The NOT Instruction

The High Level
The **NOT** instruction is a type of **Arithmetic/Logic instruction instruction** that logically NOTs a value and stores the result in a **destination register**. When not "NOT" a bit, it inverts the value (i.e. 1-becomes-0 or 0-becomes-1) (See Table 1 below).

NOT True	False
NOT False	True

Table 1: Truth table for AND

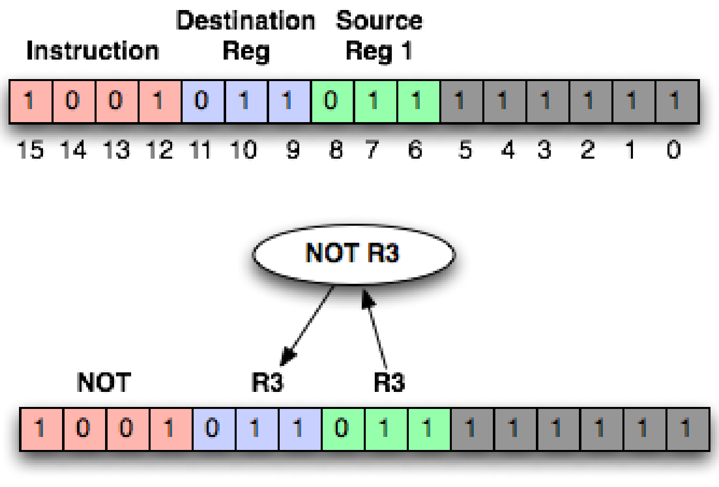

Figure 1: The NOT instruction

Breakdown:

- Bits [15-12] specify what instruction to execute (1001 is NOT)
- Bits [11-9] specify which register to store the result in
- Bits [8-6] specify which register to perform the logical NOT on
- Bits [5-0] are all 1 because they are unused

The Examples!

```
NOT R1, R2     ; R1 <-- NOT R2
NOT R1, R1     ; R1 <-- NOT R1     (Protip: This is **HALF** of the process of
                                    negativizing the value of a register)
```

Pitfalls... (aka: Erroneous code makes baby puppies cry)
The example below is erroneous. Please do NOT try to code this way!

```
   NOT R1, #1   ; (ERROR: You MUST use registers for both destination and source)
```

The example pitfall code above is incorrect because there are no forms of the NOT instruction that allow any immediate values to be used. The only allowable form that the NOT instruction takes is to use a register for both the destination and the source.