CA Project Report

Topic: Mips Processor Design

by
IMT2023552: M S Dheeraj Murthy
IMT2023008: Mathew Joseph
IMT2023046: Priyanshu Pattnaik

Introduction:

The processor is designed to simulate the working of a typical MIPS processor following the various phases such as Instruction Fetch, Instruction Decode, Execute, Memory Access and Writeback. It takes the machine code given to it by the Mars simulator and processes it. Similar to the MIPS architecture, the data and the instruction memory are stored separately. Similarly, the registers are stored in a separate Register File. Each instruction is fetched from the instruction memory and passed into the various stages of the MIPS processor where the different components handle the data appropriately.

Implemented Functions:

We use the following functions to implement 3 programs which we will discuss later.

```
// R-Format:
SLL rd,rt,rd         000000 rs     rt     rd     sa     000000
SRL rd,rt,sa         000000 rs     rt     rd     sa     000010
SLV rd,rt,rs         000000 rs     rt     rd     00000  000100
SRLV rd,rt,rs        000000 rs     rt     rd     00000  000110
JR rs                000000 rs     00000  00000  00000  001000
SYSCALL              000000 00000  00000  00000  00000  001100
MFHI rd              000000 00000  00000  rd     00000  010000
MTHI rs              000000 rs     00000  00000  00000  010001
MFLO rd              000000 00000  00000  00000  rd     010010
MTLO rs              000000 rs     00000  00000  00000  010011
MULT rs,rt           000000 rs     rt     00000  00000  011000
DIV rs,rt            000000 rs     rt     00000  00000  011010
ADD rd,rs,rt         000000 rs     rt     rd     00000  100000
SUB rd,rs,rt         000000 rs     rt     rd     00000  100010
AND rd,rs,rt         000000 rs     rt     rd     00000  100100
OR rd,rs,rt          000000 rs     rt     rd     00000  100101
XOR rd,rs,rt         000000 rs     rt     rd     00000  100110
NOR rd,rs,rt         000000 rs     rt     rd     00000  100111
SLT rd,rs,rt         000000 rs     rt     rd     00000  101010
```

```
// I-Format:
BLTZ rs,offset       000001 rs     00000  offset
BGEZ rs,offset       000001 rs     00001  offset
BEQ rs,rt,offset     000100 rs     rt     offset
BNE rs,rt,offset     000101 rs     rt     offset
BLEZ rs,offset       000110 rs     00000  offset
BGTZ rs,offset       000111 rs     00000  offset
ADDI rt,rs,imm       001000 rs     rt     imm
SLTI rt,rs,imm       001010 rs     rt     imm
ANDI rt,rs,imm       001100 rs     rt     imm
ORI rt,rs,imm        001101 rs     rt     imm
XORI rt,rs,imm       001110 rs     rt     imm
LUI rt,imm           001111 rs     rt     imm
LB rt,offset(rs)     100000 rs     rt     offset
LH rt,offset(rs)     100001 rs     rt     offset
LW rt,offset(rs)     100011 rs     rt     offset
SB rt,offset(rs)     101000 rs     rt     offset
SH rt,offset(rs)     101001 rs     rt     offset
SW rt,offset(rs)     101011 rs     rt     offset

// J-Format:
J target             000010 target
```

Source: https://opencores.org/projects/plasma/opcodes

Assembly Program 1: Nth Fibonacci number

This program takes a number N, and then outputs the Nth Fibonacci number. This is implemented using a while loop where we store n-1 in A and n-2 in B in two variables and add them up to get n. Then, we put n-1 in B and put the value of n into A. This process continues till we get the desired number.

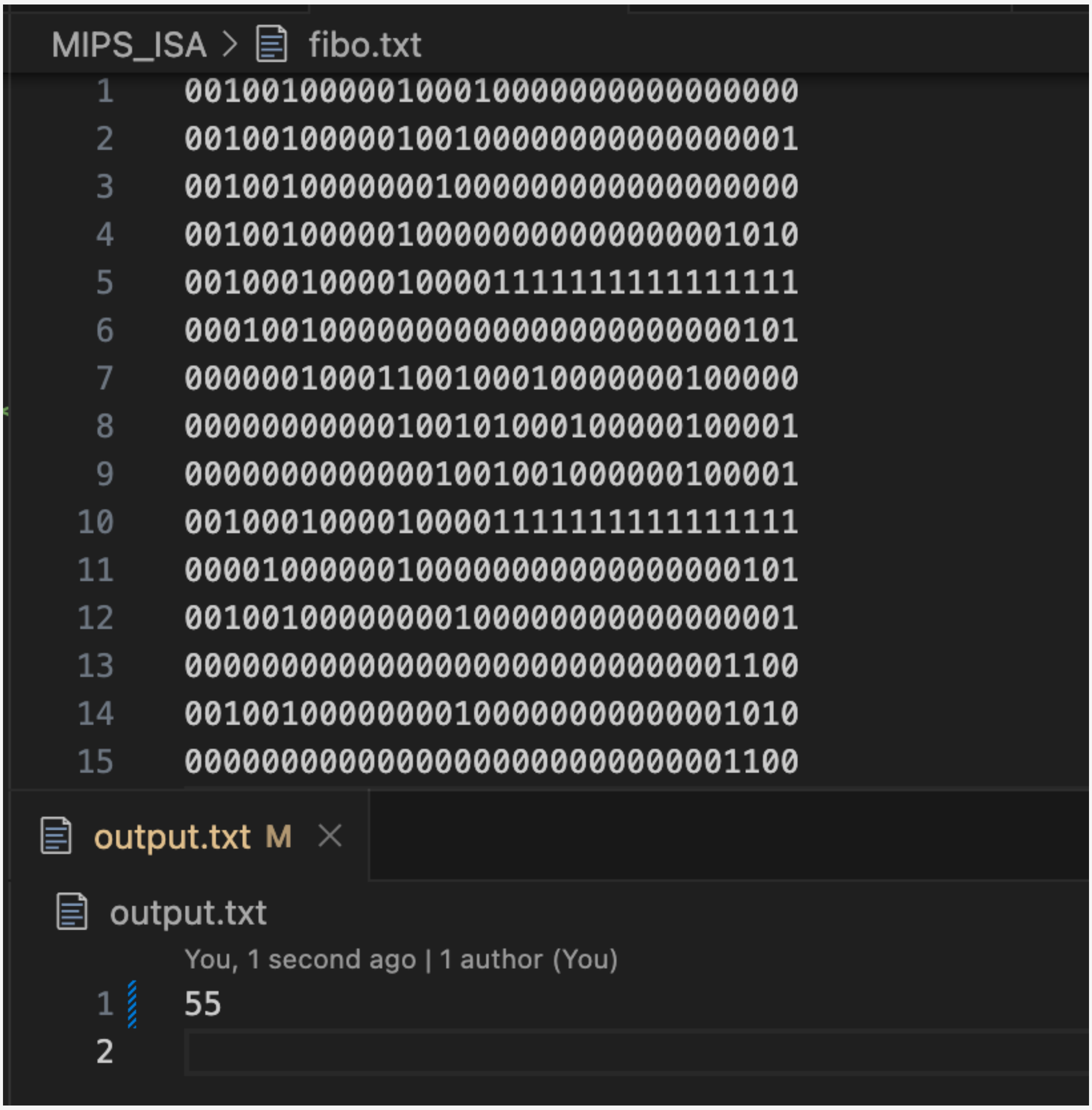
Here is the code for the above:

```
.text
.globl main

main:
     li $s1,0 #this is a
     li $s2,1 #this is b
     li $a0,0 #this is where the answer is stored
     li $s0,10 #this is n and the input
     addi $s0,$s0,-1
target: beq $s0,$zero,end
     add $a0,$s1,$s2
     move $s1,$s2
     move $s2,$a0
     addi $s0,$s0,-1
     j target
end: li $v0,1
     syscall
     li $v0,10
     syscall
```

Equivalent C code:

```
int fib(int n) {
    int a = 0, b = 1;
    int ans = 0;
    while (n--) {
        n = a + b;
        a = b;
        b = n;
    }
    return n;
}
```

Output Snippet:



Assemble Program 2: Factorial of N

This program computes the factorial of an integer n and displays it in the output window. This is implemented by iterating from 1 to n while multiplying the iterator with an answer variable and storing the product in the answer variable.

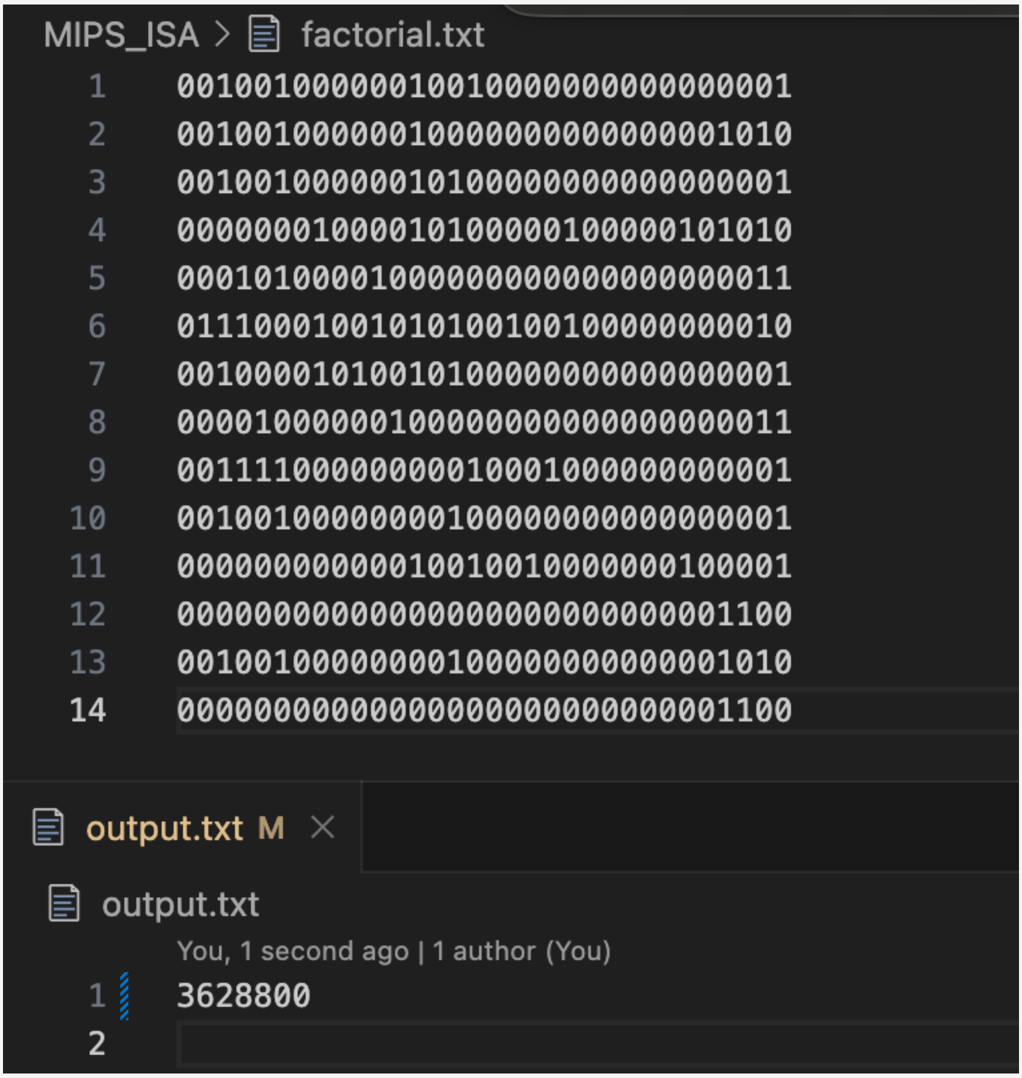
Here is the code for the above:

```
.data
result: .word 0             #variable to store the result

.text
.globl main

main:
    # initializing result to 1
    li $t1, 1                 # load 1 into $t1 (result)
    li $t0, 10                #calculating factorial of 10
    li $t2, 1                 # load 1 into $t2 (iterator)

loop:
    bgt $t2, $t0, end_loop  # if iterator > input number, jump
to end_loop
    mul $t1, $t1, $t2       # result = result * iterator
    addi $t2, $t2, 1        # iterator++
    j loop                       # loop repeat

end_loop:
    li $v0, 1                 # system calls for printing
integer
    move $a0, $t1           # it loads the factorial result
    syscall

    li $v0, 10              # calls for exit
    syscall
```

Equivalent C code:

```c
int factorial(int n)
{
    int ans = 1;
    while (n--)
        ans *= n;
    return ans;
}
```

Output Snippet:



Assembly Program 3: checking if n is prime

This program computes the primality of an integer n and displays it in the output window. This is implemented by iterating from 2 to n and checking each value if n is divisible by it. The program then returns the primality of N in the form of 0(false) or 1(true). Here is the code for the above:

```
.data
result: .word 0                    # variable to store the
result

.text
.globl main

main:
    li $t0, 131                    # load n from storage
    li $t1, 0                      # initialize quotient to 0
    li $t2, 2                      # initialize divisor to
2
#loop to check if n is divisible by any number from 2 to n-1
loop:
    bge $t2, $t0, store_result     # if divisor >= n, jump to
store_result
    div $t0, $t2                   # divide n by divisor
    mfhi $t3                       # getting the remainder
    beq $t3, $zero, not_prime      # if remainder is 0, jump
to not_prime
    addi $t2, $t2, 1               # increment divisor
    j loop                         # jump back to loop
not_prime:
    # Store 0 in result (not prime)
    li $t4, 0                      # load 0 into $t4
    sw $t4, result                 # store 0 in result
    j end                          # jump to end
store_result:
    # Store 1 in result (prime)
    li $t5, 1                      # load 1 into $t5
    sw $t5, result                 # store 1 in result
end:
    # let's print the result
    move $a0,$t5
    li $v0,1
    syscall
    li $v0, 10                     # system calls for exit
    syscall
```

Equivalent C code:

```c
int prime(int n)
{
    int i =2;
    while (i < n)
        if (n%(i++)==0) return false;
    return true;
}
```

Output Snippet: