HowToCode7

COLLABORATORS			
	TITLE : HowToCode7		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 6, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

HowToCode7

1.1 HowToCode: AGA Chipset

```
                The AGA Chipset (Amiga 1200/4000)
                ---------------------------------

**** WARNING ****

AGA Registers are temporary. They will change. Do not rely
on this documentation. No programs written with this information
can be officially endorsed or supported by Commodore. If this
bothers you then stop reading now.

Future Amigas will *NOT* support *ANY* of the new AGA registers.
If you want your product to work on the next generation
of Amigas then either:

a) Program for ECS only (*MOST* ECS will be supported. Don't rely
on Productivity or SuperHires mode via hardware though!)

b) Program your displays via the OS, either using graphics.library
(views) or intuition.library (screens). If you use the OS then any
UserCopperList code you add must *ONLY* be for ECS level instructions
or lower (so, sorry, no 24-bit rainbows).

I have decided to include this material as there are still reasons
for people to program AGA hardware, especially demo coders. PLEASE
do not let me down by using this information to write commercial
software. If this happens I will have to remove the AGA docs from
HowToCode.

And as soon as Commodore provide a suitable way for all programmers
to access the power of the A1200/4000 chipset in a supportable
way, this file will be removed and replaced with one that
tells you how to code properly. But in the meantime......

  1  How do I tell what chipset I am using?
  2  Programming the AGA hardware
  3  Monitor type problems
  4  Resetting sprites on AGA machines
```

1.2 How do I tell what chipset I am using?

```
How do I tell what chipset I am using?
--------------------------------------

Do *NOT* check library revision numbers, V39 OS can and does
run on standard & ECS chipset machines (My Amiga 3000
is currently running V39).

This code is a much better check for AGA than in howtocode4!!!!!


GFXB_AA_ALICE   equ 2
gb_ChipRevBits0 equ $ec

; Call with a6 containing GfxBase from opened graphics.library

   btst  #GFXB_AA_ALICE,gb_ChipRevBits0(a6)
   bne.s is_aa

This will not work unless the V39 SetPatch command has been
executed. If you *must* use trackloader demos then
execute the graphics.library function

    SetChipRev(chipset)


This is a V39 function (No Kickstart 3.0? Then you haven't got
AGA!).

You can set the chipset you require with the following parameters:

   Normal = $00
   ECS    = $03       (Only on ECS or higher)
   AGA    = $0f       (Only on AGA chipset machines)
   Best   = $ffffffff (This gives best possible on machine)


This is called in the system by SetPatch.

The code in howtocode4 also had major problems when being
run on non ECS machines (without Super Denise or Lisa), as the
register was undefined under the original (A) chipset, and
would return garbage, sometimes triggering a false AGA-present
response.
```

1.3 Programming the AGA hardware

```
Programming AGA hardware
------------------------
```

1.4 Bitplanes:

```
Bitplanes:
---------

Set bits 0 to 7 bitplanes as before in BPLCON0 (for 0 to 7 bitplanes)

For 8 bitplanes you should set bit 4 (BPU3) of BPLCON0
bits 12 to 14 (BPU0 to BPU2) should be zero.

Using 64-colour mode (NOT extra halfbrite) requires setting the
KILLEHB (bit 9) in BPLCON2.

Super Hires can be enabled by bit 6 (SHRES) of BPLCON0
```

1.5 Colour Registers:

```
Colour Registers
----------------

There are now 256 24-bit colour registers, all accessed through the original
32 12-bit colour registers. If you suspect this sounds like it could
be messy, then you're right, it is!

AGA works with 8 differents palettes of 32 colors each, re-using
colour registers from COLOR00 to COLOR31

You can choose the palette you want to access via bits 13 to 15 of
register BPLCON3.

BANK2    BANK1    BANK0
bit 15 | bit 14 | bit 13 | Selected palette
-------+--------+--------+-----------------------------
   0   |    0   |    0   | Palette 0 (color 0 to 31)
   0   |    0   |    1   | Palette 1 (color 32 to 63)
   0   |    1   |    0   | Palette 2 (color 64 to 95)
   0   |    1   |    1   | Palette 3 (color 96 to 127)
   1   |    0   |    0   | Palette 4 (color 128 to 159)
   1   |    0   |    1   | Palette 5 (color 160 to 191)
   1   |    1   |    0   | Palette 6 (color 192 to 223)
   1   |    1   |    1   | Palette 7 (color 224 to 255)
```

```
To move a 24-bit colour value into a colour register requires
two writes to the register:

First clear bit 9 (LOCT) of BPLCON3
Move high nibbles of each colour component to colour registers

Then set bit 9 (LOCT) of BPLCON3
Move low nibbles of each colour components to colour registers

For example, to change colour zero to the colour $123456

   lea      (CUSTOM.L),a0
   move.w   #$0135,COLOR00(a0)
   move.w   #$0200,BPLCON3(a0)
   move.w   #$0245,COLOR00(a0)
   move.w   #$0000,BPLCON3(a0)
```

1.6 AGA Sprites

```
Sprites
-------

To  change  the  resolution  of the sprite, just use bit 7 and 6 of
register BPLCON3

bit 7 | bit 6 | Resolution
------+-------+-----------
  0   |   0   | ECS Defaults      (Lo-res/Hi-res = 140ns, Superhires = 70ns)
  0   |   1   | Always lowres     (140ns)
  0   |   1   | Always hireres    (70ns)
  1   |   1   | Always superhires (35ns)
--------------------------

For 32-bit and 64-bit wide sprites use bit 3 and 2 of register FMODE
($dff1fc) Sprite format (in particular the control words) vary for each
width.

bit 3 | bit 2 | Wide        | Control Words
------+-------+-------------+---------------------------------
  0   |   0   | 16 pixels   | 2 words (normal)
  1   |   0   | 32 pixels   | 2 longwords
  0   |   1   | 32 pixels   | 2 longwords
  1   |   1   | 64 pixels   | 2 double long words (4 longwords)
-------------------------------------------------------------
Wider sprites are not available under all conditions.


It  is possible to choose the color palette of the sprite.
This is done with bits 0 to 3 (even) and 4 to 7 (odd)
of register $010C.

bit 3 | bit 2 | bit 1 | bit 0 | Even sprites
bit 7 | bit 6 | bit 5 | bit 4 | Odd Sprites
------+-------+-------+-------+-----------------------------------------
  0   |   0   |   0   |   0   | $0180/palette 0 (coulor 0)
```

```
  0   |   0   |   0   |   1   | $01A0/palette 0 (color 15)
  0   |   0   |   1   |   0   | $0180/palette 1 (color 31)
  0   |   0   |   1   |   1   | $01A0/palette 1 (color 47)
  0   |   1   |   0   |   0   | $0180/palette 2 (color 63)
  0   |   1   |   0   |   1   | $01A0/palette 2 (color 79)
  0   |   1   |   1   |   0   | $0180/palette 3 (color 95)
  0   |   1   |   1   |   1   | $01A0/palette 3 (color 111)
  1   |   0   |   0   |   0   | $0180/palette 4 (color 127)
  1   |   0   |   0   |   1   | $01A0/palette 4 (color 143)
  1   |   0   |   1   |   0   | $0180/palette 5 (color 159)
  1   |   0   |   1   |   1   | $01A0/palette 5 (color 175)
  1   |   1   |   0   |   0   | $0180/palette 6 (color 191)
  1   |   1   |   0   |   1   | $01A0/palette 6 (color 207)
  1   |   1   |   1   |   0   | $0180/palette 7 (color 223)
  1   |   1   |   1   |   1   | $01A0/palette 7 (color 239)
-------------------------------------------------------------------------
```

See Resetting sprites for an OS legal method of setting
sprite resolution.

1.7 Alignment Restrictions

```
Alignment Restrictions
----------------------
```

Bitplanes, sprites and copperlists must be, under
certain circumstances, 64-bit aligned under AGA. Again to benefit from
maximum bandwitdh bitplanes should also only be multiples of 64-bits wide,
so if you want an extra area on the side of your screen for smooth blitter
scrolling it must be *8 bytes* wide, not two as normal.

This also raises another problem. You can no longer use
 AllocMem() to allocate bitplane/sprite memory directly.

Either use AllocMem(sizeofplanes+8) and calculate how many
bytes you have to skip at the front to give 64-bit alignment
(remember this assumes either you allocate each bitplane
individually or make sure the bitplane size is also an
exact multiple of 64-bits), or you can use the new V39
function AllocBitMap() .

 The Magic FMode Requester

1.8 The Magic FMode Register

```
The Magic FMode Register
------------------------
```

If you set your 1200/4000 to a hiresmode (such as 1280x512 Superhires
256 colours) and disassemble the copperlist, you find fun things
happen to the FMODE register ($dff1fc). The FMODE register determines
the amount of words transferred between chipram and the Lisa chip
in each data fetch (I think)....

```
$dff1fc bits 0 and 1 value

$00 - Normal (word aligned bitmaps) - for standard ECS modes
      and up to 8 bitplanes 320x256

$01 - Double (longword aligned bitmaps) - for 640x256 modes in
      more than 16 colours

$10 - Double (longword aligned bitmaps) - Same effect, for 640x256 modes
      but different things happen... Not sure why!

$11 - Quadruple [x4] (64-bit aligned bitmaps) - for 1280x256 modes...

 Fetch Modes Required for Displays [table]
```

1.9 Fetch Modes Required for Displays [table]

```
Fetch Mode Required for Displays
--------------------------------

*ALL* ECS and lower screenmodes require only 1x datafetch. All modes
run *FASTER* with at least 2x bandwidth, so try and use 2x bandwitdh
if possible.
```

	Planes	Colours	Fetchmode
LORES (320x256)			
	6	64	1
	7	128	1
	8	256	1
	8	HAM-8	1
HIRES (640x256)	5	32	2
	6	64	2
	7	128	2
	8	256	2
	8	HAM-8	2
SUPER-HIRES (1280x256)	1	2	1
	2	4	1
	3	8	2
	4	16	2
	5	32	4
	6	64	4
	7	128	4
	8	256	4
	8	HAM-8	4
PRODUCTIVITY (640x480,etc)	1	2	1
	2	4	1
	3	8	2
	4	16	2
	5	32	4
	6	64	4

```
                    7         128                4
                    8         256                4
                    8         HAM-8              4

This table only shows the minimum required fetchmode for each screen.
You should always try and set the fetchmode as high as possible (if
you are 64-bit aligned and wide, then $11, if 32-bit aligned and wide
$01, etc...)

Bits 2 and 3 do the same for sprite width, as has been mentioned elsewhere...

Remember... To take advantage of the increased fetchmodes (which give
you more processor time to play with!) your bitmaps must be on 64-bit
boundaries and be multiples of 64-bits wide (8 bytes)
```

1.10 Smoother Hardware Scrolling

```
Smoother Hardware Scrolling
---------------------------

Extra bits have been added to BPLCON1 to allow smoother hardware
scrolling and scrolling over a larger area.

Bits 8 (PF1H0) and 9 (PF1H1) are the new hi-resolution scroll bits for
playfield 0 and bits 12 (PF2H0) and 13 (PF2H1) are the new bits for
playfield 1.

Another two bits have been added for each bitplane at bits 10 (PF1H6)
and 11 (PF1H7) for playfield 1 and bits 14 (PF2H6) and 15 (PF2H7) to
increase the maximum scroll range from 16 lo-res pixels to 64 lo-res
pixels (or 256 superhires pixels).

Normal 0-16 positions therefore are normal, but it you want to
position your screen at a (super) hires position you need to set
the new bits, or if you require smooth hardware scrolling with either
2x or 4x  Fetch Mode .
```

1.11 What is Ham-8 format?

```
What is HAM-8 Format?
---------------------

Ham8 mode is enabled when the HAM bit is set in BPLCON0 and 8 bitplanes
are specified.

Ham-8 uses *lower* two bits as the command (either
new register (%00), or alter Red, Green or Blue component, as in
standard HAM), and the *upper* 6 bits (planes 2 to 7) as the
register(0 to 63), or as an 6 bit hold-and-modify value to modify
the top 6 bits of an 8-bit colour component.

The lower two bits of the colour component are not altered, so
```

initial palettes have to be chosen carefuly (or use Art
Department Professional or anything that selects colours better)

1.12 Monitor type problems

Monitor Problems

Unfortunately the A1200/AGA chipset does not have the deinterlacer
circuitry present in the Amiga 3000, but instead has new 'deinterlaced'
modes. This gives the A1200 the capability of running workbench (and
almost all OS legal software) the ability to run flicker free at high
resolution on a multiscan or Super VGA monitor.

Unlike the Amiga 3000 hardware it produces these flicker free modes
by generating a custom copperlist, so any programs that generate
their own copperlists will continue to run at the old flickery 15Khz
frequency unless they add their own deinterlace code.

This is a big problem for many A1200 owners as there are very few multiscan
monitors that support 15Khz displays now. Most multiscan monitors will
not display screen at less than 27Khz. People with A1200/4000 and this
kind of monitor *CANNOT* view any games or demos that write their own
copperlists.

Can you help them out? Unfortunately it's not easy. Deinterlacing is
done in AGA by doing two things.

Firstly different horizontal and vertical frequencies are set (These
are set to unusual values for anyone used to Amiga or PC displays!
For example, DblPal is set by default to 27Khz horizontal and 48Hz
vertical) It's important to realise that the vertical frequency
changes too!

Seondly, for non-interlaced screens, bitplane scandoubling is enabled (bit
BSCAN2 in FMODE) This repeats each scanline twice. A side effect of this is
that the bitplane modulos are unavailable for user control.

So... There are three options.

1. Write nasty copperlist code to work with both standard and
promoted displays (Not a good idea!)

2. Use the OS and set up your displays legally, asking the Display Database
for a screenmode that is available for the current monitor.

3. Give up, and say your demo requires a 15Khz monitor.

I think most people will go for option 3. The Commodore 1084/1085, Phillips
8833/8852 and the Commodore 1950/1960/1940/1942 monitors are all capable
of running 15Khz screens.

1.13 exec.library/AllocMem()

```
AllocMem -- allocate memory given certain requirements

memoryBlock = AllocMem(byteSize, attributes)
    D0           -198      D0         D1

void *AllocMem(ULONG, ULONG);
```

1.14 graphics.library/AllocBitMap()

```
AllocBitMap -- Allocate a bitmap and attach bitplanes to it. (V39)

bitmap=AllocBitMap(sizex,sizey,depth, flags, friend_bitmap)
          -918       d0    d1    d2     d3       a0

struct BitMap *AllocBitMap(ULONG,ULONG,ULONG,ULONG, struct BitMap *);

Allocates and initializes a bitmap structure. Allocates and initializes
bitplane data, and sets the bitmap's planes to point to it.

IN:
   sizex = the width (in pixels) for the bitmap data.

   sizey = the height (in pixels).

   depth = the number of bitplanes deep for the allocation.

   flags = BMF_CLEAR - Clear the bitmap.

           BMF_DISPLAYABLE - bitmap displayable on AGA machines in
           all modes.

           BMF_INTERLEAVED - bitplanes are interleaved

   friend_bitmap = pointer to another bitmap, or NULL. If this pointer
                   If present, bitmap will be allocated so blitting
                   between the two is simplified.

SEE ALSO
       FreeBitMap()
```

1.15 graphics.library/FreeBitMap()

```
FreeBitMap -- free a bitmap created by AllocBitMap

FreeBitMap(bm)
  -924     a0

VOID FreeBitMap(struct BitMap *)

Frees bitmap and all associated bitplanes
```

```
IN:
   bm  =  A pointer to a BitMap.
```

1.16 Resetting AGA sprite resolution

```
This is a totally OS-legal way of resetting sprite resolution to 140ns
(ECS default). call FixSpritesSetup: *BEFORE* your  LoadView(NULL)  in your
startup code, and ReturnSpritesToNormal: *BEFORE* the LoadView(wbview)
that returns workbench in your exit code:

Here is the assembler version of the code: See  startup.asm
for an integrated example of this code:

; Setup code - assumes V39 Kickstart or higher


FixSpritesSetup:
        move.l   _IntuitionBase,a6          ; open intuition.library first!
        lea      wbname,a0
        jsr      _LVOLockPubScreen(a6)

        tst.l    d0                         ; Could I lock Workbench?
        beq.s    .error                     ; if not, error
        move.l   d0,wbscreen
        move.l   d0,a0

        move.l   sc_ViewPort+vp_ColorMap(a0),a0
        lea      taglist,a1
        move.l   _GfxBase,a6                ; open graphics.library first!
        jsr      _LVOVideoControl(a6)       ;

        move.l   resolution,oldres          ; store old resolution

        move.l   #SPRITERESN_140NS,resolution
        move.l   #VTAG_SPRITERESN_SET,taglist

        move.l   wbscreen,a0
        move.l   sc_ViewPort+vp_ColorMap(a0),a0
        lea      taglist,a1
        jsr      _LVOVideoControl(a6)       ; set sprites to lores

        move.l   wbscreen,a0
        move.l   _IntuitionBase,a6
        jsr      _LVOMakeScreen(a6)
        jsr      _LVORethinkDisplay(a6)     ; and rebuild system copperlists

; Sprites are now set back to 140ns in a system friendly manner!

.error
        rts

ReturnSpritesToNormal:
; If you mess with sprite resolution you must return resolution
; back to workbench standard on return! This code will do that...
```

```
        move.l   wbscreen,d0
        beq.s    .error
        move.l   d0,a0

        move.l   oldres,resolution          ; change taglist
        lea      taglist,a1
        move.l   sc_ViewPort+vp_ColorMap(a0),a0
        move.l   _GfxBase,a6
        jsr      _LVOVideoControl(a6)       ; return sprites to normal.

        move.l   _IntuitionBase,a6
        move.l   wbscreen,a0
        jsr      _LVOMakeScreen(a6)         ; and rebuild screen

        move.l   wbscreen,a1
        sub.l    a0,a0
        jsr      _LVOUnlockPubScreen(a6)

.error
        rts


oldres          dc.l  0
wbscreen        dc.l  0

taglist         dc.l  VTAG_SPRITERESN_GET
resolution      dc.l  SPRITERESN_ECS
                dc.l  TAG_DONE,0

wbname          dc.b  "Workbench",0
```

1.17 intuition.library/LocPubScreen()

```
LockPubScreen -- Put a lock on a Public Screen.

screen = LockPubScreen( Name )
 D0          -510         A0

struct Screen *LockPubScreen( UBYTE * );

Prevents a public screen (or the Workbench) from closing.
```

1.18 intuition.library/UnlockPubScreen()

```
UnlockPubScreen -- Remove lock from a Public Screen.

UnlockPubScreen( Name, [Screen] )
     -516          A0    A1

VOID UnlockPubScreen( UBYTE *, struct Screen * );
```

```
Releases a lock from  LockPubScreen()

IN:
   Usually Name = NULL and Screen = pointer returned by LockPubScreen()
```

1.19 graphics.library/VideoControl()

```
VideoControl -- Parse tags on viewport colormap.

err = VideoControl( cmap , tags )
d0         -708      a0    a1

ULONG VideoControl( struct ColorMap *, struct TagItem * );

Process the tag commands on the colormap.

IN:
   cm   = pointer to struct ColorMap
   tags = pointer to a table of videocontrol tagitems.

OUT:
   error = NULL if no error occured.
```

1.20 graphics.library/SetChipRev()

```
SetChipRev -- Enables Chip Set features

chipbits = SetChipRev(Rev)
   -888               d0

IN:
   Rev - Revision to be enabled ($ffffffff for best possible)

OUT:
   chipbits - State of chipset on exit.

Only call this routine once. It is called by the OS in SetPatch, but you
should use it if you are writing Non-DOS demos or games.
```