Hardware Bugs & Warnings

The following sections describe known bugs in the operation of the Jaguar hardware. Side-effects of these bugs should not be relied on, as they may be fixed in future versions of the hardware.

GPU/DSP Bugs & Warnings

1) The scoreboard mechanism does not work on the data of any indexed store instruction. This means that any indexed store instruction that stores data from a long latency operation (such as a divide or external load) should place an 'or' instruction prior to the store. For example:

```
div       r0,r3
store     r3,(r14+6)
```

should be written as:

```
div       r0, r3
or        r3, r3
store     r3,(r14+6)
```

2) In any instruction where the destination register is written to without being read, the destination register will not be protected by the scoreboarding mechanism of the GPU/DSP. This includes MTOI, NORMI, RESMAC, all MOVE variations, and all LOAD variations.

If one of these 'destination write-only' instructions writes to the same destination register as a prior instruction and there have been no intervening reads from that register, it is possible for the second instruction to complete before (or simultaneously with) the first, causing the register to become corrupt. This bug only becomes a problem when doing 'dummy' instructions as shown in the following example:

```
div     r2,r4  ; Divide starts
               ;(takes 18 ticks)
moveq #4,r4    ; Move completes
              ; before divide
```

Although this code doesn't make much sense, it might appear at the end of a loop as shown below:

```
loop:
      ...
      jr    EQ,loop
      div   r2,r4

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Any number of instructions could
; appear here. Unless one of them reads
; R4, the result of the MOVEQ will be
; unreliable.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

      moveq #4,r4
```

In this case, when the loop condition fails, the DIV/MOVEQ instruction sequence will occur and register R4 will be corrupted. This can be prevented by causing the destination register to be read prior to the move as is shown in the following example:

```
loop:
      ...
      jr    EQ,loop
      div   r2,r4

      or    r4,r4
      moveq #4,r4
```

Please note that these examples illustrate one particular sequence (DIV/MOVEQ). Any instruction which writes to a register followed later in the instruction stream by a 'destination write-only' instruction with no intervening reads of that register is unreliable.
In practice, this creates two cases. If a DIV or LOAD instruction is used to write to a register, a read of that register must be inserted prior to any

'destination write-only' instruction that writes to the same register.

In addition, any instruction which writes its result into a register and is immediately followed by a 'destination write-only' instruction which writes to the same register will also corrupt the register. This effect is shown in the example below:

```
loop:
      ...
      jr     EQ,loop
      add  r10,r12
      moveq #1,r12 ; ADD will trash this
```

You should also note that a 'dummy' instruction sequence, as shown above, is rare. In normal program code where the result of a register write is used, the bug does not occur. This is illustrated in the following example:

```
      load  (r2),r4
      add   r4,r6
      moveq #4,r4 ; Safe because R4 was
                  ; read above
```

3) Neither the DSP or the GPU will reliably execute '**jr**' or '**jump**' instructions unless they are in internal RAM.

4) The DSP may not be used in high priority. The DMAEN bit of D_FLAGS should always be 0. Otherwise, doing an external load or store will cause the DSP to hang, needing a reset to recover.

5) The GPU and blitter may not be used in high bus priority while the object processor is running. The DMAEN bit of G_FLAGS should be 0, and the BUSHI bit of B_CMD should be 0.

No bus master may operate at a higher priority than the object processor. If something else gets the bus between the second and third phrases of an object header, then the line buffer address can be corrupted, causing horizontal black stripes and possibly other artifacts in the display.

6) The DSP and the GPU must not be stopped by an external processor writing directly to the D_CTRL or G_CTRL registers. Only the GPU should turn off the GPU, and only the DSP should turn off the DSP.

If one processor wants to shut down another one, the best way is to ask them to do it to themselves. For example, place a special code into a semaphore and then cause an interrupt for the processor you want to shut down. The interrupt handler would see the semaphore and shut down the processor itself.

7) The DSP must not do an external write unless it is preceded by an external read that will complete for the write starts. This problem is intermittent and could be missed by testing. Be careful in any DSP code that writes to external memory.

Example #1:

```
      Load  (rl),r2
      or    r10,r11
      store r11,(r3)
```

Example #2:

```
      Load  (rl),r2
      or    r2,r11
      store r11,(r3)
```

Example #3:

```
      Load  (rl),r2
      or    r2,r2
      or    r10,r11
      store r11,(r3)
```

Example #1 will not work correctly but example #2 will. This is because the result of the load is required for the **or** operation to be performed. To make example #1 work change it to example #3.

8) The value in the High Data Register in the GPU is changed after ANY external **load,** not just **loadp.** This means that if an interrupt in running in the GPU that **loads** from external memory the underlying program may not use **loadp**.

 © 1994 Atari Corp.

9) There is a bug in the divider of the GPU and DSP. If you try to do two consecutive divides without there being at least 1 clock cycle of idle time between them, then the result of the second divide will be wrong.

This will only occur when the two divides are separated by less than 16 clock cycles, and the second divide as the quotient of the first divide as one of its register operands, and there is no score-board dependency on the quotient of the first one prior to the second.

The work-around should be to either make sure that more than 16 clock cycles occur between divide instructions, or make sure that an instruction which is dependent on the quotient of the first divide occurs before the second divide.

Example #1:

```
div   r0,r1
moveq #3,r5
div   r5, r1
```

should be like this:

```
div   r0,r1
moveq #3,r5
or    rl,rl
div   r5,r1
```

Example #2:

```
div   r0,r1
moveq #3,r5
div   r5,r1
```

should be like this:

```
div   r0,r1
moveq #3,r5
or    rl,rl
div   r5,r1
```

10) DSP matrix multiplies only work in the lower 4K of DSP RAM. The DSP matrix register can only point to memory locations in the first 4K of DSP RAM. Only address lines 2-11 are programmable; the rest of the matrix address is hard-wired to $F1Bxxx.

11) When you write a value to the G_FLAGS or D_ FLAGS registers, it may not appear to have changed in the following two instructions because of pipe-lining effects. If you are going to use the flags set by a STORE instruction, or are changing one of the other bits such as the register bank, then ensure that there are two NOP instructions after the STORE to either of these registers.

Blitter Bugs & Warnings

1) The Y add control bits in the Al and A2 address generators in the blitter are not differentiated between properly. The A2 Y add control bit is ignored. The Al Y add control bit affects both address generators. However, if the Y sign bits are set in either address, the corresponding add control bit has to be set for the number to be negative.

Either do not use this function, or use it on both address generators.

2) SRCSHADE only works if the GOURZ bit is set. No actual Z-buffer data needs to be calculated or written, but GOURZ must be set.

3) If A1_CLIP *x* is not on a phrase boundary, then clipping occurs on the right side even if the Al_CLIP bit is not set. This applies to the destination even if the DSTA2 bit of the B_CMD register is set.

To avoid this problem, set Al_CLIP to 0 if not clipping, and when using DSTA2 make sure the source is an even phrase width.

4) Unaligned blits in 2 bit per pixel mode are not reliable. Use 1 bit per pixel blits instead.

5) If Z-buffer operation is enabled and the ADDDSEL or SRCSHADE bits are set, then the data is sometimes corrupted.

To work around this, break the operation into two blits, one to do the SRCSHADE or ADDDSEL into an offscreen buffer, and then a second one to perform the Z-buffer operation onto the screen.

 © 1994 Atari Corp.

Object Processor Bugs & Warnings

1) It is possible for the last column of pixels of a RMW (Read-Modify-Write) object to be corrupted if it is followed by another bitmap object. This will happen on the right side unless the REFLECT bit is set, in which case it will happen on the left side.

To work around this problem, you can ensure that the last pixels of the source data are all transparent (i.e. pad the object data). Or you can make sure that the next object in the object list will not appear on the same scanlines as the RMW object. Or you can place an always-false branch object after the RMW object.

2) Setting the VSCALE field of a scaled bitmap object to a value greater than 7.0 (%111.00000) will fail. As documented, values as high as 7.1F (%111.11111) may be used with the HSCALE field.

3) Setting the HSCALE field of a 24-bit scaled bitmap object to any value other than 1.0 will cause the object to be distorted.

Miscellaneous Hardware Bugs & Warnings

1) There is a bug in the Jaguar UART. If a start bit is detected at a certain phase in the UART's divide by 16 timer, it will be shifted in twice, resulting in a left shift of the data byte.

The problem may be avoided by preceeding a data packet with a dummy byte where the MSB is set (e.g. $80). The receiver code should discard this dummy byte. Subsequent bytes should be exactly aligned (i.e. 2, 3, or 4 stop bits exactly, before the next start bit). This will result in causing the falling edge of the next start bit to miss the phase of the UART counter which causes the problem.

If a gap is left after a byte which is more than 2 bit times long, or is not exactly aligned with the previous byte, then the dummy byte must be retransmitted (to align the UART counter again).

2) The **clr.l <ea>** and **move.l <ea>,-(an)** instructions of the 68000 do not work correctly when writing to Jaguar GPU & DSP hardware registers and internal RAM.

The address ranges with this restriction are $F02000 to $F07FFF and $F1A000 to $F1F000. These instructions may be safely used on memory addresses outside these ranges.

Because the 68000 has a 16-bit data bus, 32-bit writes to memory actually occur as two separate 16-bit writes which happen in succession. With certain instructions such as those shown above, the order in which the high word and low word are written is reversed, which causes problems when writing to these address ranges.

While these are the only ones we know about at present, it is possible there are other instruction/address mode combinations that have this problem. The best way around it is to use the GPU and/or DSP instead of the 68000 when you want to write to Jaguar GPU/DSP registers, and to use the blitter when you want to copy information into GPU or DSP RAM.

If you are using a high-level language compiler, make sure that it does not generate **clr.l** instructions for code that accesses this address space.

 © 1994 Atari Corp.