

Welcome to yet another Mandelbrot Set explorer; this one is meant to be a fun-to-operate benchmark for current and future CPUs. There are code paths for x87 FPU, SSE, AVX, 4-operand FMA, and 3-operand FMA. The AVX paths are also split into 128 and 256 bit configurations.

Controls	
mouse left/right	zoom in/out
m/l	more/less depth (full redraw)
x/z	more/less depth (full redraw)
n/p	change palette (full redraw)
a	adjust max anti-aliasing level
t	reset window (full redraw)
r	change render path (full redraw)
s	toggle stats
b	move to preset benchmarking location (full redraw)
esc	exit and copy results to clipboard
shift-esc	just exit

The returned GFLOPS number is calculated from QueryPerformanceCounter and is highly dependent on the depth in the image as well as how much of the black space has been optimized out. No attempt is made at calculating the actual CPU frequency. The fractal algorithm for determining what color to use at the point $x_0 + iy_0$ is implemented as:

```

EscapeTime(x0_Real, y0_Real)
    local x_Real = x0, y_Real = y0, t[4]_Real;
    local i_Integer = 0, j_Integer;
    local R2_Real = 16.0;
Loop:  t[3] = x*x + y*y; x + I y = (x + I y)**POWER + x0 + I y0;
      t[2] = x*x + y*y; x + I y = (x + I y)**POWER + x0 + I y0;
      t[1] = x*x + y*y; x + I y = (x + I y)**POWER + x0 + I y0;
      t[0] = x*x + y*y; x + I y = (x + I y)**POWER + x0 + I y0;
      if t[0] > R2, goto Over;
      if (i+=4) > DEPTH, goto Max;
      goto Loop;
Over:  j = (t[3] > R2) + (t[2] > R2) + (t[1] > R2);
      return i - j - log(POWER, log(R2, t[j])) mod 512;
Max:   return Black;

```

This algorithm is vectorized into the ymmx registers and four vectors are handled per main iteration loop. If a point has overflowed or reached the maximum depth, then it is reloaded immediately and sent back to the main iteration loop as opposed to waiting for the rest of the points in the vector to overflow/maxout.

Register naming conventions are in PREF.inc

To modify a calculation path for more fun, testing or performance, find the main loop in it's .inc file. It should be straightforward from here if you keep in mind that

the four pairs of vectors are in ym0 - ym7. (ym0, ym1) is the first pair, ...ect

ym8 - ym9 are reserved for a possible future 5x unrolling (no benifit to this on AVX now)

ym10 - ym11 hold constants. ym12 also holds a constant on 3-operand FMA

ym12 - ym15 are free for calculations, except ym12 on 3-operand FMA

The paths are currently set as:

	QUADRATIC		eff.	CUBIC		eff.
	ym0	x		ym0	x	
	ym1	y		ym1	y	
	ym10	CONST 2.0		ym10	CONST 3.0	
	ym11	CONST R*R		ym11	CONST R*R	
				ym12	CONST 4.0 (3FMA only)	
SSE	movaps	xm15,xm10	88.2%	movaps	xm14,xm0	76.4%
	mulpd	xm15,xm1		mulpd	xm14,xm0	
	mulpd	xm1,xm1		movaps	xm15,xm1	
	mulpd	xm15,xm0		mulpd	xm15,xm1	
	mulpd	xm0,xm0		movaps	xm12,xm10	
	movaps	xm14,xm0		mulpd	xm12,xm14	
	subpd	xm0,xm1		movaps	xm13,xm14	
	addpd	xm1,xm14		addpd	xm13,xm15	
	addpd	xm0,m16[x0]		movapd	m16[t],xm13	
	movaps	m16[t],xm1		movaps	xm13,xm15	
	movaps	xm1,m16[y0]		mulpd	xm13,xm10	
	addpd	xm1,xm15		subpd	xm14,xm13	
				movaps	xm13,xm12	
				subpd	xm13,xm15	
				mulpd	xm0,xm14	
				mulpd	xm1,xm13	
				addpd	xm0,m16[x0]	
				addpd	xm1,m16[y0]	
AVX	vmulpd	ym15,ym1,ym10	89.7%	vmulpd	ym14,ym0,ym0	87.0%
	vmulpd	ym14,ym0,ym0		vmulpd	ym15,ym1,ym1	
	vmulpd	ym1,ym1,ym1		vmulpd	ym12,ym14,ym10	
	vmulpd	ym15,ym15,ym0		vaddpd	ym13,ym14,ym15	
	vsubpd	ym0,ym14,ym1		vmovapd	m32[t],ym13	
	vaddpd	ym14,ym14,ym1		vmulpd	ym13,ym15,ym10	
	vaddpd	ym1,ym15,m32[y0]		vsubpd	ym14,ym14,ym13	
	vaddpd	ym0,ym0,m32[x0]		vsubpd	ym15,ym12,ym15	
	vmovapd	m32[t],ym14		vmulpd	ym0,ym0,ym14	
				vmulpd	ym1,ym1,ym15	
				vaddpd	ym0,ym0,m32[x0]	
				vaddpd	ym1,ym1,m32[y0]	
4FMA	vmulpd	ym14,ym0,ym0	?	vmulpd	ym14,ym0,ym0	?
	vmulpd	ym15,ym1,ym1		vmulpd	ym15,ym1,ym1	
	vmulpd	ym1,ym1,ym10		vaddpd	ym13,ym14,ym15	
	vfmaddpd	ym1,ym1,ym0,m32[y0]		vmovapd	m32[t],ym13	
	vaddpd	ym0,ym14,m32[x0]		vfnmaddpd	ym13,ym10,ym15,ym14	
	vaddpd	ym14,ym14,ym15		vfmsubpd	ym14,ym10,ym14,ym15	
	vsubpd	ym0,ym0,ym15		vfmaddpd	ym0,ym0,ym13,m32[x0]	
	vmovapd	m32[t],ym14		vfmaddpd	ym1,ym1,ym14,m32[y0]	
3FMA	vmulpd	ym14,ym0,ym0	?	vmulpd	ym14,ym0,ym0	?
	vmulpd	ym15,ym1,ym1		vmulpd	ym15,ym1,ym1	
	vmulpd	ym1,ym1,ym10		vaddpd	ym13,ym14,ym15	
	vfmadd213pd	ym1,ym0,m32[y0]		vmovapd	m32[t],ym13	
	vaddpd	ym0,ym14,m32[x0]		vfnmadd213pd	ym13,ym12,ym15	
	vaddpd	ym14,ym14,ym15		vfmsub213pd	ym14,ym10,ym15	
	vsubpd	ym0,ym0,ym15		vfmadd213pd	ym0,ym13,m32[x0]	
	vmovapd	m32[t],ym14		vfmadd213pd	ym1,ym14,m32[y0]	