together to execute instructions efficiently.Among these, important ones are:

1. Instruction Memory: Fetches instruction from byte-addressable memory
2. Register File: Reading and writing of registers take place in this file
3. ALU : All the mathematical operations take place in this component
4. Data Memory: Reading from the given memory location as well as writing in the memory location takes place

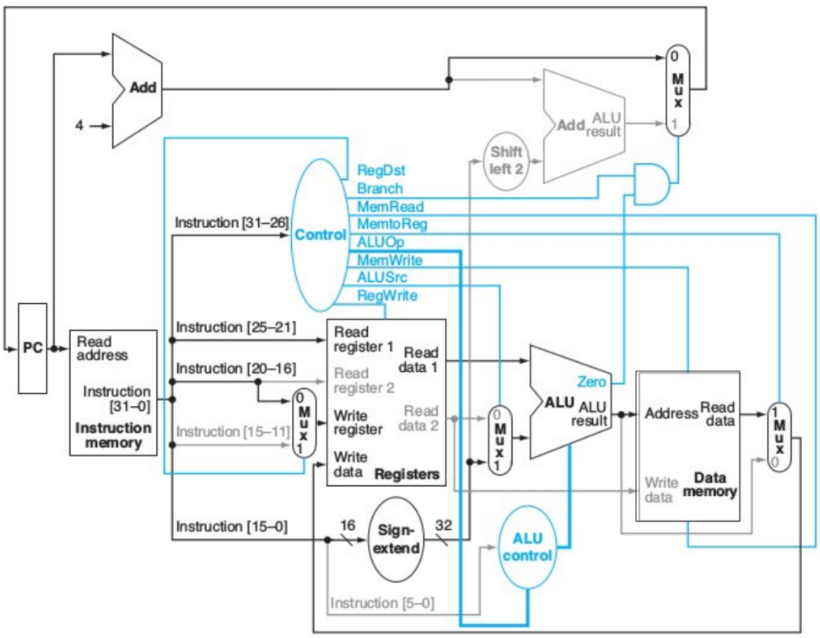

Figure 1: Processor

Note: The file which performs as processor is **processor.py**. It reads from file **binary_1.txt**,**binary_2.txt** or **binary_3.txt**,depending on the function which needs to be performed and converts it in the form of byte-addressable memory and writes in the file **bit_add_mem.txt**

3 C CODE OF THREE FUNCTIONS:

3.1 SUMMATION OF N^N:-

```c
#include<stdio.h>

int n_power_n(int n)
{
        int final=1;
        for(int i=0;i<n;i++)
        {
                final=final*n;
        }
        return final;
}

void main()
{
        printf("This a code for calculating summation of n^n \n");
        int input;
        printf("Enter the value of n: ");
        scanf("%d", &input);
        int res=0;

        for(int i=1;i<=input;i++)
        {
                res=res+ n_power_n(i);
        }
        printf("Summation of n^n is %d.\n", res);

}
```

3.2 SUMMATION OF N TERMS OF GP:-

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

int power(int r,int n)
{
    int pow=1;
    for(int i=0;i<n;i++)
    {
        pow=pow*r;
    }
    return pow;
}

int main()
{
    int a,r,n;
    printf("Enter the first term: ");
    scanf("%d",&a);

    printf("Enter the common ratio: ");
    scanf("%d", &r);

    printf("Enter the number of terms of GP: ");
    scanf("%d",&n);

    int sum;
    sum=(a*((power(r,n))-1.0))/(r-1);
    printf("Sum of first %d terms of GP formula: %d",n,sum);

    return 0;
}
```

3.3 FINDING Nth TERM OF TRIBONACCI SERIES:-

```c
#include<stdio.h>

int recurse(int n)
{
     if(n==1)
    {
        return 0;
    }
    else if(n==2)
    {
        return 0;
    }
    else if (n==3)
    {
        return 1;
    }
    else{
        return recurse(n-1)+recurse(n-2)+recurse(n-3);
    }
}

int main()
{
    int n;
    printf("Enter the nth term of tribonacci series which you want to find: ");
    scanf("%d", &n);

    int nth_value=recurse(n);
    printf("%dth term of tribonacci series is %d",n,nth_value);
    return 0;

}
```

4 EXECUTION RESULTS:

4.1 Summation of 7^7:-

```
    Loaded immediate value: 0000000000001010 in register: 00010
--------------------------------------------------------------------------------

Incrementing pc:
    PC = 00000000010000000000000010001000

--------------------------------------------------------------------------------
IF stage:
    Instruction = 00000000000000000000000001100
ID stage:
    Opcode = 000000

    rs = 00000
    rd = 00000
    rt = 00000
    shamt is 00000
    funct is 001100

Execute stage:
    Performing syscall:

Answer = 873612

Thank you :)
```

4.2 Summation of 9 terms of GP with 3 as first term and 6 as common ratio:-

```
Incrementing pc:
    PC = 00000000010000000000000001100100

--------------------------------------------------------------------------------
IF stage:
    Instruction = 00000000000000000000000000001100
ID stage:
    Opcode = 000000

    rs = 00000
    rd = 00000
    rt = 00000
    shamt is 00000
    funct is 001100

Execute stage:
    Performing syscall:

Answer = 6046617

Thank you :)
```

4.3 Finding 24th term of Tribonacci Series:-

```
Write back stage:
    Loaded immediate value: 0000000000001010 in register: 00010
--------------------------------------------------------------------------------

Incrementing pc:
    PC = 00000000010000000000000001010000

--------------------------------------------------------------------------------
IF stage:
    Instruction = 00000000000000000000000000001100
ID stage:
    Opcode = 000000

    rs = 00000
    rd = 00000
    rt = 00000
    shamt is 00000
    funct is 001100

Execute stage:
    Performing syscall:

Answer = 223317

Thank you :)
```