Register Shifting (G)

Let's discuss a type of binary multiplication and division. Generally, when a binary number is **shifted by one position to the left while inserting a 0 in the now vacant LSB (from the C-Flag)**, it is the equivalent of **multiplying the binary number by 2**. The reverse is true when we **shift the binary number to the right** by one position, inserting a **0** in the now vacant **MSB** (from the C-Flag). This is equivalent to **dividing the binary number by 2**.

This does cause a few issues.

- First, we must **make sure that the N value is large enough to accept the shift without either an overflow or a discard.**
  - **This will cause a loss in accuracy of some large or small amount.**
- **The other issue is:** What do we do with sign bits if by some chance you are shifting a "**Signed Magnitude**" system number?
  - **The sign bit isn't included in the shift.** Just leave it alone and shift the next bit over it.
  - **If the sign bit is part of a 2's complement number, then it will be included in the shift. In addition, instead of filling the vacant MSB with a 0, we fill the vacancy with a COPY of the sign bit.**

Let's look at a few examples:

Example 1: Shift to the LEFT:

Shift the register to the left.

13.25	**N**	**0**	**0**	**1**	**1**	**0**	**1**	**.**	**0**	**1_2**
26.5	**2*N**	**0**	**1**	**1**	**0**	**1**	**0**	**.**	**1**	**0_2**

- **Note that it had the same effect as multiplying by 2!**

Example 2: Shift to the RIGHT:

Next, we have an example of a shift to the right.

15	N	0	0	1	1	1	1	.	0	0_2
7.5	N/2	0	0	0	1	1	1	.	1	0_2

- **This time the effect was one of dividing by two!**

2's Complement Number Register Shifting:

Now, let's take a look at a shift in the "**2's complement**" system. In this case, when **shifting right, the sign bit is included in the shift. In addition, instead of filling the vacant MSB with a 0, we fill the vacancy with a COPY of the sign bit.**

Example 3: 2's Complement Number Shift to the RIGHT

Let's see how this works.

- 1st, we find the negative two's complement of 14_{10}: **(N=9, 2 fraction bits)**

position	2^6,	2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}
+14	0,	0	0	1	1	1	0	.	0	0_{2cm}
−14	1,	1	1	0	0	1	0	.	0	0_{2cm}

- Now we can start shifting to the right and see what happens when **a shift to the RIGHT** occurs.

	position	2^6,	2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	
Before	−14	1,	1	1	0	0	1	0	.	0	0_{2cm}	
Shift	*chk* = −14	−64,	+32	16			2		.			
After	$\frac{-14}{2} = -7$	1,	1	1	1	0	0	1	.	0	0_{2cm}	
Shift	*chk* = −7	−64	+32	16	8			1				⇐ *VERIFIED*

  - As you can see, we had the effect of **dividing:** $\frac{-14}{2} = -7$
  - **Note that the sign bit was included in the shift but a copy of it was placed in the MSB position which it vacated.**

- Let's shift to the **RIGHT** one more time:

	position	2^6,	2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	
Before Shift	$\frac{-14}{2} = -7$	1,	1	1	1	0	0	1	.	0	0_{2cm}	
After Shift	$\frac{-7}{2} = -3.5$	1,	1	1	1	1	0	0	.	1	0_{2cm}	
	chk $= -3.5$	−64		+32	16	8	4			.5		⇐ *VERIFIED*

  - Again it should be obvious that we **divided** $\frac{-7}{2} = -3.5$

- Let's shift to the **RIGHT** one <u>more</u> time!:

	position	2^6,	2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	
Before Shift	−3.5	1,	1	1	1	1	0	0	.	1	0_{2cm}	
After Shift	$\frac{-3.5}{2} = -1.75$	1,	1	1	1	1	1	0	.	0	1_{2cm}	
	chk $= -1.75$	−64	+32	+16	+8	+4	+2		.	.25		⇐ *VERIFIED*

  - We have gotten away with it again. The value after the shift is as predicted.

- **If we were to shift one more time we would lose accuracy. The results would be -64 + 32 + 16 + 8 + 4 + 2 + 1 = -1 which is obviously WRONG.**
- The key to making sure that this does not happen is to **check the C-Flag after each shift.**
  - **If it is 0, then a 0 was shifted off with no loss in accuracy.**
  - **If it is a 1, then a 1 was shifted off and there WAS A LOSS IN ACCURACY which could be significant!**

Example 4: 2's Complement Number Shift to the LEFT

Ok, now let's look at an example of a "**2's Complement**" system register **shift to the left**. In this case, the **number is treated just like any other number, with a 0 being placed in the vacant LSB**. Let's look at the example:

- Find the **negative** of $+6.25_{10}$

	position	2^5		2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	
Positive	+6.25	0	,	0	0	1	1	0	.	0	1_{2cm}	
Negative	−6.25	1	,	1	1	0	0	1	.	1	1_{2cm}	
	check = −6.25	−32		+16	+8			+1		+.5	+.25	⇐ *Verified*

- Now, shift once to the **LEFT**

Before	−6.25	1	,	1	1	0	0	1	.	1	1_{2cm}	
After	−6.25 * 2 = −12.5	1	,	1	0	0	1	1	.	1	0_{2cm}	
Shift	*check* = −12.5	−32		+16			+2	+1		+.5		⇐ *Verified*

- As expected, the result is double the original negative number.
- Shift the register to the **LEFT** again.

Before	−12.5	1	,	1	0	0	1	1	.	1	0_{2cm}	
After	−12.5 * 2 = −25	1	,	0	0	1	1	1	.	0	0_{2cm}	
Shift	*check* = −25	−32				+4	+2	+1				⇐ *Verified*

- **The result is still double the 'before shift' value.**
- Let's try one more shift to the **LEFT**.

Before	−25	1	,	0	0	1	1	1	.	0	0_{2cm}	
After	−25 * 2 = −50	0	,	0	1	1	1	0	.	0	0_{2cm}	
Shift	*Check* = *OOPS!*											*NOT Verified!!!!*

Note that everything was going great in the example above until, "**OOPS!**", **the number was no longer negative!** When the sign bit changes in a 2's complement register shift to the right, an overflow condition has occurred and you can't continue.