I have been trying to learn PASM off and on for a while. After reviewing many tutorials and much of the Parallax forums, I found it not easy to get basic information about just simply communicating with PASM. Everybody wants to blink a light. That is great but how does one do simple math, an array and other tasks that are relatively simple in SPIN or Prop C.

My project involves GPS and other sensors. I decided that I would tackle the project in PASM. So, while attempting to learn to code in assembly I got some jump starts from David and Jeff at parallax which was a great help, scoured the forums and despite finding many broken links and digging through some older tutorials, I found some information. Still everyone wants to blink a light.

I wrote a version of the tutorial that is in the LEARN section for creating Prop C libraries and was encouraged by the compliments, Thank you all.

My approach to that rewrite was from the aspect of a teacher not an engineer as I am a flight instructor and an aircraft mechanic instructor at a college in the Los Angeles area. So, I attempted to not be too geeky with the tutorial so as to appeal to the inexperienced and those who are really techy.

So here is my attempt at a PASM tutorial.

No Blinky lights in the beginning!!!!

The first thing one will need is a copy of the propeller manual that is in the propeller tool and can be found here: <https://www.parallax.com/product/122-32000>.

Here is a link to Jeff Martin’s webinar I uploaded to youtube: https://www.youtube.com/watch?v=OZHuWYW3o1A

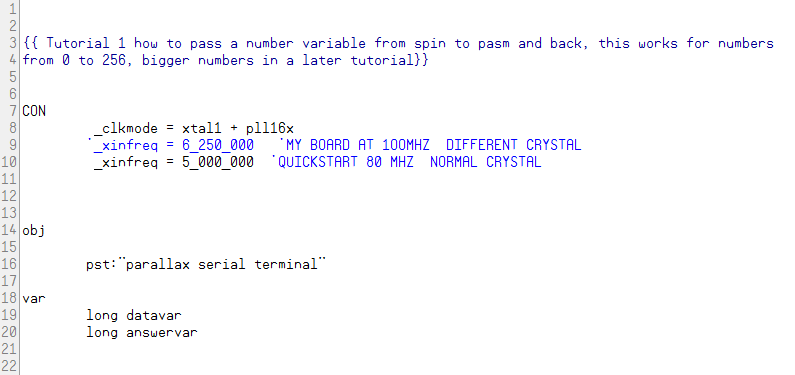
The first exercise will encompass passing variables from a spin method to a pasm method and back.

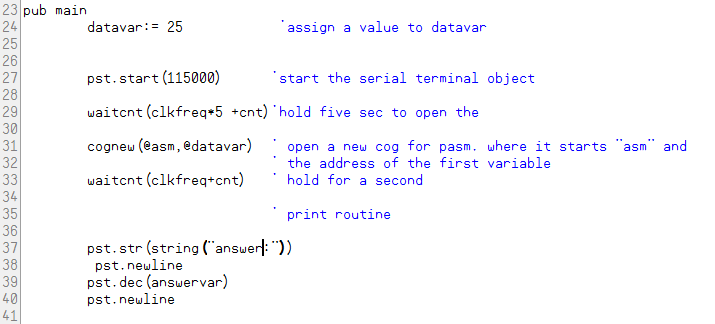
This the first piece of code that I came up with. There may be better ways to do this so bear with me.

I setup two global variables one for the spin method and the other for the pasm method. A five second waitcnt is used so as to have time to open the serial terminal when launching the code.

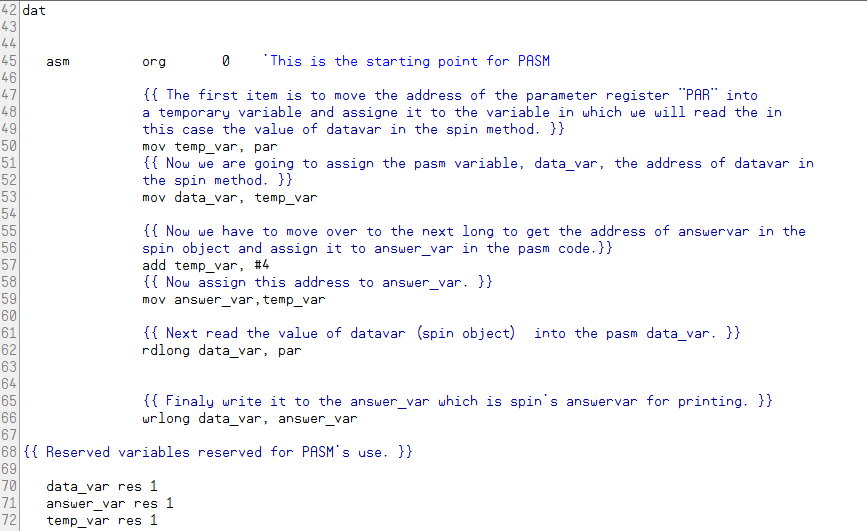
In order to launch the pasm code into a new cog this command is needed:

cognew(@asm,@datavar). The cognew means open the next cog, the @asm is the beginning of the assembly routine and the @datavar is the address of the first global variable.



The next steps are to start the serial terminal wait five seconds to allow one to open the serial terminal and then launch the cog. The code will then take the value in data var and print on the terminal. Now to the PASM method: 

The datavar is assigned a value, in this case 256 which is the maximum pasm will handle without extra work. I will tackle that at a later time. We want to keep it simple at this time. This is also because many of the other tutorials I have seen get really complicated very quickly and do not take it in baby steps. I want to make sure that everybody can grasp the concept before getting into complicated code and get lost.



The pasm code starts in a “dat” section of spin. The “asm” “org” “0” indicates the beginning of the pasm code. In the cognew there is also an @datavar expression. This tells the pasm code the address of the first variable and that address will be stored in the “par” value. “par” from what I have found means parameter.

There is a very nice webinar done by Jeff Martin in 2009 that explains a lot of information regarding pasm code. I uploaded it to youtube: https://www.youtube.com/watch?v=OZHuWYW3o1A.

Starting at:

mov temp\_var, par

This is the mov instruction description:

MOV

**Instruction:** Set a register to a value.

MOV ***Destination***, # ***Value***

**Result:** *Value* is stored in *Destination*.

***Destination*** (d-field) is the register in which to store *Value*.

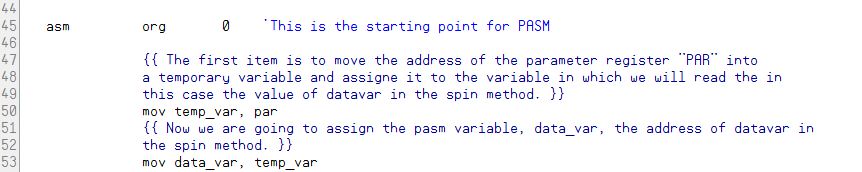
***Value*** (s-field) is a register or a 9-bit literal whose value is stored into *Destination*.

Explanation

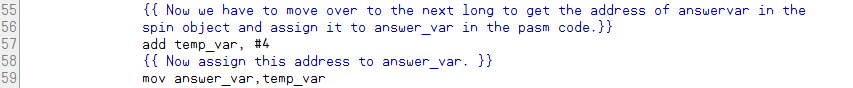
MOV copies, or stores, the number in *Value* into *Destination*.

If the WZ effect is specified, the Z flag is set (1) if *Value* equals zero. If the WC effect is specified, the C flag is set to *Value’s* MSB. The result is written to *Destination* unless the NR effect is specified.

So, our first instruction directive will take the address of the spin code datavar variable in the registers and pass it to a temporary variable that we can manipulate. The code is commented so as to follow the progression and I am using full words instead of abbreviations so as one could more easily follow the progression.



Now we have the address of the data\_var which corresponds to datavar in the spin method.



As you can see, we move over and get the address of the spin code answervar variable and assign it’s address to the pasm code answer\_var variable. This is done by adding 4 to the temporary variable. Adding 4 moves to the next adjacent long where the answer var is located in the hub.

We are next going to use the rdlong and wrlong directives. The rdlong directive will read from a location and copy the value into a destination field as is shown in the propeller manual listing.

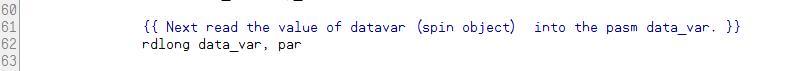
RDLONG ***Value***, # ***Address***

**Result:** Long is stored in *Value*.

***Value*** (d-field) is the register to store the long value into.

***Address*** (s-field) is a register or a 9-bit literal whose value is the main memory address to read from.

The rdlong goes from right to left. We are reading the value that is in the par register which has the location of datavar and it’s contents.



Lastly, we are going to write the value to the answer\_var location that corresponds with answervar in the spin method and then print the results in a new variable. Note: wrlong works from ***left to right.***

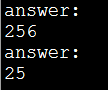


You should get a value on the serial terminal. I used 256 as this is the largest value for a single long, which is four bytes in size.

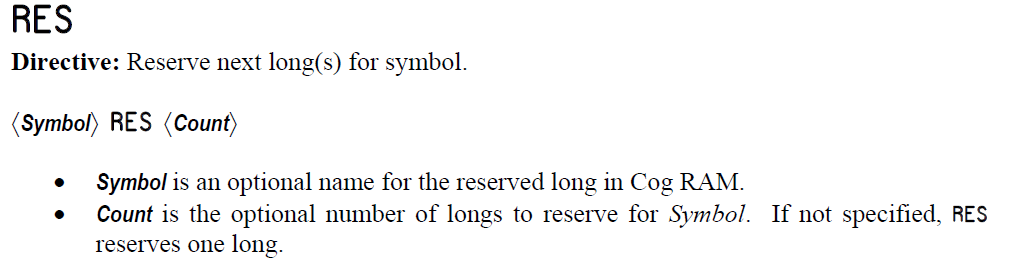


Changing the value of datavar to 25 in the spin method to verify.

BEFORE

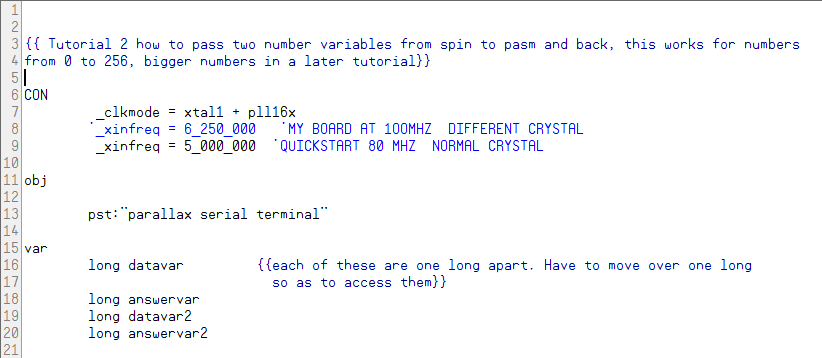


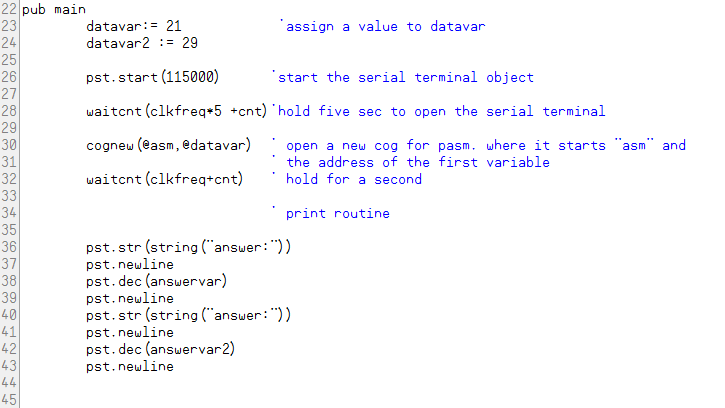
AFTER



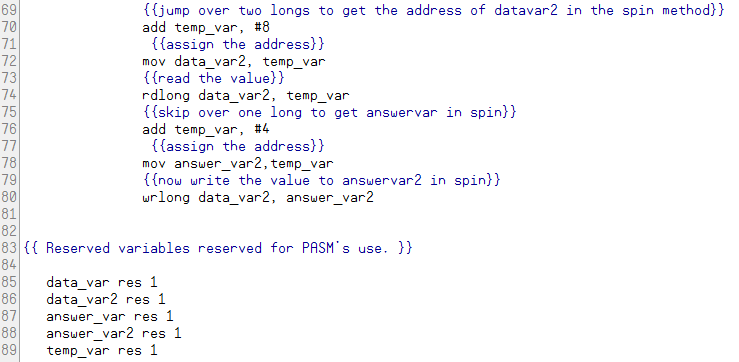
RES: We need to reserve space for the pasm variables this is self explanatory.

Now we can manipulate two variables and print them in succession. This is the new code:

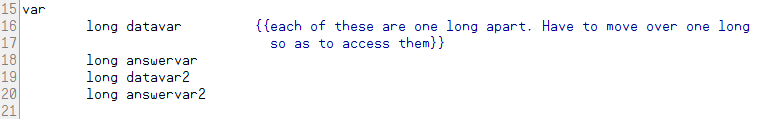








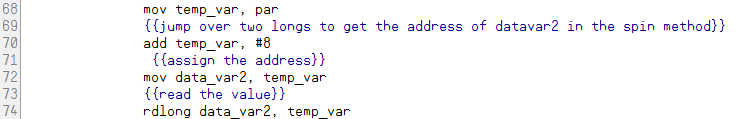
We have added a couple of items. First a new datavar named datavar2 and a new answervar named answervar2 as well as their counterparts in the pasm method. In the print area answervar2 has been added also.



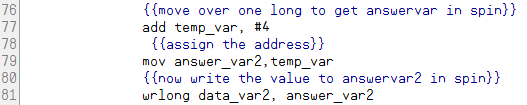
Note the order of the global variables. This will make it easy to find them in the pasm method.

The pasm routine begins just like before and we get the location of datavar from par into the temporary variable and assign the location to data\_var and read the value from par to data\_var.

Now we have to move over a couple of longs to get the new variables and values:



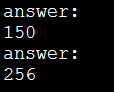
Now we can write the value to the second answer\_var. Remember wrlong is from left to right as opposed to rdlong and other directives which are right to left.



This is what you should see on the serial terminal:



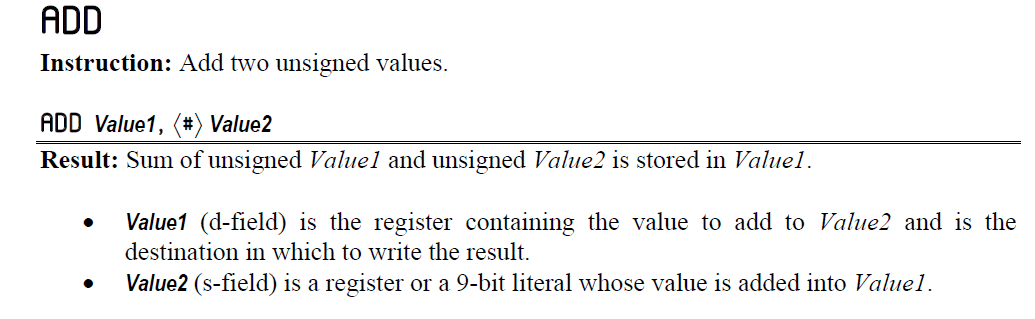
Changing the two datavar’s values:



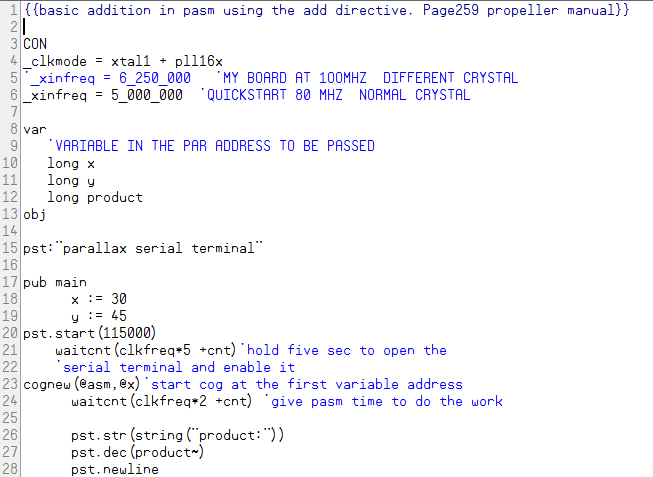
It works.

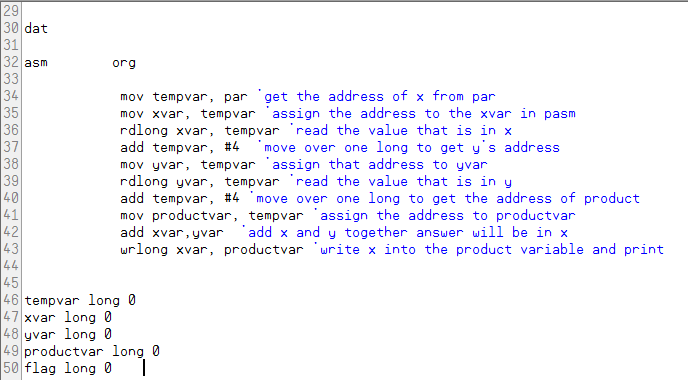
Now that we can get in and out of spin and pasm, I will present some examples of simple math.

I am trying to avoid the jump to really complicated programs with the assumption that the reader has a total comprehension of coding in assembly language of any type. I have found many tutorials do that.

These tutorials were good but confusing when they jump ahead and get very complex. Since I am a teacher, I teach flying and aircraft mechanics, I have to assess the background of each student. Academic learning can be difficult and painful, so if the instructor keeps it simple and explains the concept with easy examples that build up slowly, the student has a better chance of understanding and correlating the subject matter.

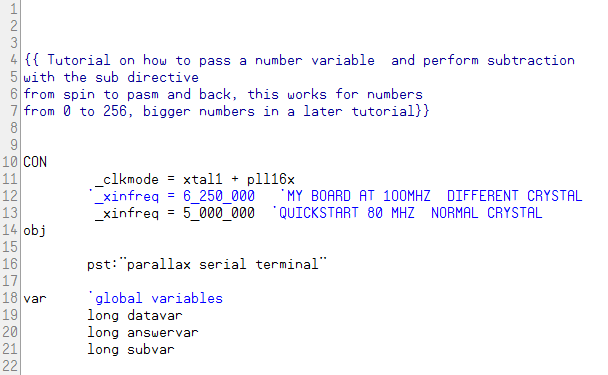
That results in a much better outcome. First addition, note the global variable name change. We are going to repeat the above code and make some changes:

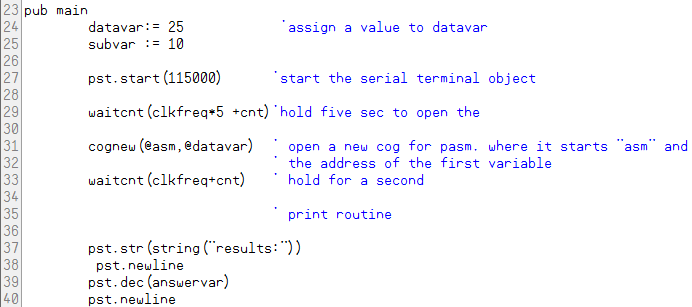


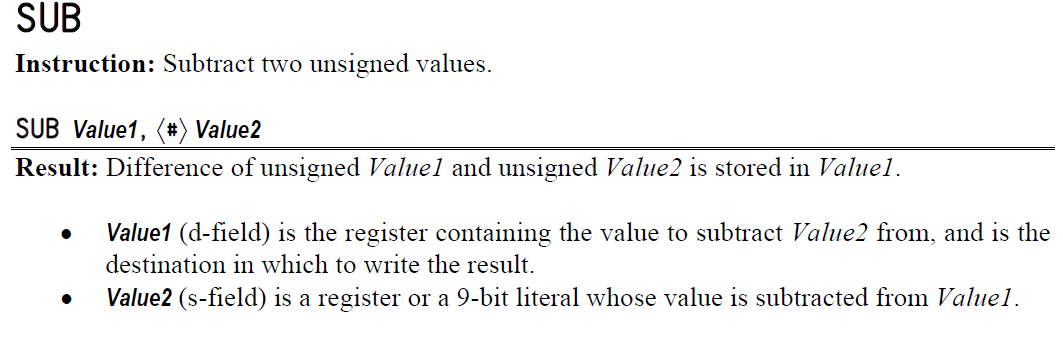


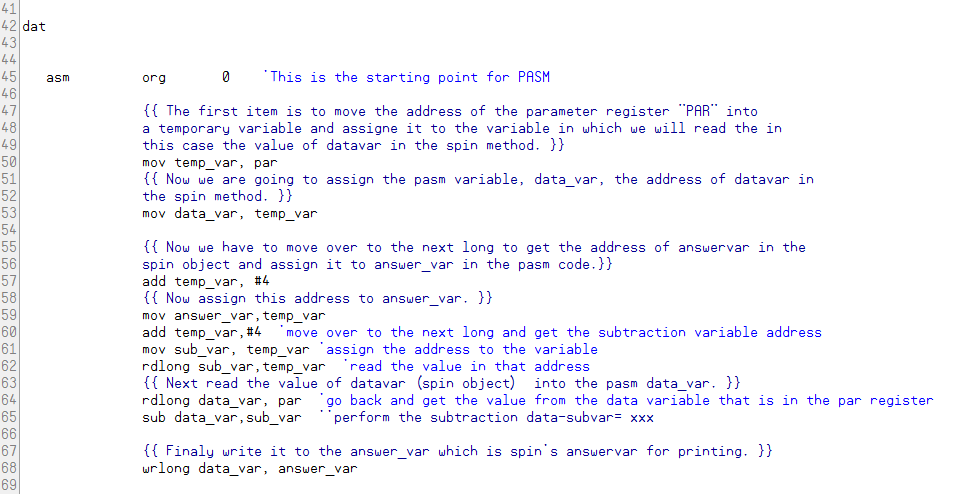


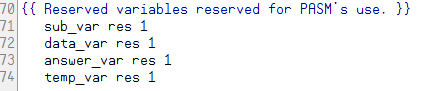
Subtraction:











What we have done is simply, at lines 60 and 61, added a new variable as well at line 71, these will be the subtraction variables. Next perform the subtraction and then write to our answer variable.

You should get this:



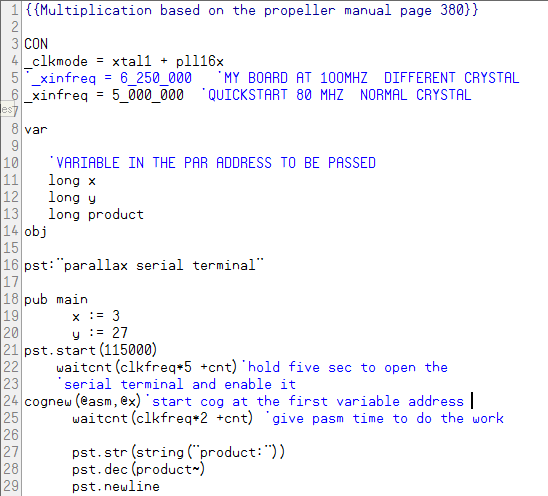
25-10=15

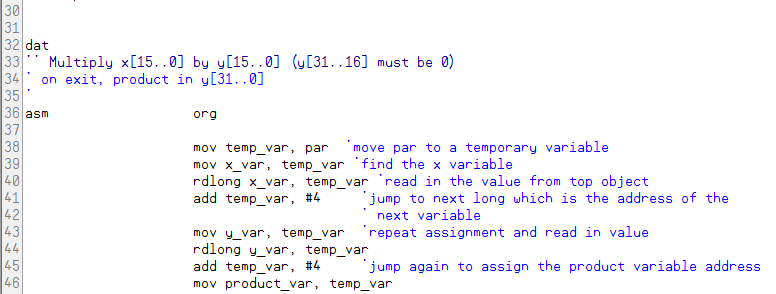
Change subtraction variable to 12.

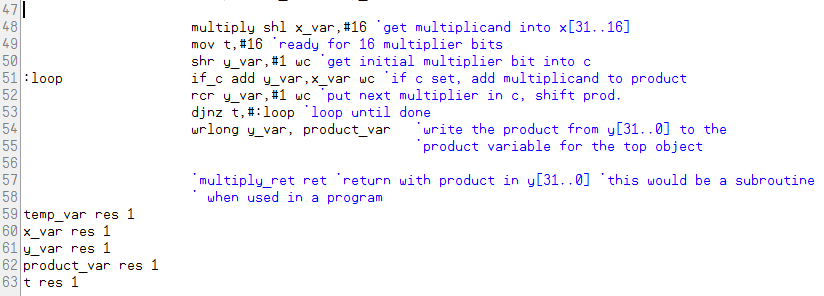


25-12=13

Multiplication this is from the propeller manual page 380:







3\*27=81

Change 27 to 9.

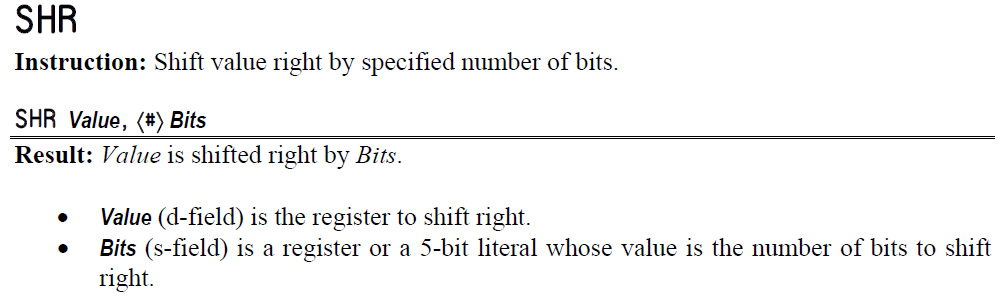
3\*9=27

Basically, we are doing multiplication by addition:

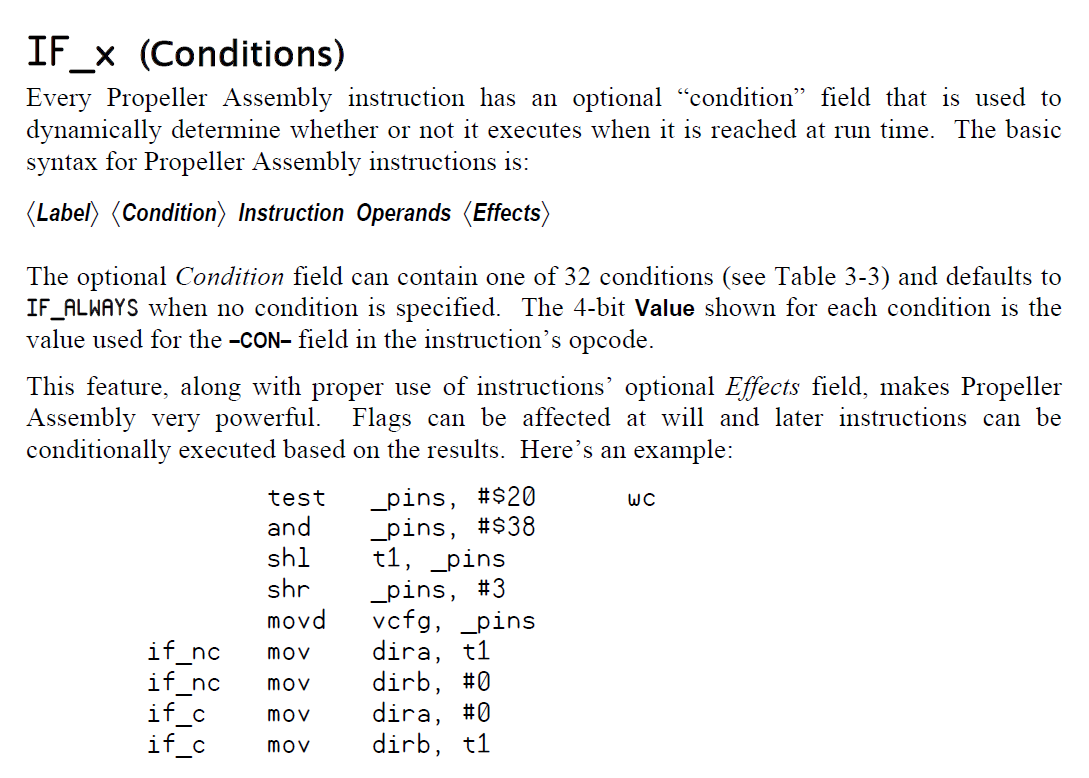
27+27+27=81

3+3+3+3+3+3+3+3+3=27

We run the loop until the carry flag is empty. This is repeated addition. Jeff and Dave at Parallax told me that there are many ways to do this. I am working on this myself. Basically, it is repetitive addition and that can be done in a loop until the number of iterations required are completed.

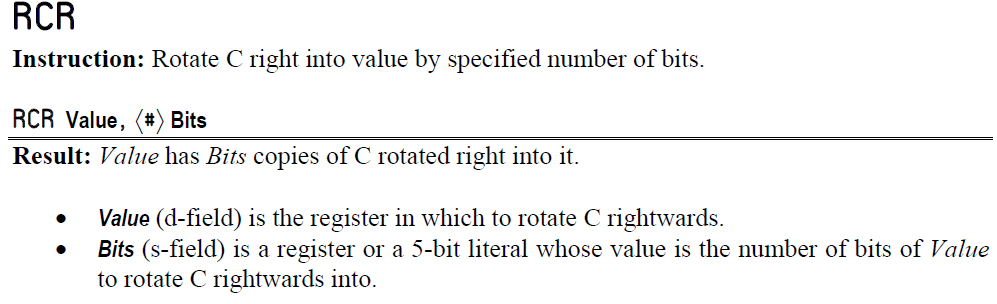


IF: conditional statements.

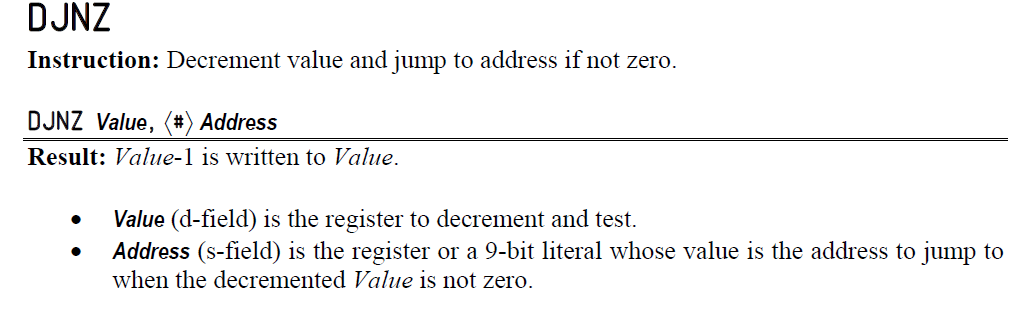


Please refer to the propeller manual for the full list of conditins.

RCR:

RCR allows the code to rotate the value of the carry flag into a variable. This is used to produce the product during each iteration of the multiplication process. Which in the end of the loop, would be the answer if one did multiplication via the addition process.

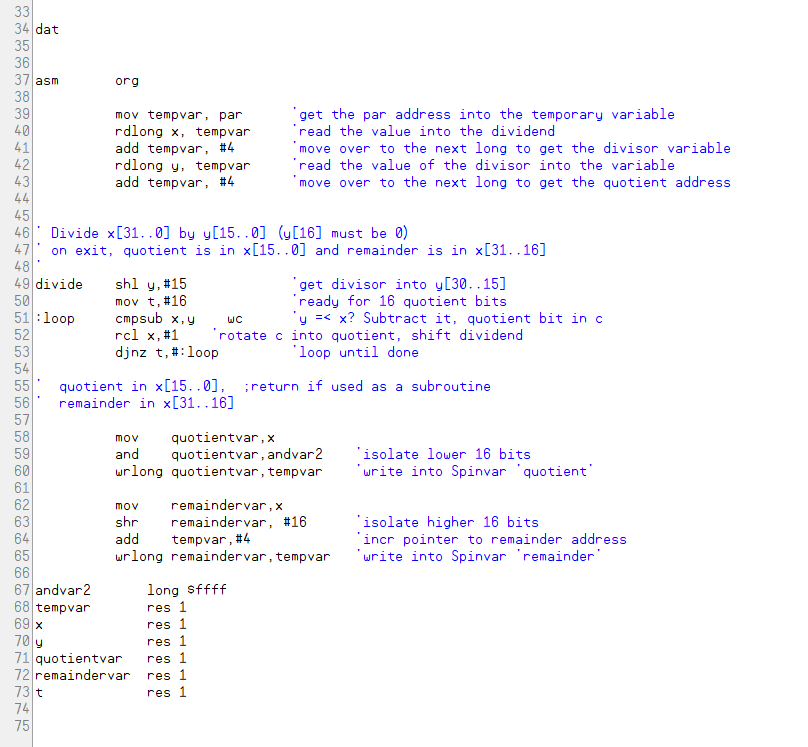
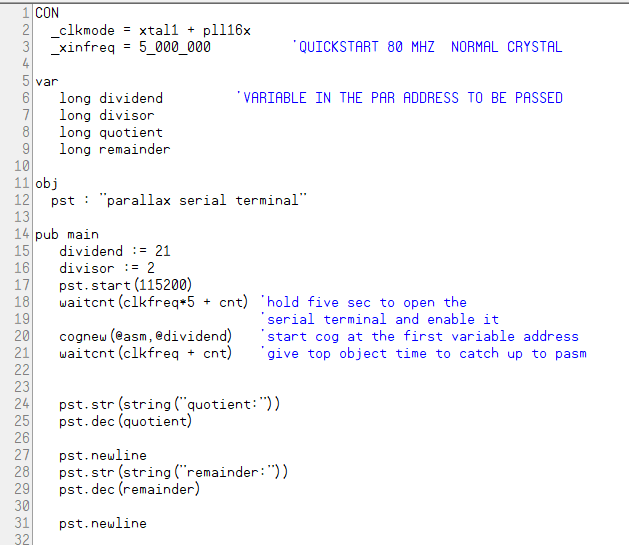
DJNZ:

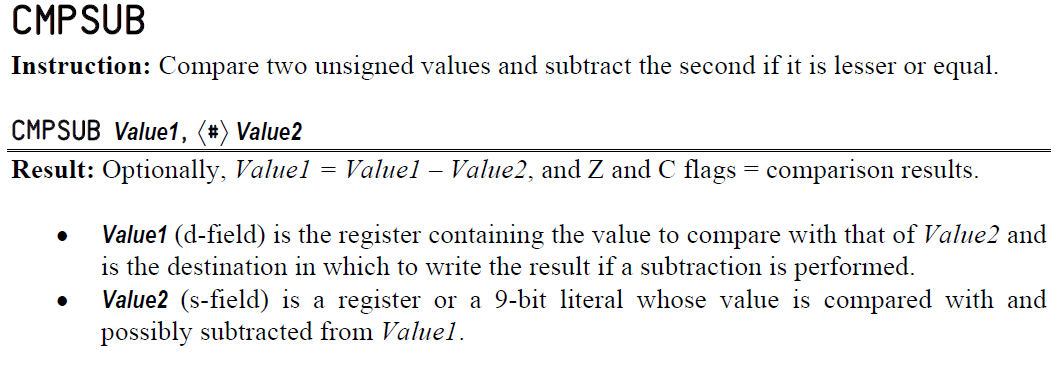


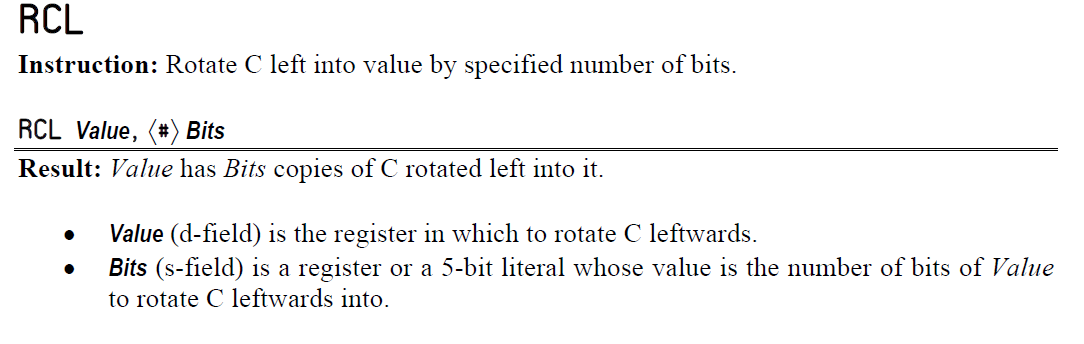
This directive allows for repetition while decrementing a particular value of choice and when the result is not zero jump to a particular point in the code until the result is zero. At that point the code will drop down to the next instruction in line.

SHR: There is a shift right and shift left these are self explanatory in the propeller manual as shown. The code will shift left or right by the number specified.

Division:







Counting up and down:

