



Scaling using $y=mx+b$ - a "no-math" shortcut approach - (part 2 of 2)

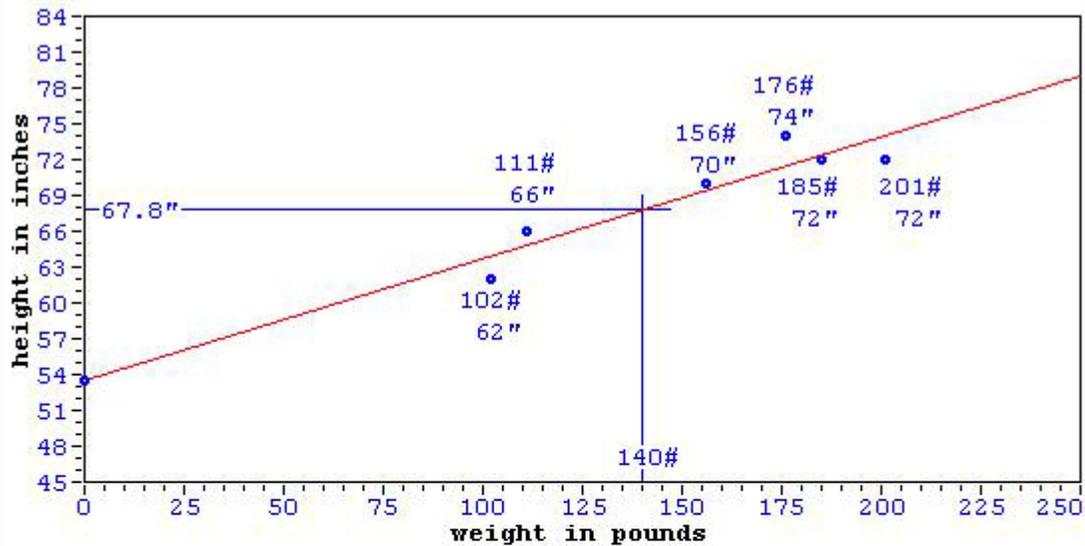
(discussion continued from part 1) ...

earlier in this discussion I told you how I was introduced to " $y=mx+b$ " by a fat man who was my very first math teacher when I started back to school as an adult ... he was a nice man and he certainly knew math ... the only fault that I found in his approach was that he never told me what the math was used for ... quite possibly he didn't really know ... so everything that I learned from him was rather abstract ... and I quickly forgot most of it as soon as the final exam was over ...

time marched on and the tech school that I was attending hired me as a full-time lab technician ... now I was getting paid to learn PLCs and had access to more neat toys than I knew what to do with ... eventually I started working with analog signals and "scaling" started to rear its ugly head ... my usual approach to scaling problems was the same one used by many untrained technicians working today ... basically I'd just keep plugging in one number after another until I finally got close enough to make the system work ... hunt-and-peck ... trial-and-error ... whatever you want to call it ... I was "getting by" but I knew that sooner or later I'd have to deal with this scaling subject ...

and then I took a required class in statistics ... the instructor for that one was a trim little lady ... you could have sold tickets just to watch her erase the board ... I paid attention in statistics class ... now the little instructor lady never mentioned " $y=mx+b$ " and frankly I didn't miss it one bit ... but one night she started teaching us about "2-variable statistics" to use for "linear regression" ... whatever the heck that was ... and here's a figure similar to the one she started putting on the board ...

Figure 11 - Using 2-variable statistics for linear regression



based on the six data points entered above ...

$m = 0.101997277$

$b = 53.5067558$... so ...

$y = 0.101997277x + 53.5067558$... or ...

$\text{height} = 0.101997277 * \text{weight} + 53.5067558$

using linear regression for a prediction, a student who weighs 140 pounds should be 67.8 inches tall ...

and incidentally, the correlation coefficient = 0.917686533

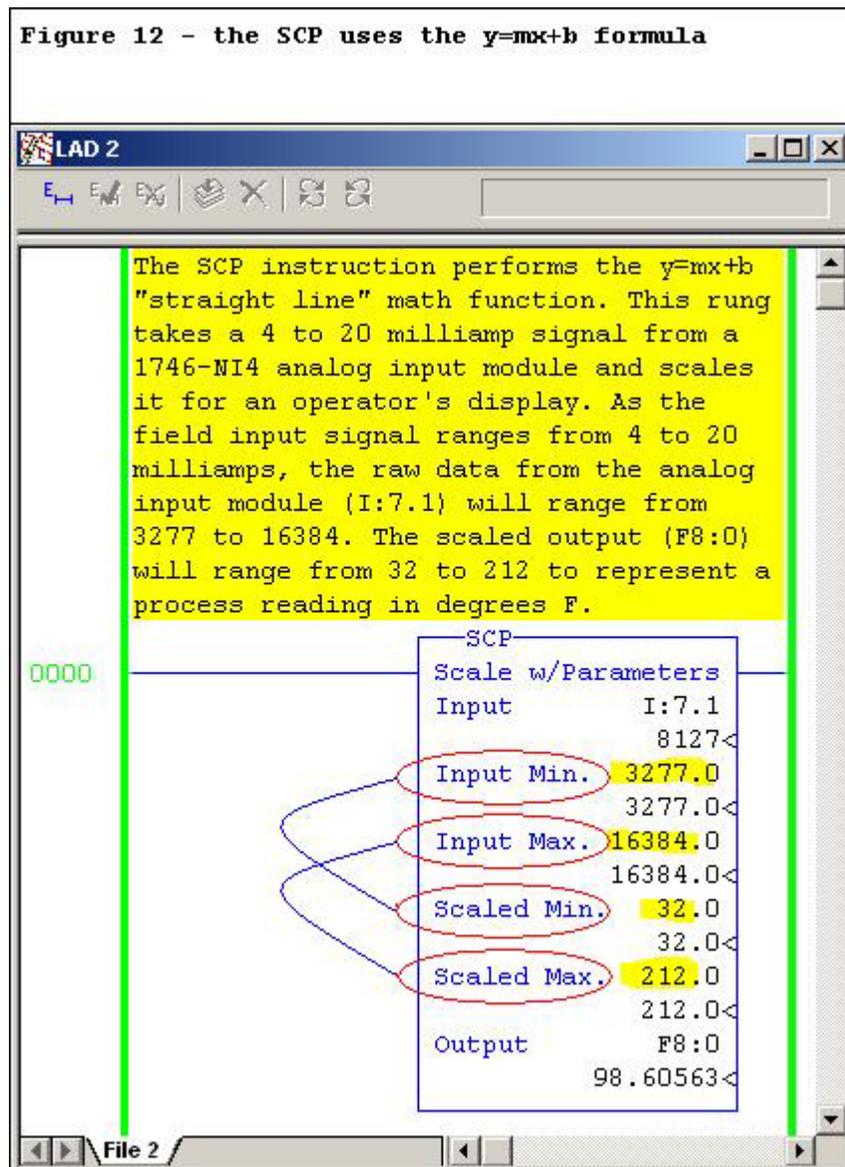
she went around the room asking students for their weight and then for their height ... she found the weights on the x-axis of the chart ... and the heights on the y-axis ... and she plotted a data point where each student's weight and height converged ... similar to the six data points that I've sketched on Figure 11 ... then she said something like this:

"now we're going to use our TI-36X calculators in the 2-variable statistics mode to enter the x and y coordinates of each data point ... once we've done that, then we'll have the calculator come up with a straight line which will best fit all of the data ... the calculator will automatically give us the "Slope" of the line ... and it will automatically give us the "Intercept" where the line crosses the y-axis" ...

WHAM! ... the light bulb in my head lit up! ... this was the same old "y=mx+b" thing that I had learned (and forgotten) about a year ago ...

when I finally tuned back in again, the cute little lady was telling us how "linear regression" would allow us to statistically predict either one of the two variables when given the other one ... so I could use the TI-36X calculator to take a number from the x-axis (think "raw input data") and automatically calculate a corresponding value on the y-axis (think "scaled value") ... in one moment of crystal clarity, all of my PLC scaling issues were solved ... this event is what is known as an "epiphany" ... and that's how I came up with the "Ron-Beaufort-no-math" shortcut approach to solving linear scaling problems ... and I've been using it - and teaching it - ever since ...

now one more figure and we'll be done for today ... many of Allen-Bradley's processors have the SCP (Scale with Parameters) instruction available ... secret handshake: the SCP is just "y=mx+b" in disguise ...



here we have an SCP which is set up exactly like the "degrees F" scaling exercise that we covered in Figure 8 ... notice that the "Input" entries

are 3277 for the minimum and 16384 for the maximum ... and the "Scaled" entries are 32 for the minimum and 212 for the maximum ... I've added the little lines on the left to show how "this value" is related to "that value" ... and these are exactly the same numbers that I entered into my TI-36X calculator earlier ... remember? ...

```
3277 ... [flip/flop] ... 32 ... [sum] ...  
16384 ... [flip/flop] ... 212 ... [sum] ...
```

once the coordinates of the two known data points were entered, I was able to scale the raw input data reading ...

```
8127 ... [2nd] [y'] ...
```

and the calculator displays "98.60563058" ... the same "output" as the SCP ... so basically my shortcut method turns my TI-36X into an Allen-Bradley SCP instruction ... but my SCP is on steroids ... because now I can easily obtain the "m/Slope/Rate" of the line ... and also the "b/Intercept/Offset" point at which the line crosses the y-axis ... these are values which the SCP doesn't display ... but it still must solve for both of these values internally each and every time the processor executes the instruction ... and that takes up scan time ...

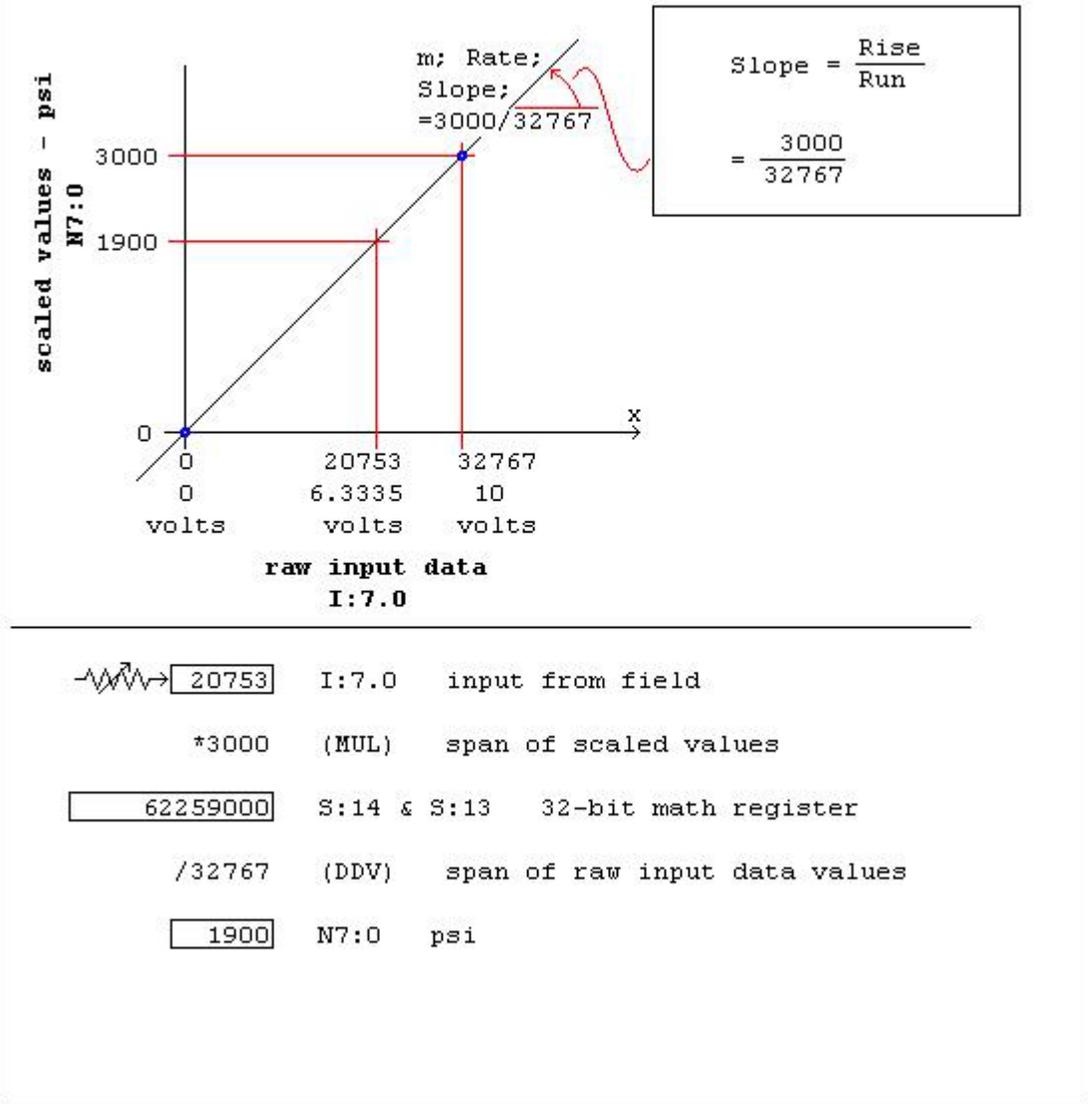
so I can (if I choose) easily replace this scan-time hog of an SCP with a much more efficient set of math functions ... one function to multiply by the "m/Slope/Rate" value ... and another function to add the "b/Intercept/Offset" value ... and of course both of these values can be easily obtained from my trusty TI-36X calculator with just a few simple keystrokes ... and with absolutely NO MATH! ...

so far we've been concentrating on techniques for analog scaling for SLC-5/02 processors on up ... these are the ones that have the SCL instruction available ... now let's tackle the subject of doing scaling in an SLC-5/01 system which doesn't even have the SCL instruction ...

so here is a technique for scaling which uses only "standard math" as the book calls it - and no floating point numbers ... note that the book considers the DDV (Double Divide) instruction to be a "standard math" function ...

a good way to start this discussion would be to take another look at Figure 7 ... in the solution to that example, I calculated the value for "m/Slope/Rate" as 0.091555528 by dividing the "Rise" (3000) by the "Run" (32767) ... and that was fine for the method that we were using in that particular discussion ... but now let's go a little bit deeper ... Figure 13 gives an example of another way of skinning the cat - one which will work with the SLC-5/01 processor that doesn't have the SCL instruction available ...

Figure 13 - Scaling without SCL



here we have exactly the same problem that we had back in Figure 7 ... specifically, a 0 to 10 VDC input signal from the field gives us raw data readings which range from 0 to 32767 ... we want to convert the raw data into scaled values which range from 0 to 3000 (psi) ... the biggest hurdle to getting that done is that the processor that we're working with doesn't support floating point math ... nailing that issue down: we don't have anywhere to store the "m/Slope/Rate" value (0.091555528) in the PLC's memory ... so let's take the specific example given in the figure and see how we might proceed with a solution ... specifically, let's come up with a scaled value to go with a raw input data signal of 20753 ... note that following operation is outlined step by step at the bottom of Figure 13 - and the program rungs will be shown later in Figure 14 ...

first we'll multiply the raw input data reading (20753) by the span of the scaled values (3000) and specify N7:0 as the destination of the MUL instruction ... mathematically the result "SHOULD BE" 62259000 ... but we have an "issue" with that great big old number ... the problem is that a

number that large will not fit into the 16-bit integer location N7:0 ... and so the processor (bless its little heart) does the best that it can ... it stuffs N7:0 with the largest number that it will hold (32767) and it sets the "Math Overflow" bit (S:5/0) to let us know that it ran into a math problem along the way ... and then the processor also does something else which is quite useful for our purposes ... it moves the full result of the multiplication into the 32-bit "Math Register" ... specifically, after the multiplication, words S:14 and S:13 (taken together) will contain the number 62259000 ... now if we only had a way to divide that 32-bit number by 32767, we'd have made significant progress toward our final solution ... and luckily the DDV (Double Divide) instruction lets us perform exactly the division operation that we need ... so we'll use a DDV to divide the value stored in the 32-bit math register (62259000) by the span of the raw input data (32767) ... and we'll specify N7:0 as the destination of the division ... bingo! ... now N7:0 contains the result of our scaling operation (1900) ...

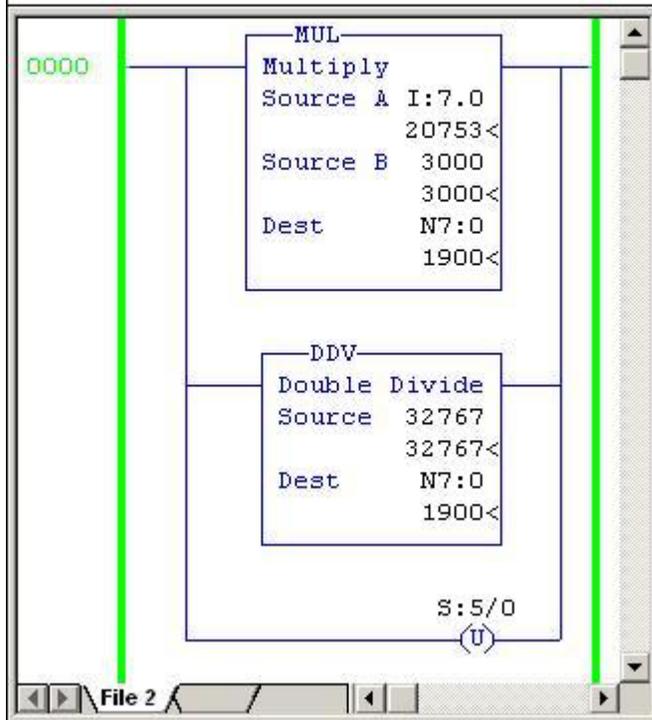
a quick recap: usually our first step in handling this example scaling operation would be finding the "m/Slope/Rate" value - by dividing the "span" of the scaled values (3000 in this example) by the "span" of the raw input data values (32767 in this example) ... but that approach would result in a floating point value (0.09155528) that our limited processor could not store ... so generally we would "work around" this "no-floating-point" limitation by using an SCL instruction ... the SCL would allow us to first multiply the original "m/Slope/Rate" number by 10000 before typing it into the SCL's "Rate" entry ... BUT ... the SLC-5/01 processor doesn't have this SCL instruction available ... and so we need another type of "work around" to help us with our scaling ...

so instead of DIVIDING first and then MULTIPLYING second, we structured our program to MULTIPLY first and then DIVIDE second ... doing the math in that reversed "order of operations" gives the same final result but it allows us to make use of the 32-bit "Math Register" during our DDV (Double Divide) operation ...

in a nutshell, the same techniques for the "Slope of the line" equals the "Rise" over the "Run" that we covered earlier in this thread still apply even when you don't have an SCL available ... and the sketch of the graph that we've been using doesn't have to change - EXCEPT for one minor thing ... notice that the top of the graph in Figure 7 shows the value of "m/Slope/Rate" as a decimal number (0.09155528) ... but the top of the graph in Figure 13 shows the value of "m/Slope/Rate" as a FRACTIONAL number (3000/32767) ... that pretty much demonstrates the only difference in the approach ...

and here in Figure 14 we have a program rung which would handle the scaling operation that we just discussed ... compare this with the SCL approach used in Figure 1 ...

Figure 14 - Scaling without SCL



note that occasionally a program may require some type of "range limiting" operation located just above this "scaling rung" ... this approach would keep any weird "out of range" numbers from exceeding the capabilities of the math operations ...

and definitely notice the unlatch instruction which turns off the "Math Overflow" bit S:5/0 immediately after the scaling is complete ... this is required because of the "overflow" condition (integer greater than 32767) which results after the MUL step ... if this bit is not turned off before the end of the scan, a processor fault will occur ...

incidentally, the Destination (N7:0) is used as a "dummy" place holder in the MUL step ... specifically, we don't care what value the MUL places in its Destination ... all we care about is the value which the MUL will simultaneously place in the 32-bit "Math Register" ...

and for those who aren't familiar with the DDV, here's a tip that might help get you started ... many people are confused when they see only one source (32767 in our example) being specified in the DDV ... the fact is that the DDV's single "Source" entry specifies "the-number-we're-dividing-BY" (in other words, the "divisor") ... we don't have to specify "the-number-that's-being-divided" (in other words the "dividend") because the DDV only (and always) divides just ONE thing - and that one thing is always the value stored in the 32-bit "Math Register" (words S:14 and S:13) ...

hint: that's why the value stored in N7:0 by the MUL step is a "dummy" value ... we weren't planning to use that value later anyway ... what we really wanted was just the value simultaneously stored by the MUL in the

32-bit "Math Register" ... but since every MUL must have a valid "Destination" entry, we just used N7:0 as a temporary "dummy" placeholder ... then in the very next step, we overwrote the "dummy" value in N7:0 with the results of the DDV operation ... at this point N7:0 can no longer be considered a "dummy" ... now (at least until the next processor scan comes along) N7:0 holds the scaled value that we've been working for ...

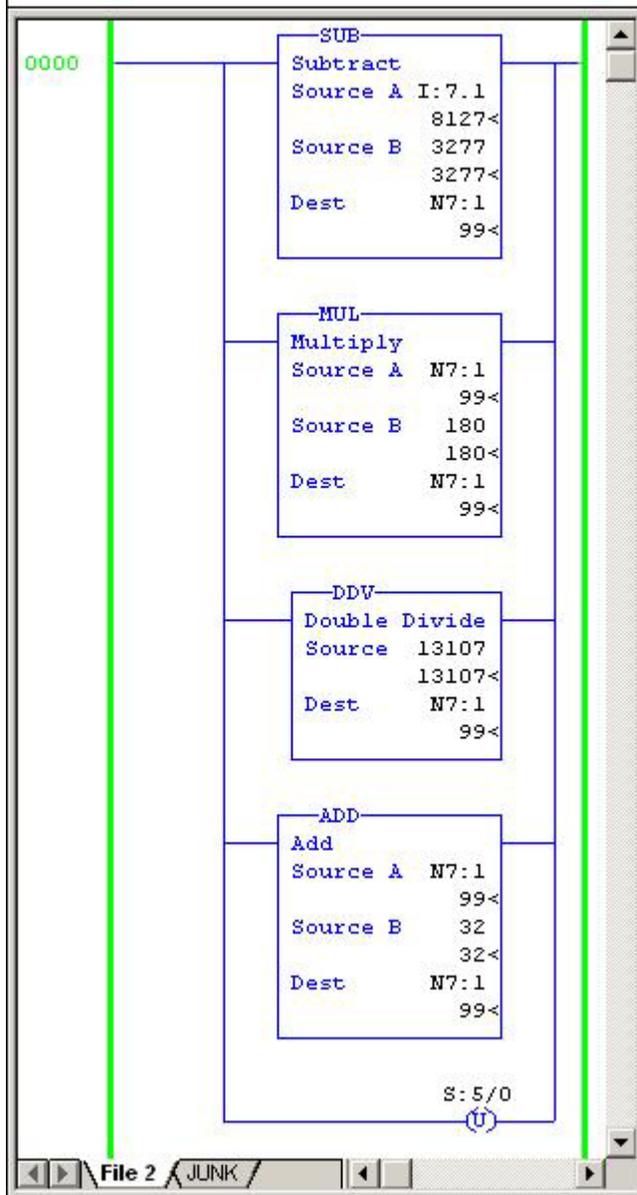
now we'll go a little bit deeper into this "Standard Math" approach to scaling ... and this example will have a little more meat on its bones ... specifically, we'll have an "offset" for our raw input data range - and another "offset" for our scaled values range ... this is exactly the same example that we worked through "longhand" for Figure 8 ... remember all of the hoops that our crusty old-timer had to jump through to avoid using "y=mx+b"? ... well, the hoops are back ... and remember (or go back and check) that one of those hoop-jumping steps resulted in a decimal number (0.370031281) ... that's an "issue" ... because if we're working with an SLC-5/01 system, we simply don't have anywhere in the processor's memory to store such a floating point value ... and so our "work around" will be the same "reverse-the-order-of-operations" trick that we used for the math in our last example ... specifically, we'll multiply first and then we'll divide - rather than the other way around ... the two things that we really have to watch are those pesky offsets ...

the processor stuffs the maximum value (32767) into N7:1 ... and sets the "Math Overflow" bit (S:5/0) to let us know that "something went wrong" with a math step ... and (most important) the processor simultaneously stores the full results of the multiplication step (873000) in the 32-bit "Math Register" at S:14 and S:13 ... next we'll do a DDV (Double Divide) operation ... this will take the value stored in the 32-bit "Math Register" (873000) and divide it by the span of our raw input data values (13107) ... the result (67) is stored at N7:1 ... finally, we add the offset of our scaled values (32) to the results stored at N7:1 (67) ... and that gives us our final answer (99) as "degrees F" stored at N7:1 ... and that's as close to 98.6 degrees as we can get with this particular system ...

quick summary: this is scaling for the SLC-5/01 processor with no floating point ... and no SCL workaround ... just "Standard Math" ... but to take care of the "m/Slope/Rate" value, be sure to multiply first ... and then use double-divide ...

Figure 16 shows a ladder logic program which will perform the scaling conversion that we just discussed ...

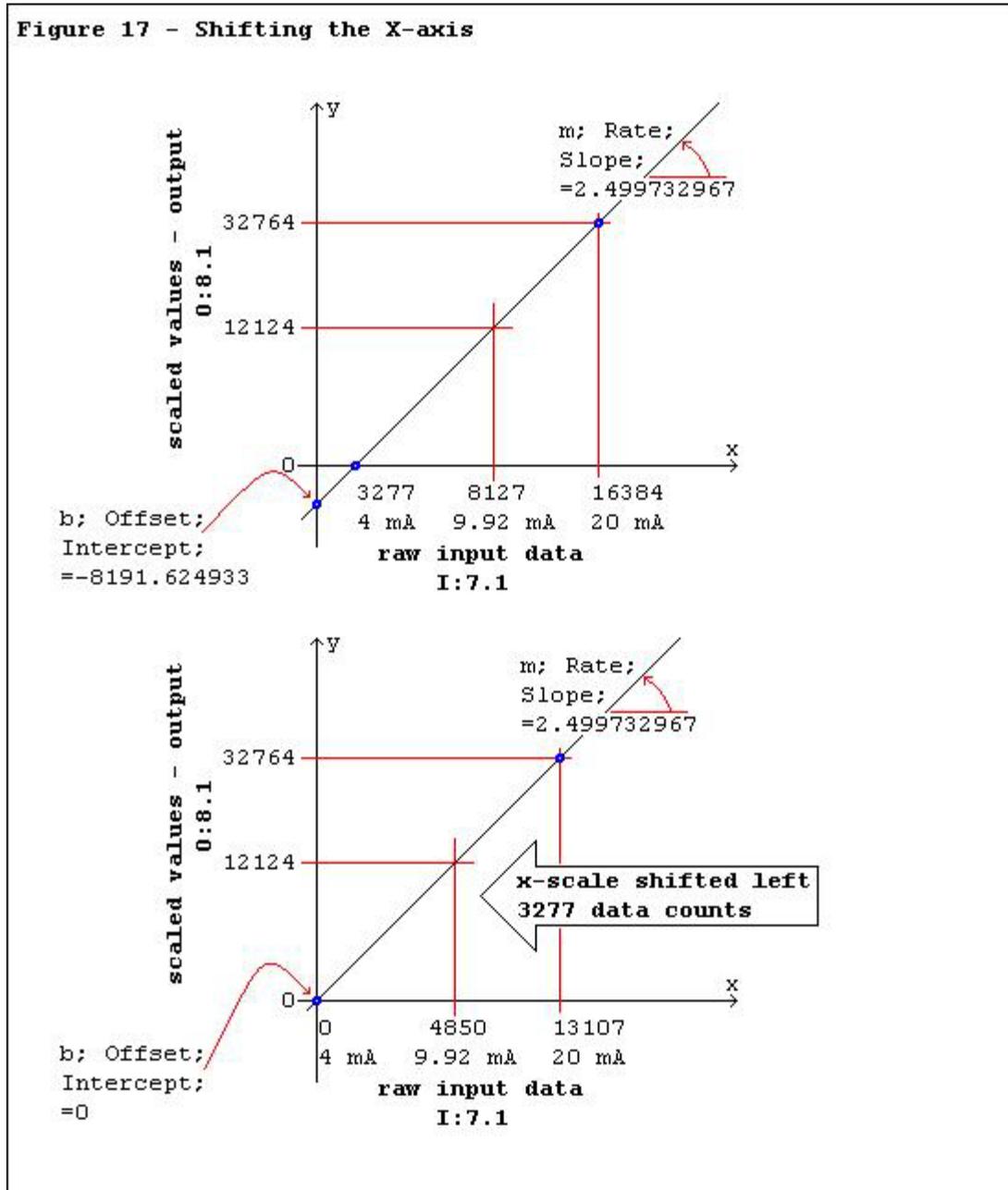
Figure 16 - scaling without SCL



and as I mentioned earlier, programs like this one often need some type of "limiting" arrangement to keep the input values from getting out of hand ... one good programming habit to get into is to check the operation of your math functions by feeding in the extreme limits of what the input values might possibly be ... and don't forget to check with extreme values of negative numbers too ...

next we'll take a look at a technique that often comes in handy when working with the SCL instruction ... in some cases, the ranges of values involved in a scaling operation exceed the capabilities of the SCL ... for a preview example: consider a situation where the value of "m/Slope/Rate" is 2.4997 ... now suppose that the input to the SCL is greater than 13108 ... in that case, the SCL would generate an internal value of something

greater than 32767 ... oops! ... greater than 32767 is not allowed and so a "Math Overflow" bit S:5/0 will be set ... and if it's not turned off again by the end of the scan, then the processor will fault ... and even if it IS turned off to prevent a fault condition, the math might not be "quite right" because the actual value could not be accurately stored ... in some applications, math which is not "quite right" can be a very bad thing ... you need to keep your eye on issues like this ...



the top of Figure 17 shows a situation where a 4 to 20 mA analog input signal is being used to control a 0 to 10 VDC analog output module ... let's solve this by using my no-math shortcut method on our TI-36X calculator ...

3277 [flip/flop] 0 [sum] ... 16384 [flip/flop] 32764 [sum] ... [2nd] [SLP] gives 2.499732967 for "m/Slope/Rate" ... [2nd] [ITC] gives -8181.624933 for "b/Intercept/Offset" ... and now for our example raw input data reading ... 8127 [2nd] [y'] gives a converted scaled value of 12123.70489 ... so we'd take this number and "round off" to 12124 as shown in Figure 17 ... and that's what we'd expect to see when the SCL does the scaling conversion for us ... but that's not quite the results we get ... actually the SCL will give you a scaled value of 12123 for this example ... now that's just one point lower than the 12124 that the TI-36X told us to expect ... but why the difference? ...

keep in mind that the values shown in Figure 17 are correct for working with the calculator ... but ... remember that the PLC handles this type of math using only integer numbers ... specifically, there is a certain amount of inconsistency to be expected due to "round off" issues ... in other words, we can accurately calculate "m/Slope/Rate" to be 2.499732967 as shown in the figure ... but when you use that value for the SCL instruction's Rate entry, the closest that you can come to the calculator's number is 2.4997 ... (entered as "24997/10000") ... so the PLC loses a little bit of accuracy ...

likewise ... we can accurately calculate "b/Intercept/Offset" to be -8191.624933 as shown in the figure ... but when you type that value into the SCL instruction's Offset entry, the closest that you can come to the calculator's number is -8192 ... so the PLC loses a little bit more accuracy ... so don't be surprised to see minor inconsistencies between what your TI-36X calculator says and what the PLC actually produces ... again, there are differences between the ways that the two devices handle "round off" issues ... fortunately those types of issues are usually minor "nickel-dime" stuff that won't significantly affect the operation of your system ...

but this next issue isn't "nickel-dime" folks ... this one is a major issue ...

suppose that you're the programmer who's setting up the scaling shown at the top of Figure 17 ... you've decided to use an SCL for this program ... you've calculated the value for "m/Slope/Rate" (2.499732967) and cranked it into the SCL as a Rate entry of 24997 ... and that's fine because it's as close as we can get with the system we're working on ... you've also calculated the value for "b/Intercept/Offset" (-8191.624933) and cranked it into the SCL as an Offset entry of -8192 ... and that's ok too because it's as close as we can get with the system we're working with ... so now you're ready to check things out ... you dial in an input signal of about 9.92 mA ... the raw input data reading at I:7.1 goes up to 8127 ... the TI-36X calculator says that we should expect our SCL to convert this to a "rounded off" scaled value of 12124 ... we see 12123 at O:8.1 ... we're just one point off ... not too bad ... we knew we'd have a little bit of inconsistency due to "round off" ... so no problem ...

you keep cranking the input current up little-by-little and keep checking the scaled values - and the resulting output voltage ... everything is working fine ... so far ...

you're finally up to 13108 for your raw input data reading ... the calculator tells you to expect a scaled value of 24574.8748 ... the SCL gives you an actual scaled value of 24574 ... that ain't bad at all ...

and the output module's voltage is tracking right along ... let's go one more tiny itty-bitty step up ...

you finally reach 13109 for your raw input data reading ... the calculator has told you to expect a scaled value of 24577.37453 ... BUT ... the SCL goes completely nuts and gives you an actual scaled value of 32767 ... WIDE OPEN! ... all of a sudden you notice that the analog output module is cranking out a wide open value of 10 VDC ... I'm glad that we didn't have this thing wired up as the speed control signal to a motor drive just yet ... and if you didn't have that "unlatch" rung for S:5/0 at the end of ladder File #2, then the processor would have faulted ... but come to think of it, maybe a "shut-down-everything-fault" would have been a better course of action than suddenly driving an output signal wide open ... we'll have to think about that later on ...

but back to our nutty "wide open" signal problem ... this is NOT just a simple "round off" issue ... and there's nothing wrong with your math solution either ... the problem lies in the PLC's method of handling its internal calculations as integer values ... let's nail the problem down so that we'll know what to look for from now on ... and then we'll finish up for today with a couple of practical ways to fix it ...

here's the play-by-play ... consider what the SCL was doing when you had the raw input data simmering along at a reading of 13108 ... each time the SCL was executed, it took the "Rate" entry (24997) and divided it by 10000 and got an internal result of 2.4997 ... so far so good ... next it multiplied that internal number (2.4997) times the "Source" value from I:7.1 (13108) and got a result of 32766 ... still chugging along just fine ... next the faithful SCL added the "Offset" value (-8192) to the previous result (32766) and came up with a result of 24574 ... close enough to the 24575 that the calculator told you to expect ...

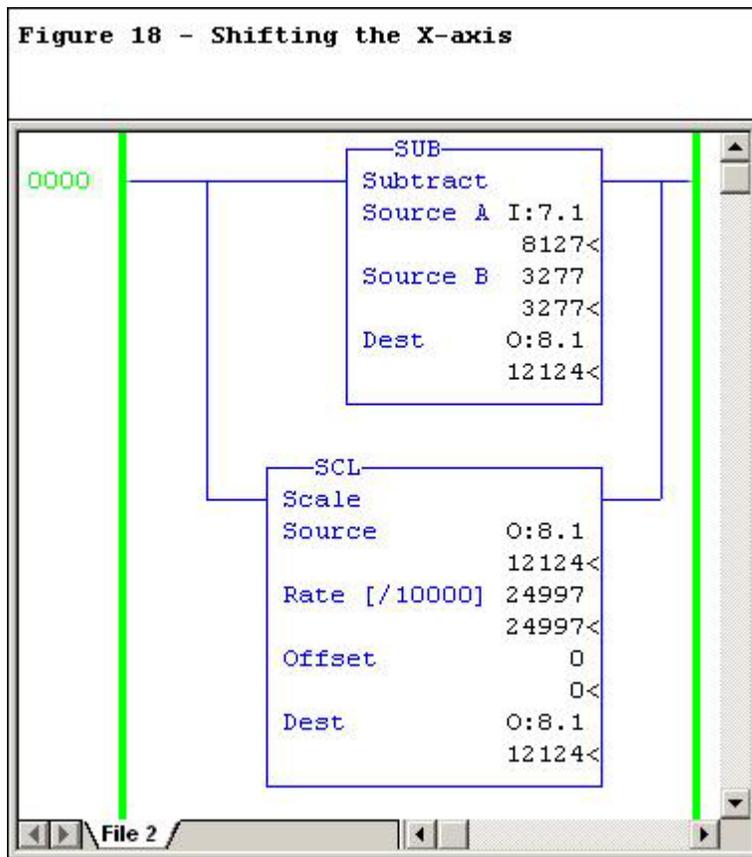
now consider what the SCL was doing when you finally cranked the raw input data signal up just one point higher to a new reading of 13109 ... the very next time that the SCL was executed, it took the "Rate" entry (24997) and divided it by 10000 and got an internal result of 2.4997 ... still ok ... next it multiplied that number (2.4997) times the "Source" value from I:7.1 (13109) and got a result of 32769 ... oops! you just broke the bank ... the integer-type math of the SCL can't store a number bigger than 32767 ... so it does the best it can (bless its little heart) and stores the maximum (32767) and turns on the "Math Overflow" bit (S:5/0) to tell you that something went wrong with the math somewhere along the way ... and that's about it ... once that internal overflow had been generated, the SCL just "gave up" and didn't even try to add in the "Offset" value (-8192) which would have brought us back within range with a final scaled result of 24577 ... so the sad story is, that even though it's been set up "right", this particular SCL will "hit the wall" when its raw input data reading comes in at 13109 or anything above ... from there on up, the scaled value just runs wide open at 32767 ... rats! ...

ok ... so now how do we fix this type of scaling problem? ... well, there are two common methods ... one uses the "standard math/MUL/DDV" approach ... we'll cover that one later ... another method keeps right on using the SCL instruction - but "shifts" the x-axis to prevent a math overflow ... and since we're already into SCLs, we'll look at that method first ...

take a look at the chart at the bottom of Figure 17 ... here we've "shifted" the entire raw input data range to the left on the chart ... basically we've subtracted the input range's minimum value (3277) from the minimum end (3277-3277=0) ... and from the maximum end (16384-3277=13107) ... and from the sample raw input data signal (8127-3277=4850) ... now notice that all of this "shifting" didn't affect the value of our "m/Slope/Rate" at all ... but the "b/Intercept/Offset" value is now a nice round even (and easy to deal with) 0 ... and even after we've "shifted" things around, our 9.92 mA sample input reading still converts to the expected 12124 corresponding scaled value ...

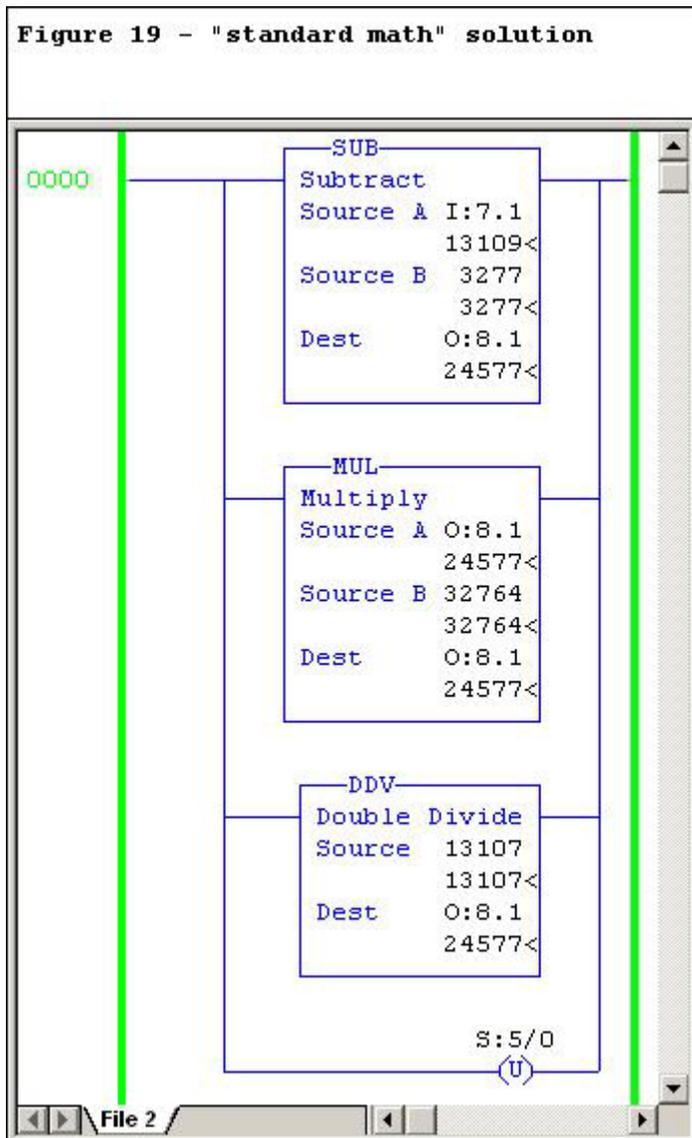
so now what about that "breaks the bank" raw input signal of 13109? ... well, look at the chart ... as long as the field signal doesn't exceed its 20 mA maximum, then our sensitive SCL won't ever see another raw input signal above 13107 ... and if you want some good advice, most systems should have this type of input signal "clamped" between a suitable maximum and a suitable minimum ... and you should consider programming suitable alarms to let someone know if the signals ever try to get out of range ...

Figure 18 shows one common approach to programming the "shifted" x-axis which we just discussed ...



and again, remember that this demonstrates a bare-bones minimum ... many systems require that the input signals be "clamped" and also possibly "alarmed" to insure safe, reliable operation ...

now let's take a look at the "standard math/MUL/DDV" approach to preventing that pesky "math overflow" problem ... and the basic program rung that we need is shown in Figure 19 ...



notice that first we subtract the "offset" (3277) from the raw input data signal (13109) ... that gives us an intermediate value (9832) which we'll store temporarily in our output address ... next we multiply the intermediate value (9832) by the span of the scaled values (32764) ... that gives us another intermediate value (322135648) ... we'll use O:8.1 as a "dummy" location this time - just so that we'll have a valid Destination for the MUL instruction ... actually the only location that we really care about right now is the 32-bit "Math Register" located at S:14 and S:13 ... and it's perfectly capable of handling that great big old intermediate number that our multiplication step shoves in there ... next the double-divide instruction takes the contents of the 32-bit "Math Register" (322135648) and divides it by the "span" (16384-3277=13107) of our raw input data range ... and we'll use our actual output address (O:8.1) to hold the final results (24577) ... and last but not least,

don't forget to "unlatch" the "Math Overflow" bit S:5/0 ... remember that it gets set whenever the "dummy" Destination of the MUL step "overflows" with any number greater than 32767 ... if you don't unlatch it before the end of the scan, then your processor will fault ...

hardball side trip: suppose that you have some type of "math" operation programmed on a rung located earlier in your program ... and suppose that the math in that rung somehow generated an "overflow" condition and set bit S:5/0 ... and suppose that you wanted (desperately wanted) that math "overflow" condition to "shut everything down" with a processor fault for safety reasons ... well, the unconditional "unlatching" of some of the "scaling" rungs shown in this discussion might really wreck your plans ... specifically, by simply turning the "Overflow Trap" bit off on each and every scan, the "unlatch" could defeat the fault condition that you were expecting ... as I've said before, the programming examples included in this discussion are samples - and samples only ... they're only intended to demonstrate the math concepts involved ... you'll have to decide for yourself how to apply these programming methods for the safe and reliable operation of your own applications ...

and so by using this "standard math" solution to our scaling problem, we didn't have to "shift" the x-axis to accommodate the SCL's limitations to "integer math" overflows ... and notice that the "standard math" solution was able to handle our "break the bank" problem input (13109) without even a hiccup ... and going further, it will also handle inputs all the way up through our expected maximum of 16384 ... with "no problems" due to math overflows ...

but once again, the "warning" ... you need to consider "clamping" and "alarming" your inputs to this (and to all) "scaling" operations ...

in closing, for more about this "scaling" subject, both the SLC-500 Instruction Set Reference Manual and the SLC-500 Analog I/O Modules User Manual have several examples that you might find helpful ... just be aware that there are MANY typographical errors in the "scaling" sections of both of these official books ... (even the guys at Allen-Bradley are human and we all drop a stitch from time to time) ... and there are also a few places in the books where unconventional "rounding off" can cause some confusion ... if you try working through these samples and have trouble getting the numbers to "work-out-right", then just double-check the book's solution against the numbers from your handy TI-36X calculator ... it should only take a second or so - and NO MATH! ...

best regards,
Ron