L10 - `Oprnd2`

Feb 25, 2021	CSE 2312	Dr. Losh	UT Arlington	Luke Sweeney

`Oprnd2`

Page 1-18 of the technical manual shows `oprnd2` and it's formatting options. `Oprnd2` (or "operand 2") is the last 12 bits of the assembly command.

Look at the table on page 1-12. When you write a line of assembly, it gets translated into 32 bits in the format of one of those rows, depending on the operation. You can see `Oprnd2` at the end, the last 12 bits.

A problem arises because `Oprnd2` is only 12 bits. The first option on the operand 2 table (page 1-18) is an "immediate value" (like a hardcoded value) that is 32 bits. But we obviously can't fit a 32 bit number into 12 bits.

Constants

A 32-bit immediate (constant) value is encoded into a 12 bit `Oprnd2` field. The 12 bits are split into the following

```
4 bits      8 bits
---------------------
|   s   |       n   |
---------------------

value (m) = n ROR (2*s)
```

This last line reads "value `m` is `n`, rotate to the right by 2 times `s`".

`s` is a 4 bit number, and you'll rotate by `2 * s`. You can rotate from `0-30` (`s` can be `0-15`).

`n` is 8 bits (`0-255`)

How `m` is calculated

```
m = 40 = 0x28

This value of m would be encoded as

s = 0, n = 0x28
```

When **rotating**, the bits on the low end of the number (LSB) are "rotated" to the high end of the number. So LSB become MSB. For example

```
0000 0011 ROR 1
becomes
1100 0000
(1 is multiplied by 2)
```

Example

Lets say we want to encode the number 10,240. This in binary is 14 bits, which is too big to fit in `Oprnd2`. So instead, we can convert it to hex and rotate if necessary.

The way Losh describes this is pretty confusing. Not sure why he brought up hex at all. This is my interpretation

```
m = 10,240 (base 10)
    0x2800 (base 16)
    0000 0000 0000 0000 0010 1000 0000 0000 (base 2, 32 bit)
                        ^^^^ ^^^^
            these bits are the important ones
            we'll put them in n, but we need to
            rotate them to the correct place. Imagine
            if those bits were positioned as the last 8
            bits of the number. How many spaces do you need
            to move to the right to get them where
            they need to be? (24 bits)

s = 12 (becauseit's multiplied by 2, it really should be 24 in the end).

    --------------------
m = | 1100 | 0010 1000 |
```

```
--------------------
  (12)     (40)
```

Negative Numbers as constants

To use a negative number as an immediate, you can use the `MVN` instruction (move-not). This will invert all the bits of the number. So `0` would become a 32 bit number with all `1`s.