HowToCode7

COLLABORATORS			
	TITLE : HowToCode7		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

HowToCode7

1.1 HowToCode: Using the Blitter

```
Using the Blitter
-----------------

  1  OwnBlitter()/DisownBlitter()
  2  Blitter Timing Problems
  3  Blitter Speed Optimisation
  4  Calculating LF Bytes
  5  Clearing with the blitter
```

1.2 OwnBlitter()/DisownBlitter()

```
If you are using the blitter in your code and you are leaving the
system intact (as you should) always use the graphics.library
functions  OwnBlitter()  and  DisownBlitter()  to take control
of the blitter. Remember to free it for system use, many system
functions (including floppy disk data decoding) use the blitter.

 OwnBlitter()  does not trash any registers. I guess  DisownBlitter()
doesn't either, although Chris may well correct me on this, and
they are fast enough to use around your blitter code, so don't
just  OwnBlitter()  at the beginning of your code and  DisownBlitter()
at the end, only  OwnBlitter()  when you need to.
```

1.3 graphics.library/OwnBlitter()

```
OwnBlitter -- get the blitter for private usage

OwnBlitter()
   -456

void OwnBlitter( void );

If blitter is available return immediately with the blitter
```

```
locked for your exclusive use. If the blitter is not available
put task to sleep. It will be awakened as soon as the blitter
is available. When the task first owns the blitter the blitter
may still be finishing up a blit for the previous owner. You
must do a WaitBlit before actually using the blitter registers.

Calls to OwnBlitter() do not nest. If a task that owns the
blitter calls OwnBlitter() again, a lockup will result.
(Same situation if the task calls a system function
that tries to own the blitter).
```

1.4 graphics.library/DisownBlitter()

```
DisownBlitter - return blitter to free state.

DisownBlitter()
    -462

void DisownBlitter( void );

Free blitter up for use by other blitter users.
```

1.5 graphics.library/QBlit()

```
QBlit -- Queue up a request for blitter usage

QBlit( bp )
-276   a1

void QBlit( struct bltnode * );

Link a request for the use of the blitter to the end of the
current blitter queue.  The pointer bp points to a blit structure
containing, among other things, the link information, and the
address of your routine which is to be called when the blitter
queue finally gets around to this specific request.  When your
routine is called, you are in control of the blitter ... it is
not busy with anyone else's requests.  This means that you can
directly specify the register contents and start the blitter.
Your code must be written to run either in supervisor or user
mode on the 68000.

IN:
      bp - pointer to a blit structure

Your routine is called when the blitter is ready for you.
In general requests for blitter usage through this channel are
put in front of those who use the blitter via OwnBlitter and
DisownBlitter. However for small blits there is more overhead
using the queuer than Own/Disown Blitter.
```

1.6 graphics.library/QBSBlit

```
QBSBlit -- Synchronize the blitter request with the video beam.

QBSBlit( bsp )
 -294    a1

void QBSBlit( struct bltnode * );

Call a user routine for use of the blitter, enqueued separately from
the  QBlit  queue.  Calls the user routine contained in the blit
structure when the video beam is located at a specified position
onscreen.   Useful when you are trying to blit into a visible part
of the screen and wish to perform the data move while the beam is
not trying to display that same area.  (prevents showing part of
an old display and part of a new display simultaneously).  Blitter
requests on the QBSBlit queue take precedence over those on the
regular blitter queue. The beam position is specified the blitnode.

IN:
   bsp - pointer to a blit structure.
```

1.7 graphics.library/WaitBlit()

```
WaitBlit -- Wait for the blitter to finish.

WaitBlit()
 -228
```

1.8 Blitter Timing

```
Blitter timing
--------------
Another common cause for demo crashes is blitter timing.

Assuming that a particular routine will be slow enough that a blitter
wait is not needed is silly. Always check for blitter finished, and
wait if you need to.

Don't assume the blitter will always run at the same speed too. Think
about how your code would run if the processor or blitter were running
at 100 times the current speed. As long as you keep this in mind,
you'll be in a better frame of mind for writing code that works
on different Amigas.

Another big source of blitter problems is using the blitter in interrupts.

Many demos do all processing in the interrupt, with only a

.wt   btst     #6,$bfe001  ; is left mouse button clicked?
      bne.s    .wt
```

loop outside of the interrupt. However, some demos do stuff outside the interrupt too. Warning. If you use blitter in both your interrupt and your main code, (or for that matter if you use the blitter via the copper and also in your main code), you may have big problems....

Take this for example:

```
   lea      $dff000,a5
   move.l   GfxBase,a6
   jsr      _LVOWaitBlit(a6)
   move.l   #-1,BLTAFWM(a5)                ; set FWM and LWM in one go
   move.l   #source,BLTAPT(a5)
   move.l   #dest,BLTDPT(a5)
   move.w   #%100111110000,BLTCON0(a5)
   move.w   #0,BLTCON1(a5)
   move.w   #64*height+width/2,BLTSIZE(a5)  ; trigger blitter
```

There is *nothing* stopping an interrupt, or copper, triggering a blitter operation between the WaitBlit() call and your final BLTSIZE blitter trigger. This can lead to total system blowup.

Code that may, by luck, work on standard speed machines may die horribly on faster processors due to timing differences causing this type of problem to occurr.

You can prevent this by using OwnBlitter()

The safest way to avoid this is to keep all your blitter calls together, use the copper exclusively, or write a blitter-interrupt routine to do your blits for you, which is very good because you avoid getting stuck in a waitblit-loop.

Always use the graphics.library WaitBlit() routine for your end of blitter code. It does not change any registers, it takes into account any revision of blitter chip and any unusual circumstances, and on an Amiga 1200 will execute faster (because in 32-bit ROM) than any code that you could write in chipram.

1.9 Calculating LF Bytes

Instead of calculating your LF-bytes all the time you can do this

```
A  EQU   %11110000
B  EQU   %11001100
C  EQU   %10101010
```

So when you need an lf-byte you can just type:

```
   move.w   #(A!B)&C,d0
```

1.10 Blitter Clears

```
Blitter clears
--------------
If you use the blitter to clear large areas, you can generally
improve speed on higher processors (68020+) by replacing it by
a cache-loop that clears with movem.l instead:

        moveq  #0,d0
        moveq  #0,d1
        moveq  #0,d2
        moveq  #0,d3
        moveq  #0,d4
        moveq  #0,d5
        moveq  #0,d6
        sub.l  a0,a0
        sub.l  a1,a1
        sub.l  a2,a2
        sub.l  a3,a3
        sub.l  a4,a4
        sub.l  a5,a5

        lea    EndofBitplane,a6
        move.w #(bytes in plane/156)-1,d7
.Clear
        movem.l d0-d6/a0-a5,-(a6)
        movem.l d0-d6/a0-a5,-(a6)
        movem.l d0-d6/a0-a5,-(a6)
        dbf d7,.Clear

; final couple of movems may be needed to clear last few bytes of screen...

This loop was (on my 1200) almost three times faster than
the blitter.

With 68000-68010 you can gain some time by NOT using blitter-
nasty and the movem-loop.
```

1.11 Blitter Speeds

```
BLITTER SPEEDS. (from the Hardware Reference Manual)
----------------------------------------------------

Some general notes about blitter speeds. These numbers are for the
blitter only, in 16-bit chip ram.

                  n * H * W
   time taken = -----------
                    7.09   (7.15 for NTSC)

time is in microseconds. H=blitheight,W=blitwidth(#words),n=cycles

n=4+....depends on # DMA-channels used

A: +0          (this one is free!)
B: +2
```

```
C or D: +0     In line-mode, every pixel takes 8 cycles.
C and D: +2

So, use A,D,A&D for the fastest operation.
Use A&C for 2-source operations (e.g. collision check or so).
```