back

särki.ASM - 360 byte intro

särki.ASM - 360 byte intro explained

Four k intros are boring ? Too many bytes ? Now try this: 360 Bytes. Piru was kind enough to comment the source of his 360 byte demo.
Download the executable with source here

```
; Written by Harry "Piru" Sintonen sintonen@iki.fi.
; This source code is funware (read: freeware). Have fun with it.
; Freely exploitable for non-commercial purposes.
;
;
; Even more optimization, hints:
; - at program start d0=1 if no arguments are given.. :-)
; - look at y- and x-loops.. reversing y-loop would pay off. also it could be
;   possible to combine the loops to one.
; - reverse maxiter loop.
; - remove OpenScreen failure test.
; - replace lbm test with loop counter.
; - who cares if CloseLibrary() isn't called.
;
;
; known features (bugs) of the current version:
;
; - if run on 68000 or 68010 will crash, need 020+
; - if run on pre-3.1 (V40) will crash, need Kickstart 3.1
; - if actiview's viewport modeid & MONITOR_ID_MASK isn't valid modeid intro
;   will just quit.
;
;
; to compile:
; phxass särki.ASM m=68020 noexe
; phxlnk särki.o


        incdir  "include:"
        include "exec/types.i"


; First some constants. Screen width and height and maxiter for mandelbrot
; calculation. Note that the code is optimized for the width=256 case, so you
; can't change it easily. Also the system default palette and the current
; MAXITER magically give nice result. If you change MAXITER or make the loop
; reversed expect gfx to screw up.


WIDTH   EQU     256
HEIGHT  EQU     200
MAXITER EQU     64


; Since the mandelbrot is zoomed we must save some variables at some point. I
; use stack and clever register selection to optimize this thing (more about
; this later). Anyhow the structure below represents the stack when
; everything is pushed into it.


        STRUCTURE da_stack,0
        ULONG   yc
        ULONG   deltai
        ULONG   curr_y
        ULONG   start_r
        LABEL   d3_save
        ULONG   d4_save
        ULONG   d5_save
        ULONG   d6_save
        ULONG   d7_save
        LABEL   da_stack_SIZEOF


; To make things a bit easier I use EQUR to define some registers. Whoever
; invented EQUR should have free beer for rest of his life...


kr      EQUR    d0
ki      EQUR    d1
zr      EQUR    d2
zi      EQUR    d3
ci      EQUR    d7
iter    EQUR    d5
fix     EQUR    d6
array   EQUR    a0
tmp     EQUR    a1
deltar  EQUR    d4
curr_r  EQUR    a3
cr      EQUR    a4


; Here are some constants that are needed in the code and 'call' -macro for
; lazy typers like me.


SA_1F           EQU     $8000001f
SA_Width                EQU     $80000023
SA_Height               EQU     $80000024
SA_Depth                EQU     $80000025
SA_DisplayID            EQU     $80000032

sc_RastPort             EQU     84
gb_ActiView             EQU     34
MONITOR_ID_MASK         EQU     $FFFF1000

_LVOTaggedOpenLibrary   EQU     -$32a
_LVOCloseLibrary        EQU     -$19e

_LVOOpenScreenTagList   EQU     -$264
_LVOCloseScreen         EQU     -$42

_LVOGetVPModeID         EQU     -$318
_LVOWriteChunkyPixels   EQU     -$420


call    MACRO
        jsr     (_LVO\1,a6)
        ENDM


; Ah! Finally the code entrypoint. We use private V39+ exec LVO -$32a to
; open graphics library. gfxbase is pushed to stack and current stack
; pointer is stored to a5. (a5) can be then used to load gfxbase to a5 when
; needed and move.l a5,sp can be used to clean up the stack at exit.


_main   move.l  (4).w,a6
        moveq   #1,d0
        call    TaggedOpenLibrary
        move.l  d0,-(sp)

        move.l  sp,a5


; Next we build OpenScreenTagList taglist to stack. First a null to end the
; taglist.

        clr.l   -(sp)


; The next code fragment does the same as 'move.l #SA_Width,d0' but looks
; more obfuscated. Yeah! :-)

        moveq   #31,d0
        bset    d0,d0                   ; d0=$8000001f
        addq.l  #SA_Width-SA_1F,d0


; This code is obvious. It builds stack so it will be:
;
;  SA_Depth
;  6
;  SA_Height
;  HEIGHT
;  SA_Width
;  WIDTH
;  0

        pea     (WIDTH).w
        move.l  d0,-(sp)

        addq.l  #SA_Height-SA_Width,d0
        pea     (HEIGHT).w
        move.l  d0,-(sp)

        addq.l  #SA_Depth-SA_Height,d0
        pea     (6).w
        move.l  d0,-(sp)


; This intro is gfxcard aware. This is achieved by querying the modeid
; of the active viewport and masking just the monitor id out of it. For
; native modes this will give 'low res' mode (320x256 or 320x200). For
; graphics cards we get the first 8-bit mode (most likely 320x240 or
; 320x200). Yes, this is a hack, if the first modeid isn't available
; we're in trouble.
;
; Will push the following to stack:
;  SA_DisplayID
;


        move.l  (a5),a6
        move.l  (gb_ActiView,a6),a0
        move.l  (a0),a0
        call    GetVPModeID
        andi.w  #MONITOR_ID_MASK&$FFFF,d0
        move.l  d0,-(sp)
        pea     SA_DisplayID


; Next open intuition so we can open the screen. Again store the base to
; stack. Note that before storing the base we move stack pointer to a1,
; since this is the register taglist must be given to OpenScreenTagList.

        move.l  (4).w,a6
        moveq   #3,d0
        call    TaggedOpenLibrary
        move.l  sp,a1
        move.l  d0,-(sp)


; Open sesame! Err, screen. Again store pointer to stack. If the screen
; refuces to open exit cleanly.

        move.l  d0,a6
        sub.l   a0,a0
        call    OpenScreenTagList
        move.l  d0,-(sp)
        beq     .noscr


; Next d3-d6 are set up to start position for the zoom. d7 is the zoom
; speed. BITS denotes the bits used for whole number in fixed point math.


        ; Zoom to double spiral:
        ; -.775952266857 +.1347029785251
        ;
        ; -.775952266857 - 1.375 = -2.150952266
        ; -.775952266857 + 1.375 = +0.599047734
        ;
        ; +.134702978525 - 1.200 = -1.065297022
        ; +.134702978525 + 1.200 = +1.334702979
        ;
        ; the following values are calculated with formula:
        ; x * 1<<(16-BITS)

        move.l  #-17621,d3
        move.l  #4907,d4
        move.l  #-8727,d5
        move.l  #10934,d6
        moveq   #127,d7
BITS    EQU     3                       ; 3:13 fixed point


; Now the main loop. First load the screen buffer pointer (graphics
; WriteChunkyPixel assumes huffer in a2) and do the zoom...


.main   lea     (buffer,pc),a2

        add.l   d7,d3
        sub.l   d7,d4
        add.l   d7,d5
        sub.l   d7,d6


; Push zoom position so it won't get lost... Also set up a pointer
; for writing the screen buffer.
```

```
        movem.l d3-d7,-(sp)
        move.l  a2,a0


; Initialize the mandelbrot calculation. Since start_r = d3 we don't
; need to push it to stack. clever. Calculate deltas needed to move along
; the axis.


;;      move.l  d3,-(sp)                ; push start_r
        move.l  d5,-(sp)                ; push curr_y
        sub.l   d3,d4
        sub.l   d5,d6
        asr.l   #8,d4                   ; * 256
        divs.w  #HEIGHT,d6
        ext.l   d6


; Push y movement delta and initialize y and x loop counters. Also set up
; fix register that is used to 'fix' the fixedpoint value after multiply.

        move.l  d6,-(sp)                ; deltai
        clr.l   -(sp)                   ;(yc,base)
        moveq   #16-BITS,fix


; Now the outer y-loop. Increment the current y-position (curr_y) by
; deltai. Also set up variables for x-loop.

.yloop  lea     (deltai,sp),tmp
        move.l  (tmp)+,d0               ; get deltai
        move.l  (tmp),ci                ; ci = curr_y
        add.l   d0,(tmp)+               ; curr_y = curr_y + deltai
        addq.w  #1,(sp)                 ;(yc,base)
        move.l  (tmp),curr_r            ; curr_r = start_r


; The inner x-loop. Move along the x-axis by adding deltar to curr_r.
; Also init stuff for the actual mandelbrot loop.

.xloop  move.l  curr_r,zr

        add.l   deltar,curr_r
        move.l  ci,zi                   ; zi = ci
        moveq   #-1,iter
        move.l  zr,cr                   ; cr = zr


; Calculate the mandelbrot. Initerate until either the loop is run
; MAXITER times or kr+ki > 4. When the loop is done write the
; iteration count to screen buffer.

.loop   move.l  zi,ki
        move.l  zr,kr
        mulu.l  ki,ki
        mulu.l  kr,kr
        lsr.l   fix,ki                  ; ki = zi * zi
        lsr.l   fix,kr                  ; kr = zr * zr

        move.l  kr,tmp
        addq.l  #1,iter
        add.l   ki,tmp
        cmpi.w  #MAXITER,iter
        bhi.b   .nuller
        add.l   zi,zi
        muls.l  zr,zi
        move.l  kr,zr
        asr.l   fix,zi
        sub.l   ki,zr
        add.l   ci,zi                   ; zi = 2 * zi * zr + ci
        add.l   cr,zr                   ; zr = kr - ki + cr

        cmpa.l  #4<<(16-BITS),tmp
        blt.b   .loop

.nuller move.b  iter,(array)+


; The x-loop uses a byte in upper word in stack to count 256 times. The
; lower word is used for the y-loop counter.

        addq.b  #1,(2,sp)
        bne.b   .xloop

        cmpi.w  #HEIGHT,(sp)            ;(yc,base)
        blo.b   .yloop


; The screen buffer is filled now. Write it to screen with graphics
; WriteChunkyPixels() call.

        move.l  (a5),a6
        moveq   #0,d0
        moveq   #0,d1
        moveq   #0,d2
        st      d2                      ; d2=255
        move.l  #HEIGHT-1,d3
        move.l  (8*4,sp),a0
        move.l  d2,d4
        lea     (sc_RastPort,a0),a0
        addq.l  #1,d4
        jsr     (_LVOWriteChunkyPixels,a6)


; This is a neat trick: d0-d2 are used to pop variables out of stack,
; d3-d7 are restored. Nice.

        movem.l (sp)+,d0-d7


; Test for left mouse button. If not selected, loop.

        btst    #6,$bfe001
        bne     .main


; Cleanup: close the screen and libraries.

.noscr  move.l  (sp)+,a0
        move.l  (sp)+,a6
        call    CloseScreen

        move.l  a5,sp
        move.l  a6,a1
        move.l  (4).w,a6

        call    CloseLibrary
        move.l  (sp)+,a1
        jmp     (_LVOCloseLibrary,a6)
```

```
; Chunky buffer as hunk-end-BSS. Note that you need to use some good
; linker (like phxlnk) that kill zero words at end of section.


        CNOP    0,4
buffer  ds.b    WIDTH*HEIGHT


; Särki on kala. Hillos to #amycoders and #amigafin dudes.
```