Architectural Overview and Working of a Basic CPU Simulation

This report provides an overview of the working of a basic CPU simulation based on the different files shared. The architecture of the simulation involves several key components, including the Program Counter (PC), Instruction Memory (IMem), Data Memory (DMem), and Register File. Each component is responsible for different aspects of the CPU's operation, from fetching instructions to executing them and interacting with memory. The flowchart, shown below, illustrates the flow of data and control signals within the CPU simulation.

Components and Their Functions

Program Counter (PC): The Program Counter (PC) holds the address of the next instruction to be fetched from the Instruction Memory (IMem). Initially, it points to the first instruction, and after each instruction fetch, it is incremented to point to the next instruction. The PC plays a crucial role in maintaining the sequence of instruction execution.

Instruction Memory (IMem): The Instruction Memory is responsible for storing the program's instructions in binary format. It is initialized with a set of instructions that are loaded from a file. The InstructionMemory class in the code provides the methods to load instructions from a file and to fetch instructions based on the address supplied by the Program Counter. The instructions are retrieved in binary format and are passed on for execution.

Data Memory (DMem): The Data Memory simulates the storage of data during the execution of the program. The DataMemory class provides methods for reading from and writing to memory. The memory is implemented using a vector of uint16_t elements, which represent data entries at specific addresses. The writeDataToMemory function allows writing data to memory when the write-enable signal (we_dm) is set to true, and the getData function retrieves data based on the memory address.

Register File: The Register File holds a set of registers used for storing intermediate values during instruction execution. The RegisterFile class provides methods to read from and write to the registers. The write register method writes a value to a specified register if the write-enable signal (we_reg) is active. The readRegister method retrieves the value of a specified register. This component is critical for storing values that are needed for arithmetic or logical operations.

Control Logic: The control logic is responsible for coordinating the operations of various components within the CPU. It manages the flow of data by determining when to fetch instructions when to read or write data from memory, and when to perform register operations. Although not explicitly detailed in the files provided, the control unit would decode instructions and generate the necessary control signals to guide the CPU through its operations.

Instruction Fetch and Execution Cycle

The basic flow of instruction execution can be broken down into the following steps:

Fetch: The Program Counter (PC) provides the address of the next instruction to be fetched. The instruction is fetched from Instruction Memory (IMem) based on this address.

Decode: Once the instruction is fetched, it is decoded to determine the type of operation that needs to be performed. This phase typically involves identifying the opcode and operands.

Execute: Based on the decoded instruction, the relevant operation is performed. This could involve reading or writing data to the register file, performing arithmetic operations in the Arithmetic Logic Unit (ALU), or interacting with data memory.

Memory Access: If the instruction involves reading from or writing to data memory, the CPU accesses the Data Memory (DMem) to retrieve or store the required data.

Write-back: If the instruction involves updating a register, the value is written back to the register file.

Increment PC: After completing the current instruction, the Program Counter (PC) is updated to point to the next instruction, and the cycle repeats.

Recursive Factorial Calculator

The CPU simulation implements a recursive factorial calculator using instructions stored in Instruction Memory. Here's how it works:

1. Recursive Function Logic: The program implements recursion by saving the return address and intermediate values in registers or memory. Each recursive call decrements the input value (n) until the base case (n == 1) is reached.
2. Base Case: When n == 1, the function returns 1.
3. Recursive Step: The CPU multiplies n by the factorial of n - 1, fetched via recursive calls, and stores the result in a register.
4. Control Flow: The CPU uses jump (JAL, JR) and branch (BEQ, BNE) instructions to handle recursive calls and return to the previous state.

The Register File is crucial for storing intermediate values like n, the return address, and the recursive result. Data Memory is used to hold temporary variables if needed.

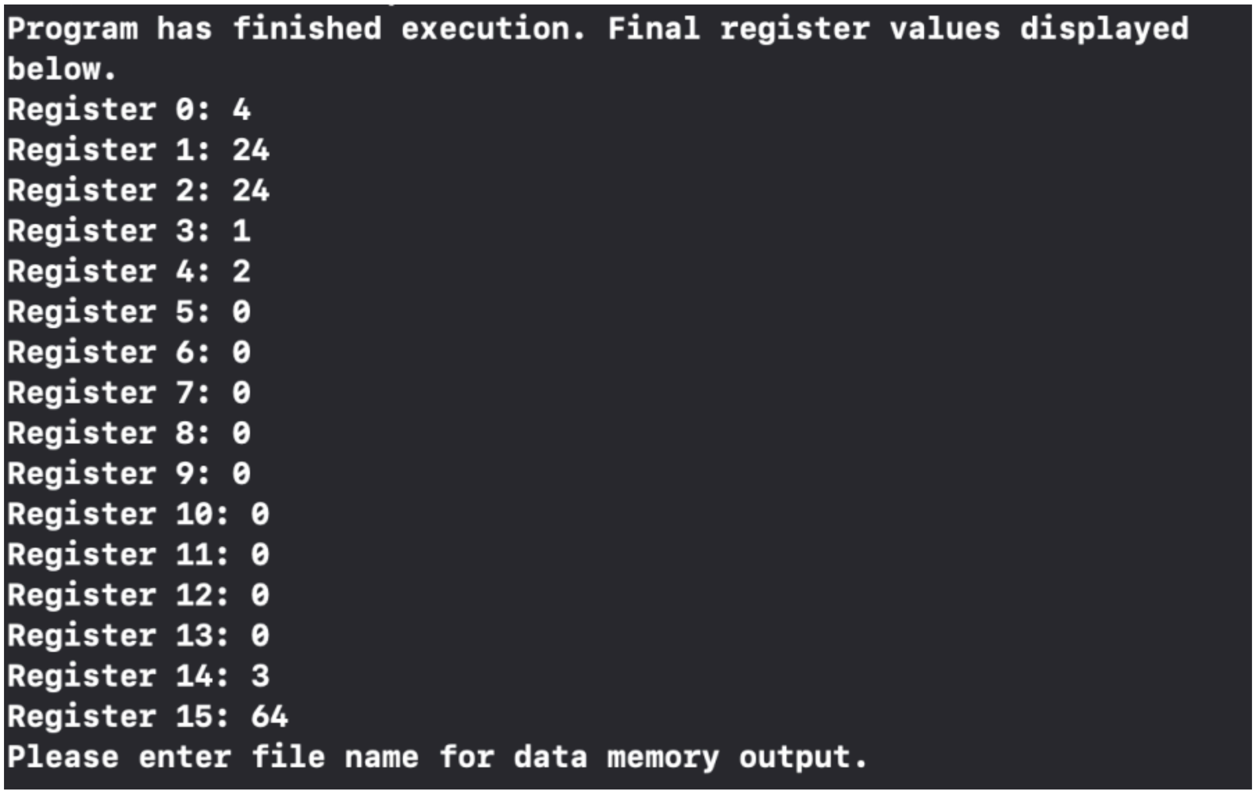
Test Program

A test program in binary format can be written to validate the CPU simulation. Here's an example of how a factorial calculation is simulated:

```
li 15 64
li 0 4
jal 6
li 1 0
add 1 2
j 25
addi 15 -2
sw 0 15 1
sw 14 15 0
li 3 0
add 3 0
li 4 2
slt 3 4
bns 3 3
li 2 1
addi 15 2
```

```
jr 14
addi 0 -1
jal 6
lw 14 15 0
lw 0 15 1
addi 15 2
mult 2 0
mflo 2
jr 14
```

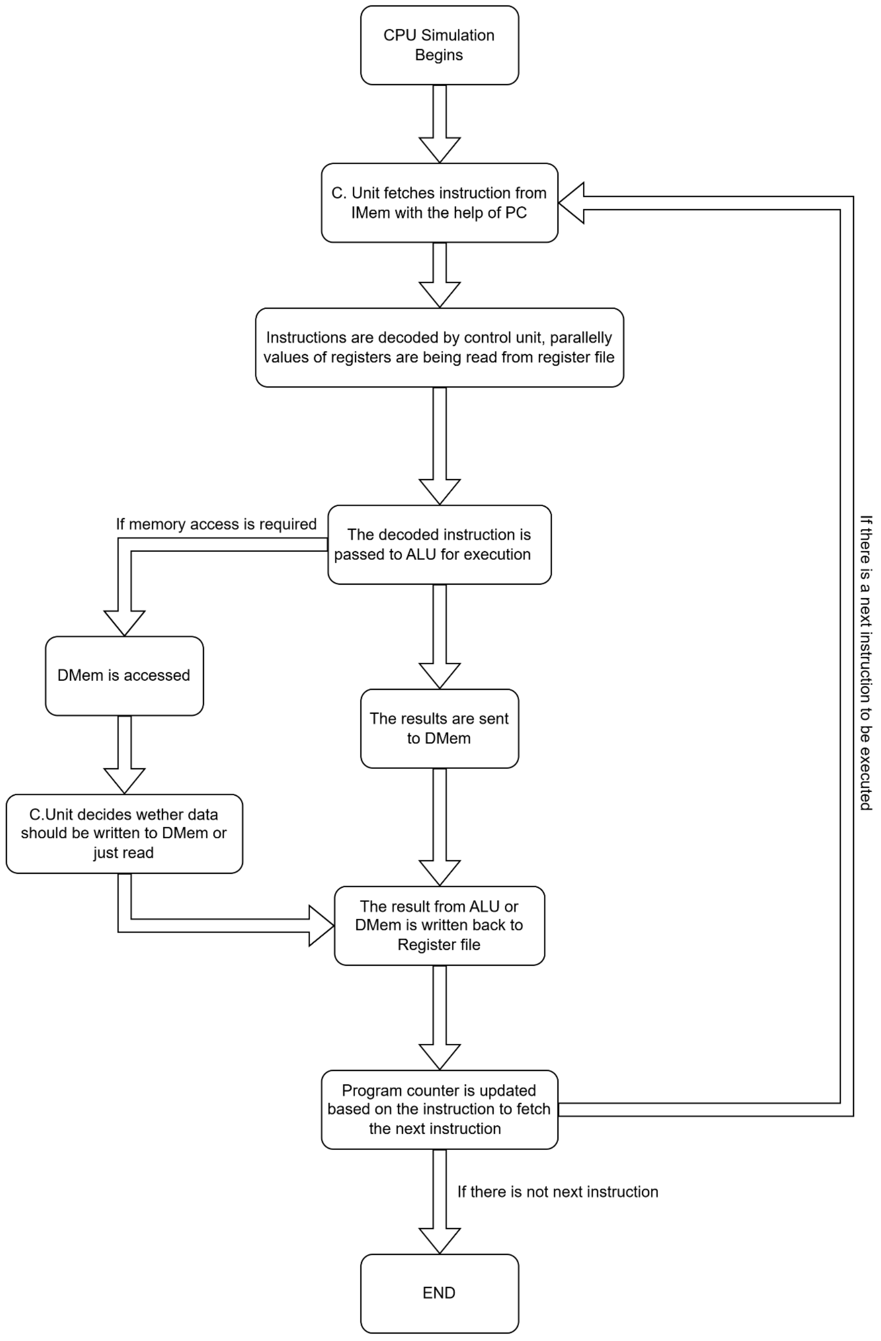
Expected Output: A screenshot of the output showcasing the result of calculating the factorial of 4 by executing the test code mentioned above.



```
Program has finished execution. Final register values displayed
below.
Register 0: 4
Register 1: 24
Register 2: 24
Register 3: 1
Register 4: 2
Register 5: 0
Register 6: 0
Register 7: 0
Register 8: 0
Register 9: 0
Register 10: 0
Register 11: 0
Register 12: 0
Register 13: 0
Register 14: 3
Register 15: 64
Please enter file name for data memory output.
```

Flowchart Overview

The flowchart below provides a detailed overview of the operation of the CPU simulation, visually representing how data flows between components during the instruction cycle. The key stages are instruction fetch, decode, execution, memory access, and write-back. The PC, IMem, DMem, and Register File interact as described, and the control unit coordinates the overall process.


CPU Simulation Begins
C. Unit fetches instruction from IMem with the help of PC
Instructions are decoded by control unit, parallelly values of registers are being read from register file
If memory access is required
The decoded instruction is passed to ALU for execution
DMem is accessed
The results are sent to DMem
C.Unit decides wether data should be written to DMem or just read
The result from ALU or DMem is written back to Register file
Program counter is updated based on the instruction to fetch the next instruction
If there is a next instruction to be executed
If there is not next instruction
END

Supported Instructions

The CPU simulation is designed to execute a wide range of instructions that allow it to perform arithmetic, logical, memory, and control flow operations. Below is the list of instructions supported by the simulation:

1. Arithmetic Instructions: add, sub, mult, div
2. Logical Instructions: or, and, not, xor
3. Comparison Instructions: slt (set less than), sgt (set greater than), seq (set equal)
4. Shift Instructions: sll (shift left logical), srl (shift right logical)
5. Memory Instructions: sw (store word), lw (load word)
6. Control Flow Instructions: j (jump), jr (jump register), jal (jump and link)
7. Immediate and Register Instructions: li (load immediate), lui (load upper immediate), addi (add immediate)
8. Branching Instructions: bis (branch if set), bns (branch if not set)
9. Special Register Instructions: mflo (move from low), mfhi (move from high)

These instructions provide the CPU simulation with the versatility to handle a variety of computational and logical tasks, enabling it to function as a basic yet robust processor.

Conclusion

The basic CPU simulation involves key components working together to fetch instructions, decode them, and execute operations while interacting with memory and registers. The flowchart below highlights the interaction between the Program Counter, Instruction Memory, Data Memory, and Register File, ensuring the CPU can execute instructions sequentially. The simulation model effectively demonstrates the fundamental operations of a CPU, including instruction fetching, decoding, execution, and memory handling.

By analyzing the structure and function of each component in the simulation, we gain a clearer understanding of how a basic CPU operates. This simulation provides a foundation for building more complex CPU architectures and can be extended with additional features, such as support for more advanced instructions, pipelining, and error handling. The flowchart integration helps visualize the entire process, making it easier to understand the data and control flow within the CPU.