Verilog Primer

R[rd], R[rs1], and R[rs2] mean the value held in register rd, rs1, and rs2 respectively. "rd" is register destination (the register that the result of an operation will be written to), rs1 is the first argument register, and rs2 is the second argument register. When you're writing an instruction, if there are 2 argument registers, rs1 is usually the one that comes first, with the exception of loads and stores. If there's only one argument, that's rs1. Some examples:

- In "add x1, x2, x3", x1 is rd, x2 is rs1, and x3 is rs2.
- In "addi x1, x2, 5", x1 is rd and x2 is rs1, and there is no rs2.
- In "beq x1, x2, label", x1 is rs1 and x2 is rs2.
- In "jal x1 label", x1 is rd.
- In "lw x1, 0(x2)", x1 is rs2 and x2 is rs1

Mem[___] means treat whatever's in the brackets as an address in physical memory. So Mem[R[rs1] + imm] means add the immediate to the contents of register # rs1, and use that as a location in memory.

PC stands for "program counter", and it is the address of a location in the text or code segment of physical memory. Along with your 31 registers x0 through x31, your processor actually also has an extra register that is holding the PC, which is the address of the instruction you're currently executing. Branch and jump instructions modify the PC, which in turn will change what instruction you execute next. The default after executing any 32-bit instruction is to add 4 to your PC, so you go on to execute the next instruction.

In verilog, xb'y means an **x**-bit **b**inary immediate with value **y**, and the curly braces { } mean concatenate what's inside. Some examples:

- {imm, 1'b0} means the immediate followed by one zero
- {imm, 12b'0} means the immediate followed by 12 zeros

The ( ) parentheses mean those index bits of the number, so M[R[rs1] + imm](15:0) means take the bottom 16 bits from whatever word in memory is at address R[rs1] + imm.

If you see a ; that means this instruction does multiple things, and each operation is separated by a ;.

If you see something like ___ ? ___ : ___ that's the ternary operator; it works in Verilog like it does in C. Essentially, it's in the format condition ? statement1 : statement2, and what gets executed is if condition then statement1 else statement2.