Consider the problem of making a digital voltmeter using the micro and an ADC:

X : Our 8 bit binary input
Y : Our scaled integer input

Last time:

$$Y = \frac{500}{256}X \qquad 0 \le X \le 255$$

This operation involved careful multiplication (8 bit X times 16 bit 500) and a shift to divide by 256.

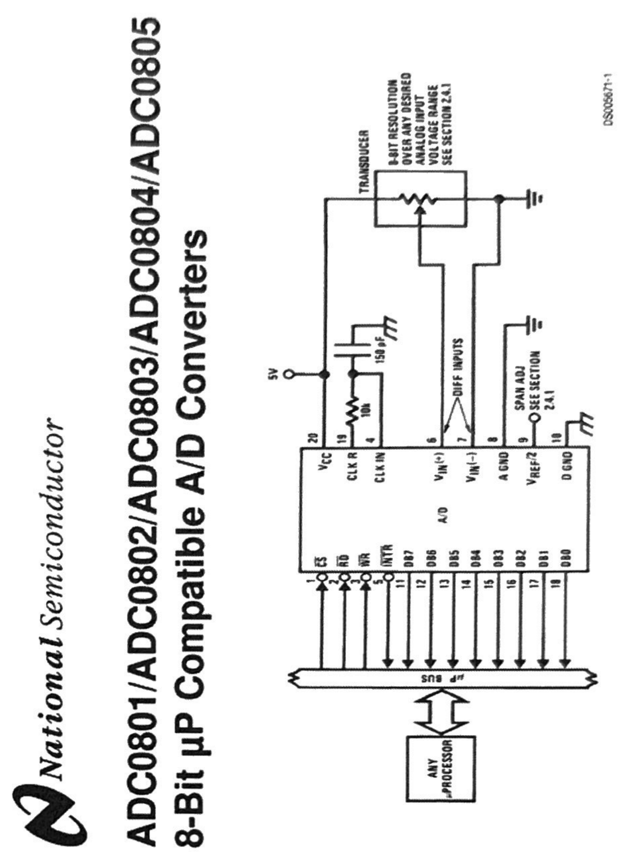
Suppose we want to average 100 measurements to computer the output Y. Now, X must hold a two-byte sum of 100 one-byte samples:

$$Y = \frac{500}{256 * 100}X \qquad 0 \le X \le 25500$$

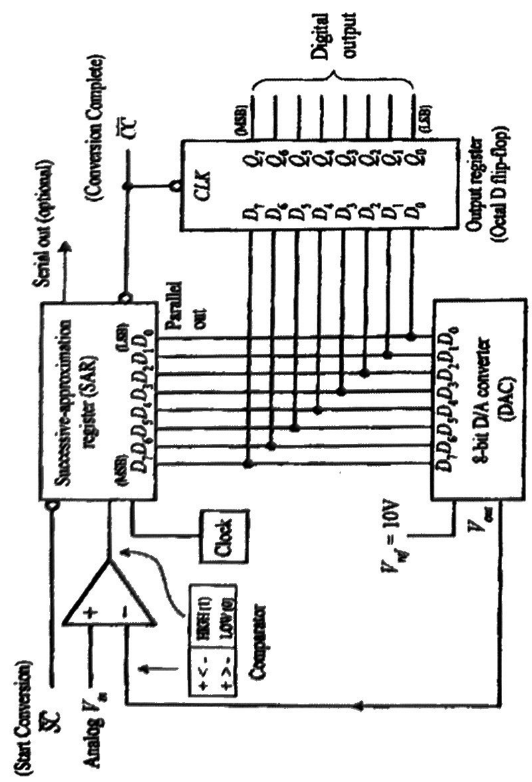
or

$$Y = \frac{1280}{65536}X \qquad 0 \le X \le 25500$$

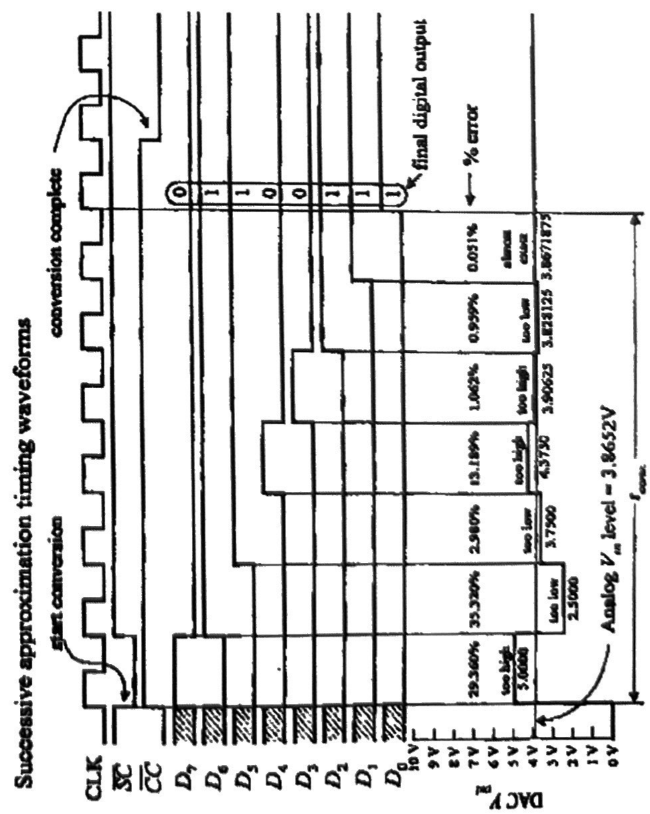
1



2



3



4

Binary	Unsigned	2's comp	offset	sign/mag	BCD	Gray
0000	0	0	-7	0	0	0000
0001	1	1	-6	1	1	0001
0010	2	2	-5	2	2	0011
0011	3	3	-4	3	3	0010
0100	4	4	-3	4	4	0110
0101	5	5	-2	5	5	0111
0110	6	6	-1	6	6	0101
0111	7	7	0	7	7	0100
1000	8	-8	1	-0	8	1100
1001	9	-7	2	-1	9	1101
1010	10	-6	3	-2		1111
1011	11	-5	4	-3		1110
1100	12	-4	5	-4		1010
1101	13	-3	6	-5		1011
1110	14	-2	7	-6		1001
1111	15	-1	8	-7		1000

- For non-negative numbers, unsigned binary is fine for all operations. What for overflow. Popular for address arithmetic.
- For addition/subtraction of signed numbers, 2's-complement is helpful.
- For multiplication/division of signed numbers, sign/magnitude format is often convenient.
- Other formats pop up in different applications. Some ADC and DAC chips use offset notation. Position encoders and other noise sensitive sensors sometimes use Gray coding.
- As we examine code today, not that registers names like X, XH, XL, Y, and XS stand for registers and bits YOU pick in the memory space!


```
TC2MS8
   MOV A, XT          ;Register XT hold signed 2com byte
   JB acc.7, Xneg     ;if bit 7 is 1, x is negative
   MOV XA, XT
   CLR XS             ;set the sign bit to 0, i.e., "positive"
   RET
XNEG:
   CPL A              ;x is negative, so find the
   INC A              ;positive version
   MOV XA, A          ;save the magnitude
   SETB XS            ;set the sign bit
   RET
```

How do we multiply an 8 bit number times a 16 bit number?

```
MUL8_16        ; byte in X, word in YH:YL. You pick these registers,
                 e.g., R1, R2, R3, etc
MOV a, X       ; load X in accumulator
MOV b, YL      ; load YL in B register
MUL A, B
MOV Z0, A      ; Z0 is register of your choice. Put low byte in Z0
PUSH B         ; push the result high byte. We'll add this to X*YH
MOV A, X       ; load x into accumulator again
MOV B, YH      ; put YH in B
MUL A, B       ; now compute X*YH
POP 0          ; get high byte of X*YL into R0
ADD A, R0      ; add low byte of X*YH to high byte of X*YL
                 NOTE! Carry may have been set!
MOV Z1, A      ; Put the sum in Z1
CLR A          ; Remember, B has high byte of X*YH. Clear A
ADDC A,B       ; Add C to high byte of X*YH. Result to A
MOV Z2, A      ; Save final result in Z2.
Ret            ; "Solution" (3 bytes) in Z0, Z1, Z2. Z2 is MSB
```


How do we multiply two 16 bit numbers (4 byte result)?
X = XH:XL, Y = YH:YL

Algorithm:

- First, multiply XL times Y. Yields ZL2, ZL1, ZL0 (msb to lsb).
- Second, multiply XH times Y. Yields ZH2, ZH1, XH0
- Compute four result bytes: Z3, Z2, Z1, Z0:

Z0 = ZL0
Z1 = ZL1 + ZH0 ; carry forward
Z2 = ZL2 + ZH1 ; carry forward
Z3 = ZH2

Decimal Adjust: A trick for performing decimal additions directly:

Example: Let 56d be represented by 56h!
Also, let 08d be represented by 08h. Try this:

```
MOV A, #56h
MOV R0, #08h
ADD A, R0
```

This program leaves 5Eh in the accumulator.

Now, if you add the command: DA A, the accumulator will contain 64h. The result of adding 56d and 8d if we interpret the result as two decimal digits! Neat! Essentially adds 5 to 59, or more accurate, 06h to the low nibble. See the INTEL spec book for more details.

How do we multiply two 16 bit numbers (4 bit result)?

```
MUL16:            ;word in XH.XL, word in YH.YL  (XL=X)
LCALL MUL8_16
MOV A, Z2
PUSH ACC          ;push ZL2
MOV A, Z1
PUSH ACC          ;push ZL1
MOV A, Z0
PUSH ACC          ;push ZL0
MOV XL, XH
LCALL MUL 8_16    ;mul XH times Y
MOV B, Z0         ;save ZH0 in B
POP Z0            ;recall ZL0
POP ACC           ;recover ZL1 in A
ADD A,B           ;add ZH0 and ZL1
MOV B, Z1         ;save ZH1 in B
MOV Z1, A         ;ZH0+ZL1 > Z1
POP ACC           ;recover ZL2 in A
ADDC A, B         ;ZL2+ZH1+C > A
MOV B, Z2         ;save ZH2 in B
MOV Z2, A; store 3rd res byte in Z2
CLR A
ADDC A,B ; ZH2+C > A
MOV Z3, A; store 3rd res byte in Z2
ret               ; done!
                  ; four result bytes
                  ; in Z0, Z1, Z2, Z3
                  ; (lsb to msb)
```