Contents

[Documentation for the exec\_bytecode() Function:- 1](#_Toc134732894)

[Function Definition: 1](#_Toc134732895)

[Program Execution: 1](#_Toc134732896)

[Instruction Execution: 1](#_Toc134732897)

[Endianness Considerations: 2](#_Toc134732898)

[Final State Display: 2](#_Toc134732899)

[Return: 2](#_Toc134732900)

# 

# Documentation for the exec\_bytecode() Function:-

The exec\_bytecode() function in this code is a MIPS simulator written in C. It executes a MIPS program, given that the program's instructions are stored in the text array, and its data is stored in the data array.

## Function Definition:

*int exec\_bytecode();*This function does not take any arguments and returns an integer. The return value is always 0, indicating successful completion of the program execution.

## Program Execution:

At the beginning of the function, the program counter (pc) is set to ADDR\_TEXT, the start of the text segment (where the program instructions are stored).

The function then enters a loop, which continues until the terminate flag is set to true. In each iteration of the loop, one instruction is fetched, decoded, and executed.  
  
Instruction Fetching and Decoding:

Each instruction is fetched from the text array using the program counter (pc). The instruction is then decoded into its components (opcode, registers, immediate value, etc.) using bitwise operations.

## Instruction Execution:

The function uses a switch statement to determine which instruction to execute based on the opcode. R-type instructions are further decoded using another switch statement based on the func field.

The implementation includes the following instructions:

* ADD: Adds the values in two source registers and stores the result in a destination register.
* NOP: No operation is performed, and the terminate flag is set to true.
* SRL: Shifts a register value right by a certain amount and stores the result in a destination register.
* ADDI: Adds an immediate value to a source register and stores the result in a destination register.
* ANDI: Performs a bitwise AND operation between a source register and an immediate value and stores the result in a destination register.
* LUI: Loads an immediate value into the upper half of a destination register.
* BNE: Branches to an address if two registers are not equal.
* LW: Loads a word from memory into a register.
* SW: Stores a word from a register into memory.
* LHU: Loads a halfword from memory into a register, zero-extends it to 32 bits.
* SH: Stores a halfword from a register into memory.

## Endianness Considerations:

The code is designed to work correctly regardless of the endianness of the host machine. It uses the ntohl function to convert data from network (big-endian) order to host order when reading from memory, and the htonl function to convert from host order to network order when writing to memory.

## Final State Display:

After all instructions have been executed, the function prints the final state of the registers and the memory to the standard output.

## Return:

The function then returns 0 to indicate successful execution.