Assignment 2: MIPS Processor Design

Team Members:

1) Aaryan Rajesh Antala (IMT2023548)
2) Parsania Ramya Parulbhai (IMT2023015)
3) Velidanda Krishna Sai (IMT2023094)

Explanation of the Programs:

We have chosen to implement the following programs in the MIPS processor :

1) Factorial Calculation
2) Nth Fibonacci Number
3) Power Calculation

We have included the following files as part of the assignment:

1) Processor.py
2) Listing.py
3) Assembly.asm (for all three programs)
4) MARS-generated Machine Codes (.txt files, for all three assembly programs)
5) C implementation of all the programs (Fibonacci.c, Power.c, Factorial.c)

I. Processor:

The Python code is a basic simulation of the MIPS Processor. It includes a class named MIPS which includes the following stages of the MIPS Processor - Instruction Fetch, Instruction Decode, Execute, Memory Access, and Memory Writeback. Upon initialisation, it sets up memory and registers. The instruction and data are stored in the memory in word-addressable form. This means the 32-bit data is split into 4 bytes and stored in four contiguous locations. This is followed by fetching the instructions and decoding them to determine their formats and components. Control signals are then determined based on the instruction type, directing subsequent stages such as ALU operations and memory accesses. The processor supports all three formats - R, J, and I. The processor includes the implementation of the following instructions: JUMP, BEQ, BNE, LOAD, STOR, ADD, MUL, and SUB.

a) Memory:

The total memory is of size 10^7 bytes, common for both instruction memory and data memory. Instructions are stored in the memory starting from location 0.

To get the values required in the programs, we stored values from 0 to 1000 in the memory, at the time of initialization of the machine. They are stored starting from address 1000 in the memory, each number stored in 4 bytes in 4 continuous locations. So, locations 1000-1003 contain 0, 1004-1007 contain 1, and so on.

b) Register File:

There is a register file that contains 32 registers, each 32-bit wide, indexed from 0 to 31, as per MIPS Architecture. The values in the registers are used during the execution of instructions to hold values that will be needed.

c) Format of stored data and instructions:

We implemented two functions to convert decimal values into binary values in 2's complement form and back. The "decimal_to_twos_complement()" function takes a decimal value and a length and converts the decimal value into 2's complement binary string of the specified length. The "twos_complement_to_decimal()" function takes a binary string and converts it into decimal form.

Thus, the program iterates through the instruction memory, executing them until the program counter reaches the end of the set of instructions provided.

II. Listing

This program converts the Machine codes generated by the MARS Assembler into a list which can used by Processor.py.

III. Assembly.asm Files

The three asm files are the assembly code for our processor which includes the instructions to be followed by the processor to execute the program.

IV. Mars-Generated Machine Codes

The three machine codes are generated by the MARS Assembler using the assembly.asm files. The machine codes are stored in three respective .txt files. These machine codes are then used by listing.py to convert the Instructions into a list, which is then used by the Processor to execute the program.

V. C Implementation of the Programs

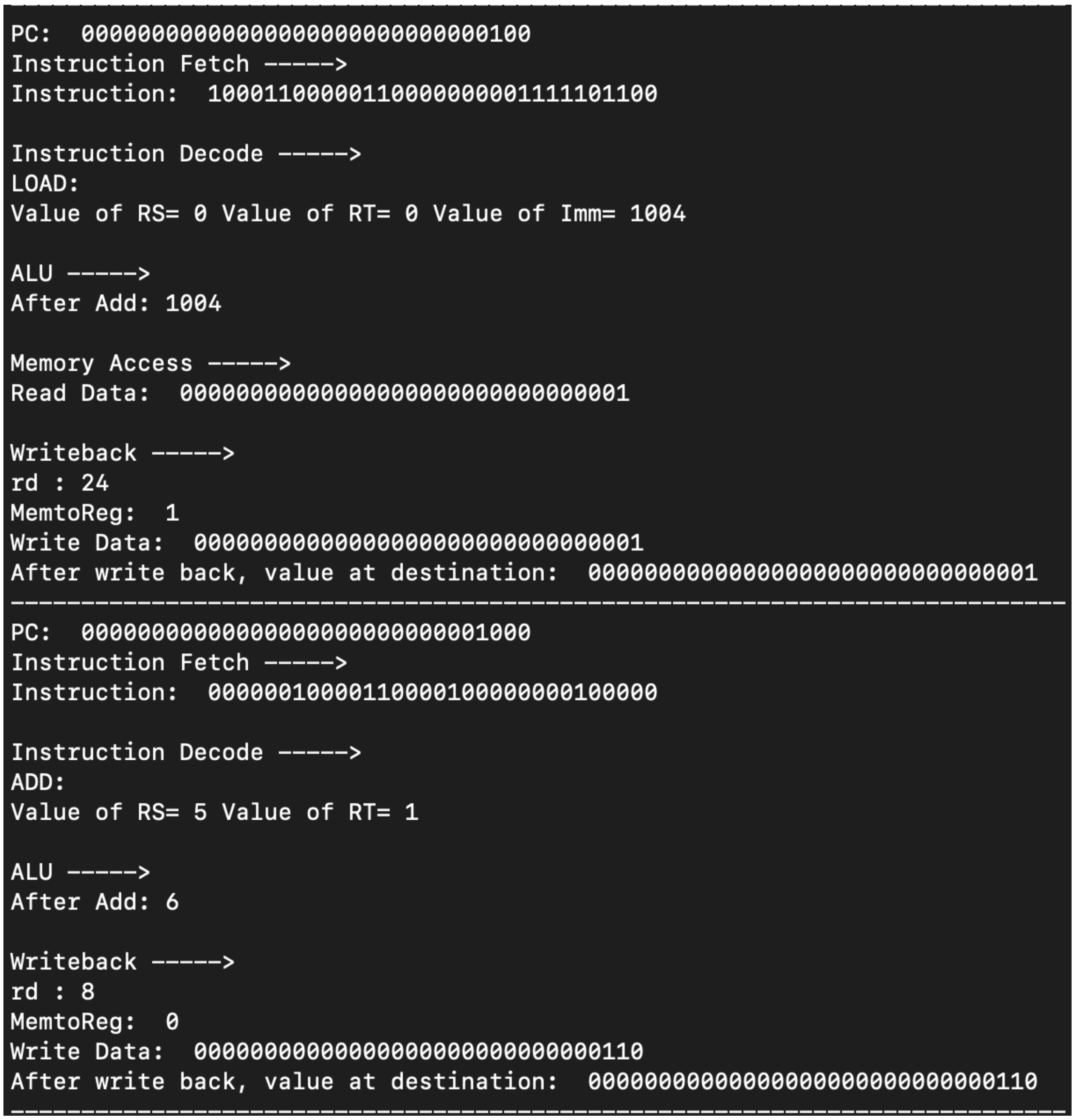
As a reference to the programs run on the MIPS Processor we have also included the C implementation of the following programs: Factorial Calculation, Power of a Number, and Nth Fibonacci Number.

Terminal Output:

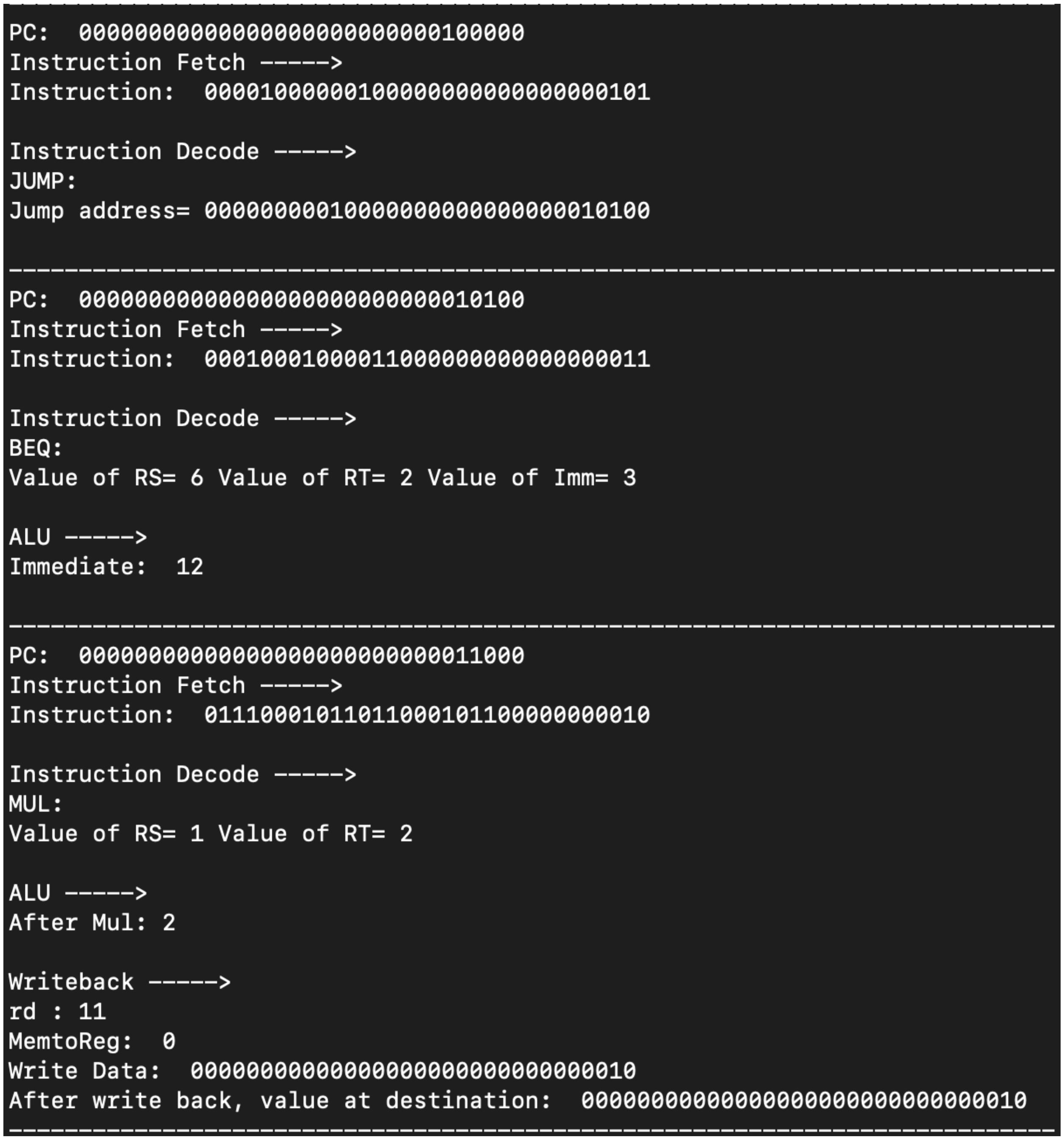
We have displayed a few screenshots of the output generated by the processor :

1) Factorial Calculation:

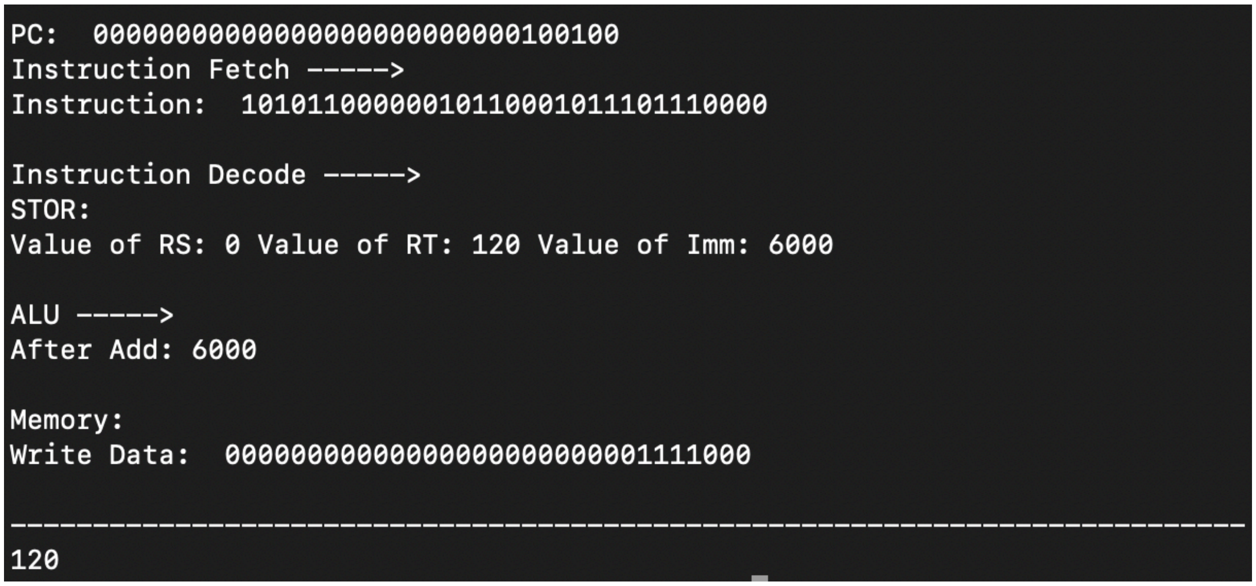
The following screenshots are for the calculation of the factorial of 5 which is 120.



```
---------------------------------------------------------------------------
PC:  00000000000000000000000000000100
Instruction Fetch ----->
Instruction:  10001100000110000000001111101100

Instruction Decode ----->
LOAD:
Value of RS= 0 Value of RT= 0 Value of Imm= 1004

ALU ----->
After Add: 1004

Memory Access ----->
Read Data:  00000000000000000000000000000001

Writeback ----->
rd : 24
MemtoReg:  1
Write Data:  00000000000000000000000000000001
After write back, value at destination:  00000000000000000000000000000001
---------------------------------------------------------------------------
PC:  00000000000000000000000000001000
Instruction Fetch ----->
Instruction:  00000010000110000100000000100000

Instruction Decode ----->
ADD:
Value of RS= 5 Value of RT= 1

ALU ----->
After Add: 6

Writeback ----->
rd : 8
MemtoReg:  0
Write Data:  00000000000000000000000000000110
After write back, value at destination:  00000000000000000000000000000110
---------------------------------------------------------------------------
```



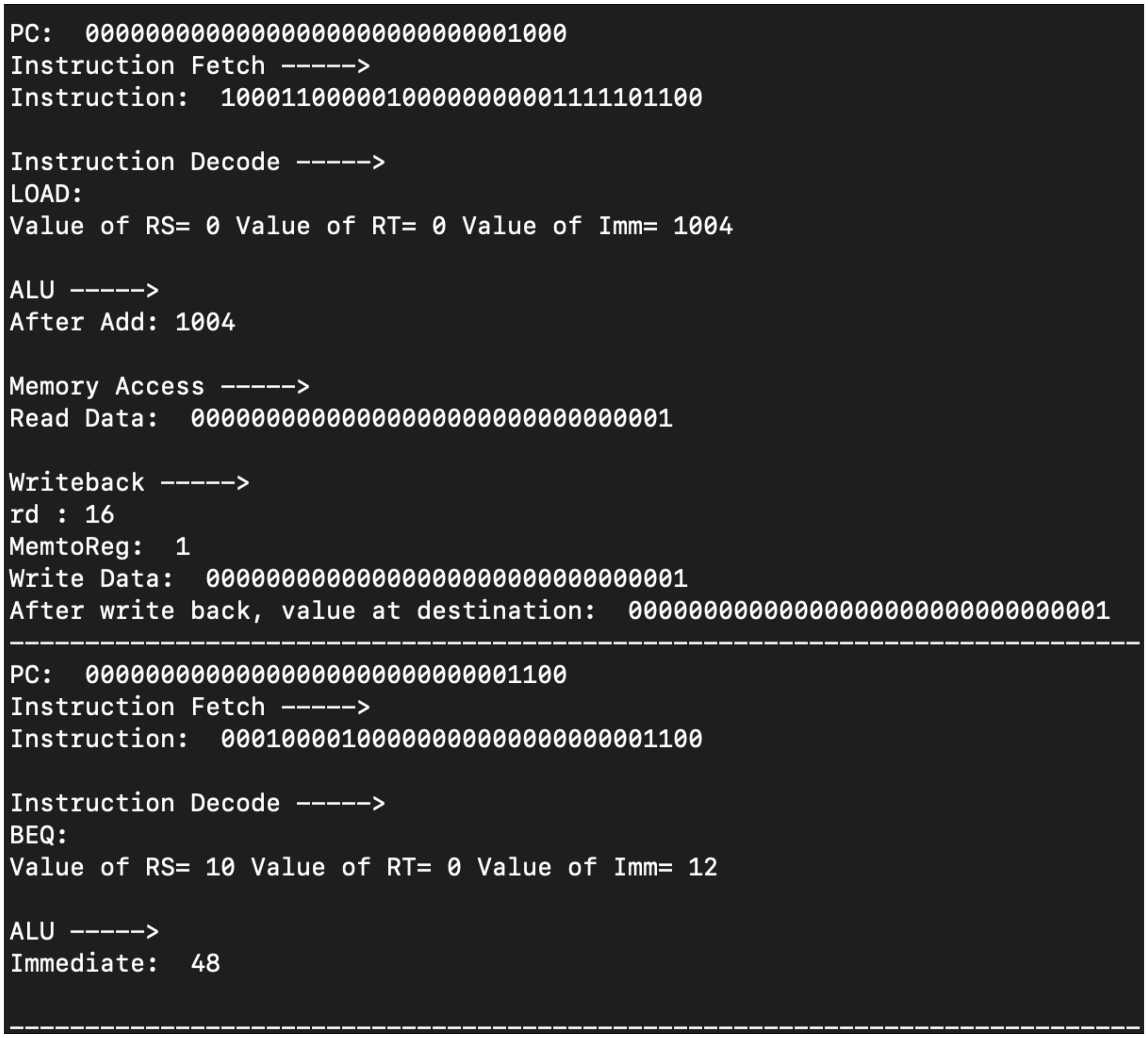
```
PC:  00000000000000000000000000100000
Instruction Fetch ----->
Instruction:  00001000000100000000000000000101

Instruction Decode ----->
JUMP:
Jump address= 00000000010000000000000000010100

-----------------------------------------------------------------------------
PC:  00000000000000000000000000010100
Instruction Fetch ----->
Instruction:  00010001000011000000000000000011

Instruction Decode ----->
BEQ:
Value of RS= 6 Value of RT= 2 Value of Imm= 3

ALU ----->
Immediate:  12

-----------------------------------------------------------------------------
PC:  00000000000000000000000000011000
Instruction Fetch ----->
Instruction:  01110001011011000101100000000010

Instruction Decode ----->
MUL:
Value of RS= 1 Value of RT= 2

ALU ----->
After Mul: 2

Writeback ----->
rd : 11
MemtoReg:  0
Write Data:  00000000000000000000000000000010
After write back, value at destination:  00000000000000000000000000000010
-----------------------------------------------------------------------------
```



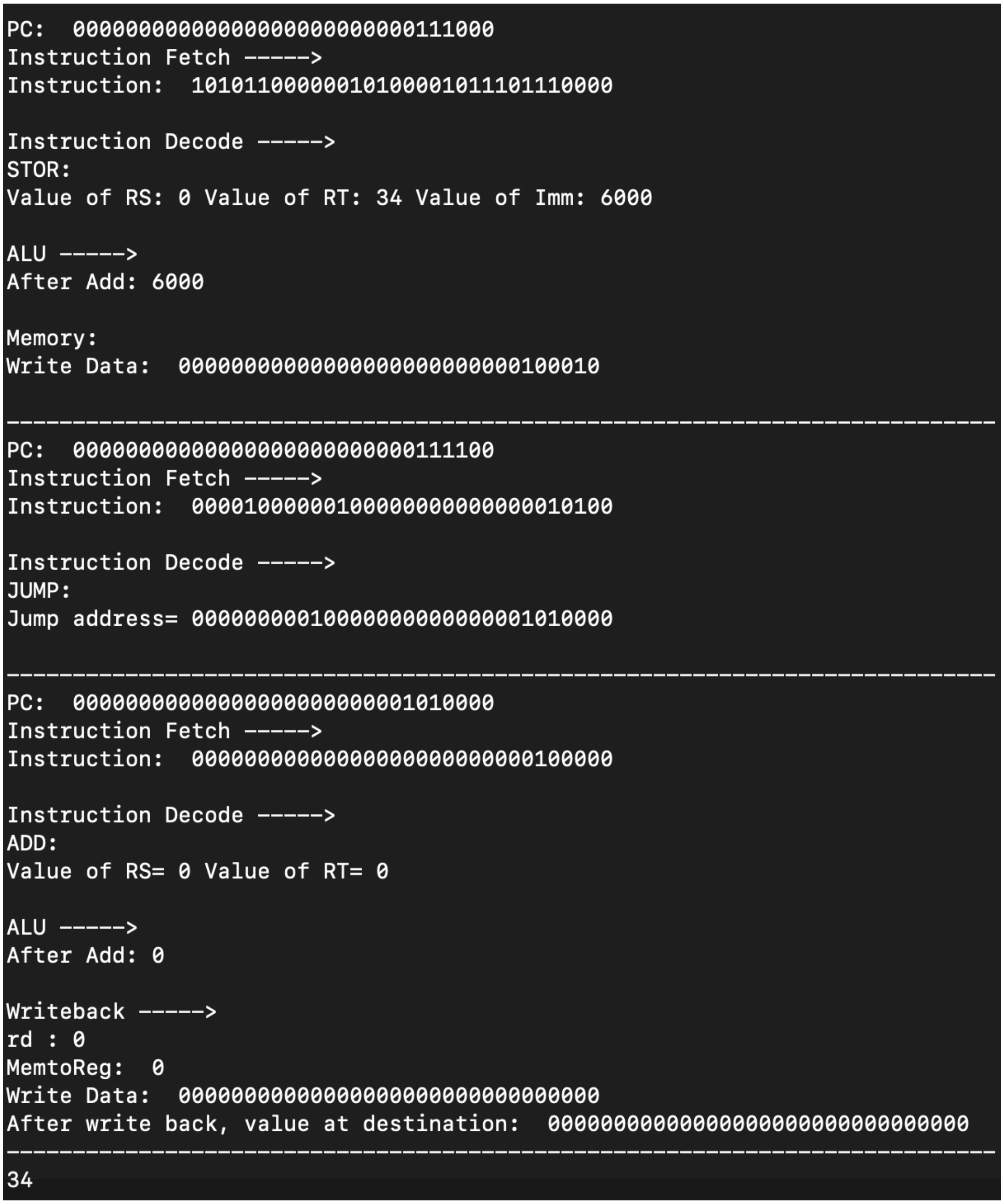
```
PC:  00000000000000000000000000100100
Instruction Fetch ----->
Instruction:  10101100000010110001011101110000

Instruction Decode ----->
STOR:
Value of RS: 0 Value of RT: 120 Value of Imm: 6000

ALU ----->
After Add: 6000

Memory:
Write Data:  00000000000000000000000001111000

---------------------------------------------------------------------------
120
```

2) Nth Fibonacci Number Calculation

The following screenshots are for the calculation of the 10th Fibonacci number which is 34:



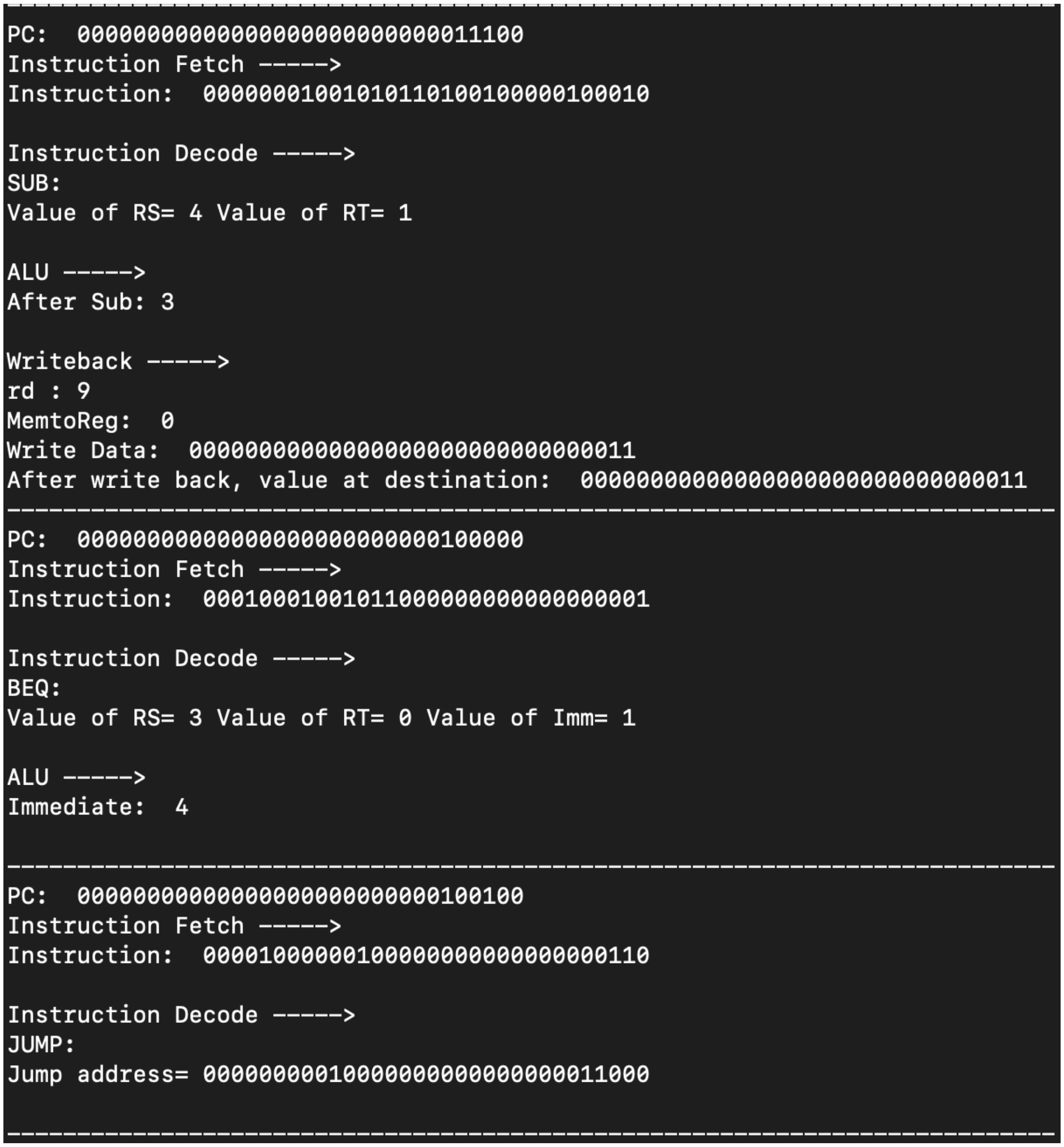
```
PC:  00000000000000000000000000001000
Instruction Fetch ----->
Instruction:  10001100000100000000001111101100

Instruction Decode ----->
LOAD:
Value of RS= 0 Value of RT= 0 Value of Imm= 1004

ALU ----->
After Add: 1004

Memory Access ----->
Read Data:  00000000000000000000000000000001

Writeback ----->
rd : 16
MemtoReg:  1
Write Data:  00000000000000000000000000000001
After write back, value at destination:  00000000000000000000000000000001
---------------------------------------------------------------------------
PC:  00000000000000000000000000001100
Instruction Fetch ----->
Instruction:  00010001000000000000000000001100

Instruction Decode ----->
BEQ:
Value of RS= 10 Value of RT= 0 Value of Imm= 12

ALU ----->
Immediate:  48

---------------------------------------------------------------------------
```



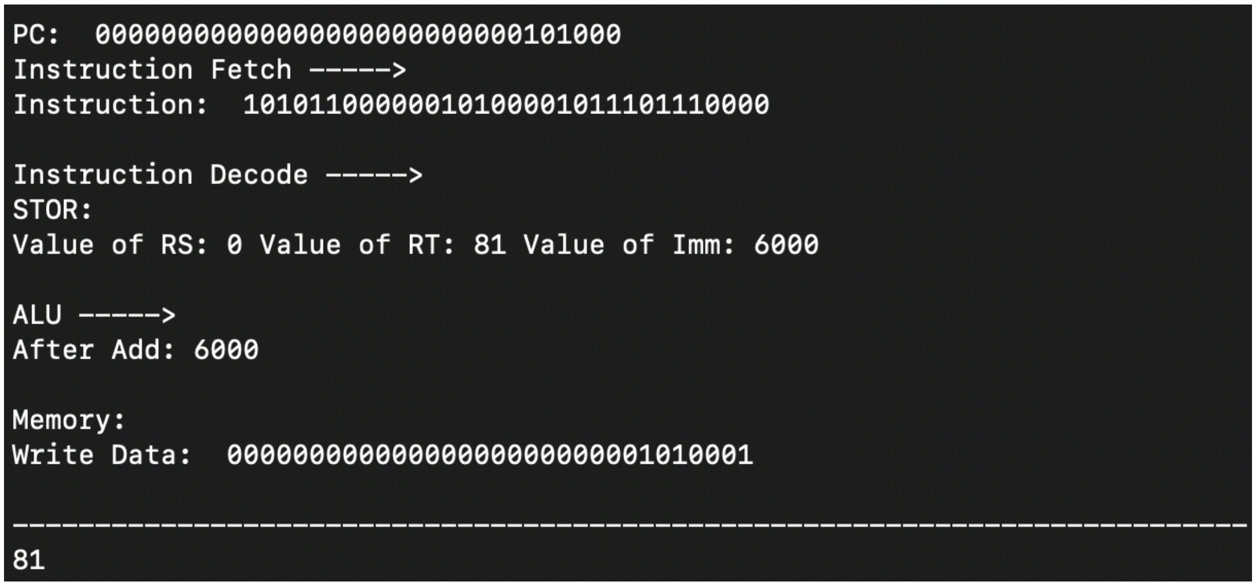
```
PC:  00000000000000000000000000111000
Instruction Fetch ----->
Instruction:  10101100000010100001011101110000

Instruction Decode ----->
STOR:
Value of RS: 0 Value of RT: 34 Value of Imm: 6000

ALU ----->
After Add: 6000

Memory:
Write Data:  00000000000000000000000000100010

---------------------------------------------------------------------------
PC:  00000000000000000000000000111100
Instruction Fetch ----->
Instruction:  00001000000100000000000000010100

Instruction Decode ----->
JUMP:
Jump address= 00000000010000000000000001010000

---------------------------------------------------------------------------
PC:  00000000000000000000000001010000
Instruction Fetch ----->
Instruction:  00000000000000000000000000100000

Instruction Decode ----->
ADD:
Value of RS= 0 Value of RT= 0

ALU ----->
After Add: 0

Writeback ----->
rd : 0
MemtoReg:  0
Write Data:  00000000000000000000000000000000
After write back, value at destination:  00000000000000000000000000000000
---------------------------------------------------------------------------
34
```

3) Power Calculation

The following screenshots show the calculation of the 4th power of 3 which is 81:



```
---------------------------------------------------------------------------
PC:  00000000000000000000000000011100
Instruction Fetch ----->
Instruction:  00000001001010110100100000100010

Instruction Decode ----->
SUB:
Value of RS= 4 Value of RT= 1

ALU ----->
After Sub: 3

Writeback ----->
rd : 9
MemtoReg:  0
Write Data:  00000000000000000000000000000011
After write back, value at destination:  00000000000000000000000000000011
---------------------------------------------------------------------------
PC:  00000000000000000000000000100000
Instruction Fetch ----->
Instruction:  00010001001011000000000000000001

Instruction Decode ----->
BEQ:
Value of RS= 3 Value of RT= 0 Value of Imm= 1

ALU ----->
Immediate:  4

---------------------------------------------------------------------------
PC:  00000000000000000000000000100100
Instruction Fetch ----->
Instruction:  00001000000100000000000000000110

Instruction Decode ----->
JUMP:
Jump address= 00000000010000000000000000011000

---------------------------------------------------------------------------
```



```
PC:  00000000000000000000000000101000
Instruction Fetch ----->
Instruction:  10101100000010100001011101110000

Instruction Decode ----->
STOR:
Value of RS: 0 Value of RT: 81 Value of Imm: 6000

ALU ----->
After Add: 6000

Memory:
Write Data:  00000000000000000000000001010001

---------------------------------------------------------------------------
81
```