Loops vs unrolled loops

Often we got tought to unroll loops to save on the overhead a loop gives us by having to decrease a counter and involving another branch. But there are situation where a loop can perform way faster, as we can set up values directly via code modification. A good example is a line algorithm.

Here we need to subtract for e.g. dx from A and in case of underrun add dy to A and advance the x-position. On every change in y-direction we also want to plot.

This could look like:

```
back
         tax
         lda pix
         ora (dst),y
         sta (dst),y
         dey
         bmi out
         txa
         sbc dx
         bcs back

move_x
         adc dy

         asl pix
         bcc back

         tax
         lda #$80
         eor dst
         sta dst
         bmi back+1
         inc dst+1
         bne back+1
out
         rts
```

Now if we unroll the main loop, we would get:

```
back
         tax
         lda pix
         ora (dst),y
         sta (dst),y
         dey
         ...
         txa
         sbc dx
```

```
         bcs back
move_x
```

This means we would invest 25 cycles if we neglect the cycles needed for moving in x-direction. Now let us do the same as loop again, but let us set up dst, dx, pix and dy directly:

```
back
         tax
pix      lda #$00
dst1     ora $2000,y
dst2     sta $2000,y
         dey
         bmi out

         txa
dx       sbc #$00
         bcs back
move_x
```

As you see, all of a sudden we need 24 cycles per run, so the loop is faster! Why not setting up the immediate values within the speedcode you might think? Well, this means, that at a minimum, you waste another 4 cycles per loop run and value to be set up, while in our case we just waste an initial 4 cycles per value, what is pretty fair.

Even more, now the loop variant of our code gives us better access to illegal opcodes as some of them work with immediate values only, like the SBX command:

```
back
pix      lda #$00
dst1     ora $2000,y
dst2     sta $2000,y
         dey
         bpl out

         txa
dx       sbx #$00         ;now we get the value of A transfered to X for
free after subtraction
                          ;and A is free again for other purposes
         bcs back
move_x
```

We now end up with 22 cycles per run and just a few bytes of code. So as you see, sometimes it is also worth trying to optimize a loop before brainlessly unrolling everything 

Now as our code shrunk to a reasonable size, one could also think of copying that code to zeropage once and thus speed up the further code manipulation happening when setting up the loop and when executing the code in move_x.