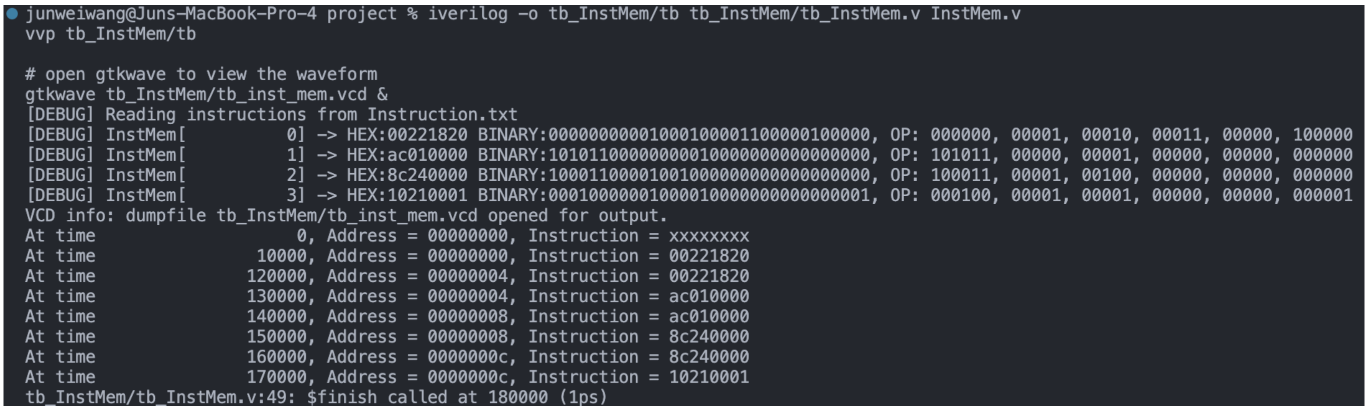
InstMem (Instruction Memory)

```
iverilog -o tb_InstMem/tb tb_InstMem/tb_InstMem.v InstMem.v
vvp tb_InstMem/tb

# open gtkwave to view the waveform
gtkwave tb_InstMem/tb_inst_mem.vcd &
```

Benchmark (verifying the correctness)

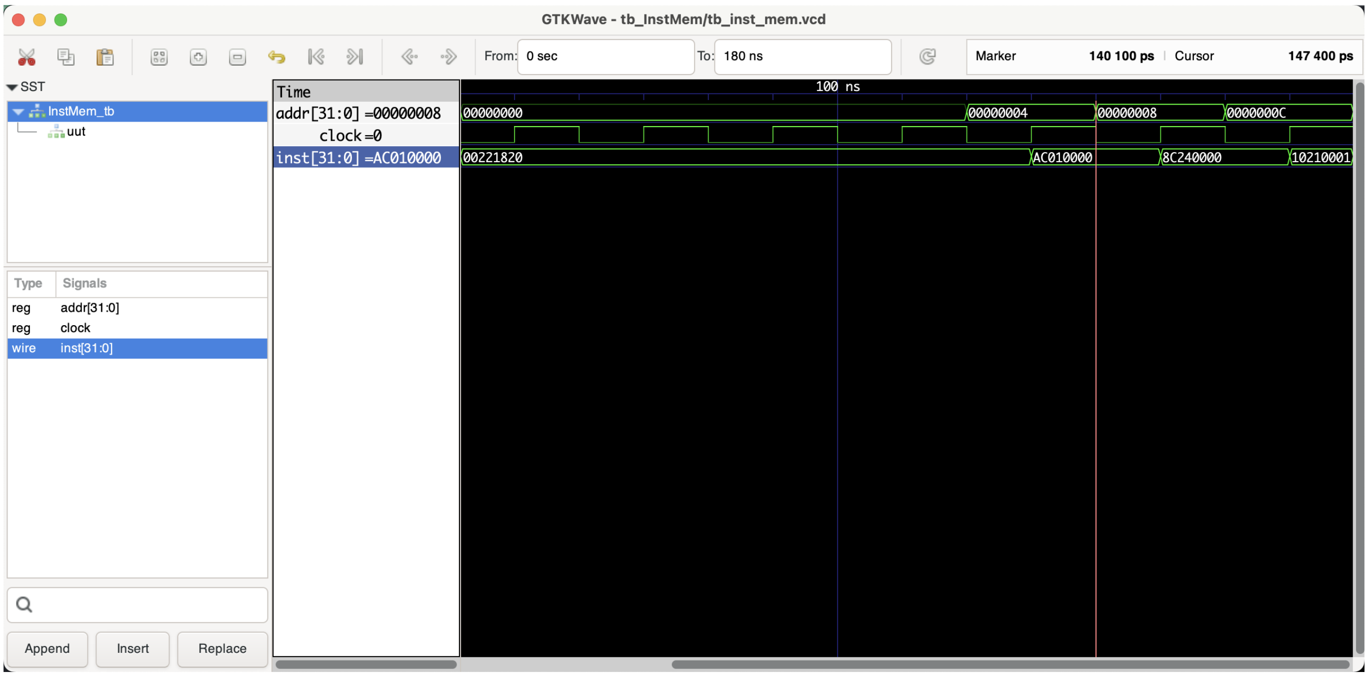
Shell output



```
junweiwang@Juns-MacBook-Pro-4 project % iverilog -o tb_InstMem/tb tb_InstMem/tb_InstMem.v InstMem.v
vvp tb_InstMem/tb

# open gtkwave to view the waveform
gtkwave tb_InstMem/tb_inst_mem.vcd &
[DEBUG] Reading instructions from Instruction.txt
[DEBUG] InstMem[          0] -> HEX:00221820 BINARY:00000000001000100001100000100000, OP: 000000, 00001, 00010, 00011, 00000, 100000
[DEBUG] InstMem[          1] -> HEX:ac010000 BINARY:10101100000000010000000000000000, OP: 101011, 00000, 00001, 00000, 00000, 000000
[DEBUG] InstMem[          2] -> HEX:8c240000 BINARY:10001100001001000000000000000000, OP: 100011, 00001, 00100, 00000, 00000, 000000
[DEBUG] InstMem[          3] -> HEX:10210001 BINARY:00010000001000010000000000000001, OP: 000100, 00001, 00001, 00000, 00000, 000001
VCD info: dumpfile tb_InstMem/tb_inst_mem.vcd opened for output.
At time                    0, Address = 00000000, Instruction = xxxxxxxx
At time                10000, Address = 00000000, Instruction = 00221820
At time               120000, Address = 00000004, Instruction = 00221820
At time               130000, Address = 00000004, Instruction = ac010000
At time               140000, Address = 00000008, Instruction = ac010000
At time               150000, Address = 00000008, Instruction = 8c240000
At time               160000, Address = 0000000c, Instruction = 8c240000
At time               170000, Address = 0000000c, Instruction = 10210001
tb_InstMem/tb_InstMem.v:49: $finish called at 180000 (1ps)
```

Waveform



Documentation

Module Declaration:

```
module InstMem(clock, addr, inst);
```

This line declares the module's name `InstMem` and specifies its interface, which consists of a clock input, an address input, and an instruction output.

Port Definitions:

- `input clock;`: The clock signal input. It synchronizes the reading of instructions from memory.
- `input [31:0] addr;`: A 32-bit address input that specifies the memory address from which the instruction should be read.
- `output reg [31:0] inst;`: A 32-bit register that holds the instruction read from the specified address.

Memory Declaration:

```
reg [31:0] Mem [0:127];
```

This line declares a 128-entry array of 32-bit registers, representing the instruction memory. Each entry corresponds to a word of memory where instructions are stored.

Initialization:

The initial block uses `$readmemh` to read a set of predefined instructions from a file named "Instruction.txt" into the instruction memory. It then iterates through the initialized memory locations and prints their contents for debugging purposes.

Functional Description:

- **Memory Reading**: On the positive edge of the clock, the module updates the `inst` output with the value from the memory location specified by the upper bits of the `addr` input (word-aligned by using `addr[31:2]`).

Word-Aligned Addressing:

- The module uses `addr[31:2]` to index the memory array. This reflects word-aligned memory addressing, commonly used in 32-bit systems. By using only the upper bits of the 32-bit address, the module effectively addresses memory in 4-byte (32-bit) increments, which is consistent with the word size of the memory.

Overall Functionality:

The `InstMem` module simulates the instruction memory of a MIPS CPU. It responds to a clock signal for synchronized read operations. The memory can be read from any given address, and it provides the instruction stored at that address through the `inst` output. The module is essential in a CPU design as it holds the sequence of instructions that the CPU will execute.

Debugging and Initialization:

- The module includes a debug message that prints out the contents of the memory after initialization. This feature is helpful for verifying that the instructions are loaded correctly and for understanding how the module behaves during simulation.

Example Usage:

In a MIPS CPU architecture, this module would serve as the instruction memory where all the executable instructions are stored. The CPU fetches instructions from this memory sequentially or based on branch and jump instructions, which provide the addresses. The `InstMem` module provides a critical function in the fetch stage of the CPU pipeline.

Conclusion:

The `InstMem` module is a fundamental component in a MIPS CPU simulation, representing the instruction memory from which instructions are fetched during the execution of a program. Its design and functionality ensure that instructions are available to the CPU in a synchronized and orderly manner, consistent with the MIPS architecture and the requirements of a single-cycle or multi-cycle processing model.