4. **Don't use INC/DEC on P4**

   On the P4 use ADD/SUB in place of INC/DEC. Generally it is faster. ADD/SUB runs in 0.5 cycles. INC
   1 cycle.

5. **Rotating**

   Avoid rotate by a register or rotate by an immediate value of anything but a 1.

6. **Eliminate unnecessary compare instructions**

   Eliminate unnecessary compare instructions by doing the appropriate conditional jump instruction ba
   flags that are already set from a previous arithmetic instruction.

```
        dec     ecx
        cmp     ecx,0
        jnz     loop_again

;gets changed to
        dec     ecx
        jnz     loop_again
```

7. **LEA is still really cool, except for on the P4 it tends to be slow.**

   You can perform multiple math operations all in one instruction and it does not affect the flags regist
   can put in in between one register being modified and a flags comparison jump on the next line.

```
top_of_loop:

    dec   eax
    lea   edx,[edx*4+3]     ; multiply by 4 and add 3. Does not affect flags
    jnz   top_of_loop       ; so the next instruction doesn't get hosed.
```

   ADC and SBB.

   Most compilers don't really make good use of ADC and SBB. You can get good speeds ups with that.
   64-bit numbers together, or adding big numbers together. Keep in mind that on the P4 ADC and SBI
   As a work around you can use "addq", and use MMX to do this. So the second optimization suggestic
   to use MMX to do the adding or subtracting. You jueds107+ 6 4]1erform multip6ptimg oradd 10.0ndo the a
   ad deca efred+4]oop:

   ;o do tbo 4]2A ioprDocumthat So tamo ts So bebe 3A ioprDocumtgain

```
unsigned char c = (the_array[i])>>24) & 0xFF;
```

What if I can get rid of the SHIFT and the AND using assembler? That would save me 2 instructions.

```
        dec     ecx
        jnz     looper
```

We can accumulate the result in a register, and then only do one write to memory.

```
        mov     ecx,AMT_TO_LOOP
looper:
        xor     edx,edx                 ;zero out register to accumulate the result :
        movzx byte ptr eax,[esi]
        add     edx,eax
        movzx byte ptr eax,[esi+1]
        add     edx,eax
        movzx byte ptr eax,[esi+3]
        add     edx,eax
        add     esi,3
        mov     [edi],edx
        add     edi,4
        dec     ecx
        jnz     looper
```

16. **When to convert a call to a jump**

    if the last statement in a routine is a call consider converting it to a jump to get rid of one call/ret.

17. **Using arrays for data structures**

    (This is non-assembler related, but it's a great one). You can use an array for data structures such a linked lists. By using an array the memory ends up being contiguous and you get a speed up due to misses.

Advanced

Avoid prefixse an array for data st1T. 2 T an e2void prefixse (prefixse get generated for a 2.1nbre

```
        mov     edi,offset dst_arr  ;pointer to the destination array which has to be
                                    ; 16-byte aligned or you will get an exception.
looper:
        movdqa  xmm0,[esi]          ;works on P3 and up
        movntps [edi],xmm0          ;Works on P3 and up
        add     esi,16
        add     edi,16
        dec     ecx
        jnz     looper
```

Handle 2 cases per loop for MMX/SSE/SSE2.

nearest 4KB block. This one has a HUGE code example, so I put it last in case you fall asleep readin