Using and/andi to extract number:

This document is explain how we can use and/andi instruction to extract a number of bits from the whole number.

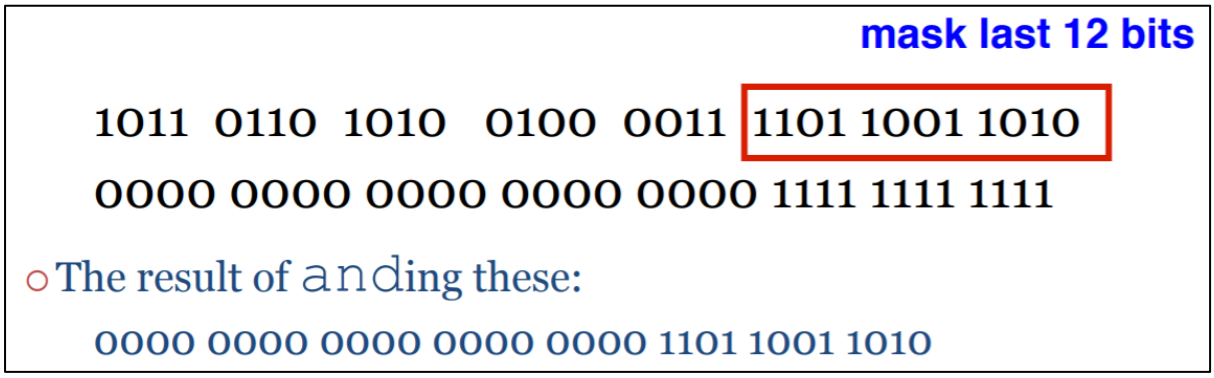
AND logic:

The and operation yields 1 iff both the source bits are 1:

1101 1010 AND 1011 0011 ---> 1001 0010

Note that **anding** a bit with 0 produces a 0 at the output while anding a bit with 1 produces the original bit. This can be used to create what is called a **mask**.

Example 1:



The second bitstring in the example is called a mask. It is used to isolate the rightmost 12 bits of

the first bitstring by masking out the rest of the string (e.g. setting it to all 0s).

Example 2:

We can use AND to extract any part of a bit sequence that we like:

x: 0101 1100 **0000 1101 0011 0101** 1010 0011

Suppose we want to extract the indicated bits.

Recall that for any value (0 or 1) of A, A and 0 equals 0 and A and 1 equals A.

So, the key to our problem is to create a suitable mask:

mask: 0000 0000 **1111 1111 1111 1111** 0000 0000

Note we put 1's where we want to capture bits and 0's where we want to annihilate them.

Now:

x and mask: 0000 0000 **0000 1101 0011 0101** 0000 0000

andi Instruction

In our assembly code that we wrote to verify IP checksum (practical week-12), we used andi instruction to extract the lower 16-bit from the 32-bit number.

The andi instruction does a bitwise AND of two 32-bit patterns. At run time the 16-bit immediate operand is padded on the left with zero bits to make it a 32-bit operand.

```
andi d,s,const      # register d loaded with
                    # bitwise AND of immediate operand const
                    # and the contents of register $s.
                    # const is a 16-bit pattern, so
                    # 0x0000 ... const ... 0xFFFF
```

The three operands of the instruction must appear in the correct order, and const must be within the specified range. The immediate operand in the source instruction always specifies sixteen bits although the zeros on the left can be omitted (such as 0x2).

In the practical work of week-12, we used **andi** instruction to extract the lower 16-bit from the whole number 32-bit.

```
andi $t1,$t1,0xFFFF
```

t1 is a 32-bit register, so after the execution of the above instruction the mask **0xFFFF** will extract only the lower 16-bit and save the result in **t1 register**.