Frequently Asked Questions About Jaguar

About The Developer Package

Q: I do not have an Atari TOS based machine, but only have ST Software to work with my developer package right now

A: The current versions of both the PC/MSDOS and Atari versions are available online on Compuserve and the Atari Software Development BBS. Or you can contact Jaguar Developer Support to obtain the PC/MSDOS versions of the tools.

* * *

Q: I am not satisfied with the examples that came with my developer's package. Is there more demo code available?

A: Atari is creating new demo code and examples all the time, and it's possible that there have been updates since you got your developer's kit.. Look on the developer support BBS and private Jaguar Developer areas of Compuserve. (See **Online Support** section of the **Getting Started** chapter of the documentation.)

* * *

Q: What am I supposed to use as a 3-D object editor?

A: You can use whatever you prefer. The conversion utility out of our 3-D package uses .3DS files from AutoCAD 3D-Studio v2.0 or v3.0 running on the PC.

About Problems Running The Development Software Or System

Q: I have trouble getting the debugger to transfer information from my PC to my Alpine Board. Either the debugger says "No Bi-directional Parallel Port Found" or it says "Error While Reading FAST" during a transfer.

A: Either you do not have a bidirectional parallel port installed in your PC, or else you need to adjust the timing of the high-speed transfer.

The Jaguar debugger communicates with the Alpine board via a bidirectional parallel port installed in your PC. Calling a port "bidirectional" means that it is capable of either receiving or transmitting 8-bits of information at a time.

Most inexpensive I/O cards for PCs are intended to be used for output only and do not feature bidirectional parallel ports. In order to work around this, some PC programs and hardware add-ons use the port's control signals to receive data, rather than using the port's data lines. This allows them to do bidirectional communication on a unidirectional port, but it is much slower because it cannot transfer as much information at once.

In order to achieve acceptible performance, the Jaguar debugger requires a true bidirectional parallel port. The Jaguar Developer's Kit includes a PC I/O card that features such a port.

If you are seeing a message that says "no bidirectional parallel port found" then either the debugger could not communicate with the Alpine because the parallel port was incapable of receiving information back from the Jaguar, or else it's possible that the Alpine board's parallel port is tied up for some reason. Reset your Alpine board. If you still see the message when you run the debugger, then you probably don't have a bidirectional port installed.

If you have installed the card included with the developer's kit, make sure it is configured correctly for your system. If you need assistance with this, please contact Developer Support.

If you have a bidirectional port installed and are seeing a message like "Error While Reading FAST" during a transfer, then the timing of the debugger's high-speed parallel transfer may require adjustment for your machine. The debugger has a variable named "PPROT" which allows you to adjust this. After loading the debugger, type the following:

pprot = *n*

Where '*n*' is a number from 1 to 9. Experiment with different numbers until you find one that works reliably. After you find one that works reliably, you can add this line to your RDB.RC file so that this adjustment is made automatically each time you run the debugger.

Another adjustment that has been known to help is changing the ISA bus speed of your machine. This is typically done in your BIOS setup screen that is accessible when you first boot the system.

* * *

Q: I have problems with running GULAM on my Falcon030.

A: You should make sure that you do NOT try to run GULAM under MultiTOS, the multitasking extensions to the TOS operating system. If you want to work under MultiTOS then you should use another shell, for example the UNIX C-Shell style shell TCSH. TCSH is available online.

* * *

Q: The newest version of RDBJAG crashes under MultiTOS.

A: When running on 68030-up systems, MultiTOS features hardware memory protection. RDBJAG installs itself into system interrupt vectors. When other programs call interrupts, this causes the system to think these other programs are accessing RDBJAG's memory. RDBJAG needs to allow other programs access to its memory. This is controlled by the 'global' memory protection flag in the program header. The most likely problem is that this flag is probably turned off and needs to be turned on. If you don't know how to do this, please contact Jaguar Developer Support. Alternately, you can run MultiTOS with memory protection turned off. This can be done with the MultiTOS CPX in the CONTROL PANEL accessory. You will then have to reboot for the changes to take effect.

* * *

Q: I tried taking out the Alpine board and put in a cartridge, but it would not run. Is the cartridge broken?

A: Most likely not. In order to run a cartridge on a development system, it is necessary to hold down the 'B' button on controller #1 when you turn on the machine. This is because you need to signal to the development console that the debugging routines are not supposed to be installed at startup, and that it should act like a standard retail version of the Jaguar.

If the cartridge runs, but you hear a lot of static noise, then you must connect a 1k resistor between lines 4 and 5 of the header at the end of the 10-connector ribbon cable that goes from the development console and plugs into the back of the Alpine board (this is the STOP button cable). This is only necessary for some systems with serial numbers starting with less than K14... (See **The Jaguar Development System ROMulator** chapter of the **Technical Overview** section of the documentation.)

* * *

Q: On my ATARI Falcon030 I cannot establish communication between RDBJAG and the Stub in the development console - Help!

A: This is a problem only with older versions of RDBJAG. The current version of RDBJAG is available online.

* * *

Q: The command OD does not work with my version RDBJAG. What is wrong ?

A: The OD command is actually a DB procedure which is defined in the OD.DB script file. This script is normally loaded automatically by the debugger through the RDB.RC startup script (along with the scripts GPU.DB and FILL.DB). These scripts implement a number of Jaguar DSP/GPU-specific debugger commands. The problem is most likely one of the following:

1) RDB.RC was not loaded at startup because it could not be located. The complete pathname for this script file must be contained in the RDBRC environment variable. See the **Configuration** section of the **Getting Started** chapter for more information.

2) The RDB.RC file has been edited and no longer loads the OD.DB script.

3) The OD.DB script file could not be found. This script must reside somewhere in the search path specified by the DBPATH environment variable. See the **Configuration** section of the **Getting Started** chapter for more information.

* * *

Q: My source code developed under the TOS based system does not assemble with the PC-based tools.

A: While we intend to maintain backwards compatibility to the highest possible degree, it is sometimes not practical or possible to do this while at the same time adding new features. See the text files in the JAGUAR\DOCS directory for information on changes to the tools.

* * *

Q: How frequently are the development tools updated ?

A: There is no particular set schedule for updates. New versions are made available as soon as they are ready. We are constantly improving our tools, This includes expansion to other platforms and strong improvements in user interface. The MADMAC Assembler, ALN linker, and RDBJAG/WDB debugger are updated often, and the most recent versions are always available online.

It would be a good idea to get into the habit of checking the online areas at least once every week or two to see what's new and improved.

About Documentation Clarifications

Q: I want to program parts of my program (i.e. the user interface in the selection menus) in the 68000 using the C compiler. How do I avoid unexpected crashes?

A: Most problems with code written in C happen because C compilers do not know about Jaguar-specific requirements such as phrase alignment or double phrase alignment. To make sure that a file that contains 68k and GPU or DSP code gets linked correctly, you should follow some major guidelines:

* Always link C-compiler code to be the last module(s)

* Phrase align the end of every segment of assembly language module you write. This means separate alignment for text, data and bss segment of each single module.

The ALN linker has an option that can automatically align the size of the segments inside each included module to a specified boundary.

* Make sure that you phrase align ALL data you are using/generating from the C-module(s). A way to achieve this is to build a customized *malloc()* routine that only gives back phrase aligned blocks of memory. Always generate the structures to work within these given blocks. You may also use hard-coded adresses to structures that have to be accessed in phrase mode.

* It may work better if you define some of your arrays and initialized data inside assembly modules, and reference them as 'extern' in your C code.

* * *

Q: I want to use the 'character painting' feature of the blitter to use a 16x16 bitmap for my font.

A: The blitter can do this in 8 bit wide segments, so you have to setup the blitter to do two blits of 8 bit source width.

* * *

Q: As there are objects that must be two-phrase aligned, is there an '.dphrase' feature in the assembler ?

A: Yes, MADMAC can do this. You can also tell the linker to automatically align each segment of each module in a variety of different ways, including single or double-phrase. But it is generally a good idea to make sure that your objects are located correctly without requiring these features, either by preallocating memory for the objects and corretly adjusting them or by hardcoding their adresses.

See the MADMAC documentation for more information.

Programming Questions

Q: How to save highscores in the EEPROM of my cartridge?

A: One of the sample programs provided in the developer's kit demonstrates how to do this. See the **Sample Programs** section for more information.

* * *

Q: I do not now how to setup sound. Where I find the documentation ?

A: Refer to the sample program source code for SIMPLE. Also investigate the Jaguar Synthesizer and Music Driver. Also look into the **Jaguar Console Hardware Release Notes** section of the **Technical Reference** chapter.

* * *

Q: My code seems to be too slow for what I want to do.

A: There are many different ways of speeding up code. In general, do not spend more time than absolutely neccesary doing 68000 code. The more you can utilize the GPU, DSP, and Blitter, the better your program will run. Here are some basic guidelines:

* Optimize all 68000 interrupt code to need the absolute minimum of time.

* Try to keep the 68000 off the bus. For example, don't run 68000 code directly from ROM space. Accessing ROM takes as much as 10 times longer than accessing DRAM.

* Don't use the 68000 or even the GPU or DSP for memory copy operations, use the Blitter.

* Use the Blitter in phrase mode where possible.

* Use the GPU and the DSP for calculations where possible. You may have them both runnning at the same time.

* You may start the Blitter and do calculations in DSP or GPU until the blit is completed.

* Be careful to interleave the instructions for any GPU or DSP code you write so that you avoid register wait states.

Please read the **Jaguar Programming Tips & General Procedures** section of **Appendix B**.

* * *

Q: Does the Jaguar feature support for analog joysticks and other special controllers?

A: Yes, you'll find a sample program included with your development system. See also the **Jaguar Controllers and Controller Ports** section of the **Technical Reference** chapter.

* * *

Q: I have set up a list of 50 objects to be displayed, but it does not run.

A: If you maintain the object list with a 68000 interrupt handler, you might be running out of time during the interrupt routine because the next interrupt occurs while you are still handling the previous one.

You might be able to solve this by optimizing your 68000 code, although if this doesn't work, you may need to move your object list update routine to the GPU. (Which is going to be the better solution in the long run.)

The main limitation is not the number of objects you have overall, it is the number of objects which must be displayed in the same horizontal line.

The main restriction on the size of the object list is the time it takes for the Object processor to scan all the objects for a given scanline, read each one's header, and copy data from the bitmap to the line buffer (keep your bitmaps in DRAM, not ROM!) This all has to take place in approx. 63.5 µsec or less on NTSC systems (PAL gives you a little more time, but we suppose and urgently suggest that you would want to have your software running everywhere). The number depends to a big part on DWIDTH and on the hardware configuration for RAM access as set in MEMCON1 (don't change this register!).

Please read the **Jaguar Programming Tips & General Procedures** section of **Appendix B**.

* * *

Q: I run out of time by using the object processor for moving objects by just changing the XPOS and YPOS fields of the objects. How to avoid that?

A: Aside from any other optimizations of your object list that may be possible, you may simply be eating up too much bandwidth with an object list that contains too many moving objects. As general rule we would like to ask you to:

* Use the Blitter to draw/move the objects if the objects are static for more than ten frames

* Move the objects with the object processor if your objects move faster than every ten frames.

* * *

Q: I want to use the MMULT instruction in GPU and/or DSP. How is the data organized if the second matrix is stored in RAM?

A: The organisation in RAM is word packed, as in the registers. However, this instruction has been designed for implementation of algorithms that operate on word packed data structures as 8x8 matrices in discrete cosine transformations so you should not use it for general purpose matrix calculations. You in general are better off if you spare the time for packing and unpacking the data

and use an explicit sequence of IMULTN, IMACN and RESMAC instead.

About Documentation Bugs And Additions

Q: My branch objects do not work as advertised. Why?

A: There is a typo in the **Jaguar Software Reference Guide** before Rev 2.2, May 3rd 1994. The word 'not' is missing in the description of the BRANCH object type. *The LINK field contains the phrase aligned address that is used if the branch is **not** taken.*

* * *

Q: Are the DSP timer divider registers JPIT2 and JPIT4 write accessed at the same memory location?

A: No, that is a typo in the Jaguar Software Reference Guide before Rev 2.2, May 3rd 1994. The location for JPIT2 write access is $F10002, for JPIT4 it is $F10006.

* * *

Q: I've created an object list that includes a GPU interrupt object, but instead of the interrupt occuring just on the scanline I've specified, it appears to occur on every line.

A: There is a typo in the **Jaguar Software Reference Manual** before Rev. 2.3 on page 17. The Graphics Processor Object does not have a YPOS field. Bits 0-2 are the object type, and bits 3-63 are DATA to be used by the GPU interrupt service routine.

To work around this, simply use branch objects immediately before your GPU object so that the GPU object is called only for the scanline(s) you desire.

About hardware features

Q: In the demos, all object lists have two branch objects in the beginning. Why?

A: The two branch objects are mandatory to keep the hardware happy. Unless your object list contains only a single stop object, always include these two branch objects at the beginning.

* * *

Q: Shading texture mapped surfaces using SCRSHADE does not work correctly.

A: The documentation states "SRCSHADE may be used with GOURZ, not with GOURD". There is a bug in the blitter that *requires* GOURZ to be set. See the **Blitter BUG List** section of the **Technical Reference** Chapter.

* * *

Q: My program code runs unreliably when I switch the Object processor on and off during the run of my code.

A: Never disable the Object processer once it is running. Your goal is probably to turn off video. You can do this by aiming the Object List pointer at a single STOP object.

* * *

Q: Do the PWM DACs not work?

A: Correct, do not use the PWM DACs. They are not even connected in the Jaguar console. Use the I²S-Bus for sound activities. Refer to the source files SIMPLE.S and SIMPLE.DAS, which are part of the SIMPLE sound example program..

* * *

Q: Accessing Jaguar registers and On-Chip RAM sometimes has unpredictable results. What is going on?

A: Never access the On-Chip RAM in the Jaguar Chipset except by reading or writing longwords. Same holds for ALL 32 bit wide registers.

* * *

Q: Every time I use the 68000 **clr.l** instruction to set registers in the Jaguar Chipset the result seems to be unpredictable. Why is that?

A: Never use the 68000 **clr.l** instruction for accessing long words in the Jaguar GPU & DSP address space. This includes both hardware registers and internal RAM. As you can perform the same operation more efficiently and more quickly using other instructions, there should be no reason to use **clr.l** anyway.

The problem has to do with the way this particular instruction writes to memory, which is different from most other 68000 instructions. This problem can also happen with certain other instruction and address mode combinations. Please see the **Hardware Bugs & Warnings** chapter for more complete information about this problem and how to work around it.

* * *

Q: Some sequences of GPU statements are not working. Is this a hardware bug ?

A: The current revision of the GPU chip has a few minor problems which mostly would appear only in cases where the running code would not really make sense. Refer to the **GPU/DSP Bug List** section of the **Technical Reference** Chapter.

* * *

Q: I want my object list update routine to do as little work as possible. Exactly which fields of the objects need to be reinitialized before each frame?

A: The following fields are changed by the object processor and must be reinitialized after the end of a frame:

Bitmap or **Scaled Bitmap**:	HEIGHT, DATA
Scaled Bitmap:	REMAINDER

Note that there are some interesting effects that can be achieved by **not** updating these fields after each frame.

One such effect is that by arranging your data as a vertical strip of frames and by setting the HEIGHT field to the height of this strip (number of frames * scanlines per frame), you can play back an animation automatically without updating the object's DATA pointer yourself. This works because the object processor will keep updating the display and incrementing the DATA field as long as the HEIGHT field is non-zero. (This requires a branch object before the bitmap object so that the proper number of scanlines are done during each frame.) You don't have to update the object until after the end of the last frame.

Wacky Questions We Removed From The Previous Sections

Q: The newer versions of RDBJAG cannot transfer data correctly to a Sylvester development system.

A: Boy, do you have an old system! If you are still working on a Sylvester, you should immediately contact Jaguar Developer Support to exchange it. The Sylvester is very outdated and should not be used for development any more.

Programming Guidelines

Below is a number of guidelines for Jaguar programming that have proven to be effective and efficient.

Filename Extensions

The use of standardized filename extensions for various types of files is strongly recommended. The table below shows the standard filename extensions used by Atari for all of our sample programs and libraries:

Extension	Filetype
.3DS	3D Object file for Autodesk 3D-Studio. Use the 3DS2JAG tool to convert into source code compatible with the Jaguar 3-D Graphics library.
.ABS	DRI/Alcyon format absolute location executable program file. Output from ALN Linker. This extension has also been used by some people for raw binary ROM image files, but this usage is discouraged.
.ASC	ASCII version of Jaguar Synth sound patch. This is a MADMAC source code file that is typically included by one of the source code files used with the Jaguar Synth and Music driver.
.AVI	Microsoft Video For Windows film file.
.BIN	Binary data. This could be a binary image of program code, data, a picture, or whatever. The LTXCONV utility used with the GASM assembler creates .BIN files containing the combined TEXT & DATA sections of the assembled file(s).
.BPG	BPEG Compressed image file. BPEG is the current flavor of JPEG used with the Jaguar.
.C	C source code file.
.CCR	Chunky-format 16-bit CRY Cinepak film
.CMP	Compressed sound sample, created from a raw 16-bit (stereo or mono) sound sample file using the SNDCMP utility.
.COF	Common Object File absolute location executable program file. Output from ALN Linker.
.CRG	Chunky-format 16-bit RGB Cinepak film
.CRY	Madmac source code file for a CRY-format graphics image, typically converted from Targa format using the TGA2CRY utility.
.DAS	DSP assembly language source. This extension is used for files that contain source exclusively for the DSP, to be assembled by either MADMAC or GASM.
.DB	Debugger script file.
.DTA	Binary image of program DATA segment. Created by FILEFIX utility.
.ENV	Envelope definition file. Used by the Jaguar Synth & Music driver.
.GAS	GPU assembly language source. This extension is used for files that contain source exclusively for the GPU, to be assembled by either MADMAC or GASM.
.H	C include file.
.INC	Madmac/GASM include file. Typically used to contain equates and macro definitions.
.J3D	3D object data in MADMAC assembler source format. Output from the 3DS2JAG utility. Must be compiled by MADMAC.
.JAG	Jaguar JPEG compressed graphics image. Created by the JAGPEG utilities. (Note that JAGPEG has been replaced by the BPEG package. Also, the 3DS2JAG utility that converts Autodesk 3D Studio into source code format for the Jaguar 3D libraries once also used the .JAG extension (it has since been changed to use .J3D).
.LTX	GASM assembler output file. The GASM assembler does not output files that are compatible with the ALN linker, so .LTX files must be converted using the LTXCONV utility.

Extension	Filetype
.LZJ	LZSS Compressed data file. This is a binary file containing raw LZSS-compressed data. It is created by the LZJAG utility. This is linked into your program, and then decompressed using the DELZJAG routines.
.MID	MIDI score file. This is a MIDI file output by a MIDI sequencer. You feed these files to the PARSE utility to create a music score usable by the Jaguar Synth & Music driver.
.O	68000/mixed object module. Object file created after assembling a .S file with MADMAC Some of the conversion utilities create MADMAC source code files that don't always end in filename extensions of .S, and they may also use the .O filename extension after being assembled.
.OJ or .OD	DSP (JERRY) object module. Object file created after assembling a .DAS file with MADMAC (Note that GASM does not create standard object modules.) Some older projects have used an extension of .OD for DSP object code. However, the .OJ extension is preferred.
.OT or .OG	GPU (TOM) object code. Object file created after assembling a .GAS file with MADMAC. (Note that GASM does not create standard object modules.) Some older projects have used an extension of .OG for GPU object code. However, the .OT extension is preferred.
.OUT	Parsed MIDI file, output by the PARSE and MERGE utilities. This is really a MADMAC source code file which is normally assembled into an object file using a .SCR extension
.PTC	Jaguar Sound Tool Patch File. These are the binary patch files used by the Jaguar Sound Tool and the Jaguar Synth.
.ROM	Alpine Board/ROM Image File. Created by FILEFIX utility, or saved from Alpine board using the debugger. Using the debugger, a ROM file can be loaded into Alpine board by "**read <file>.rom 802000**" or "**fread <file>.rom 802000**" *(FREAD uses faster I/O routines)* Using the debugger, a program can be saved to a ROM file from an Alpine board by "**write <file>.rom 802000[1FE000]**" for a 2 megabyte (16 megabit) program or "**write <file>.rom 802000[3FE000]**" for a 4 megabyte (32 megabit) program.
.S	68000/mixed assembly language source. This extension is used for files that contain source either exclusively for the 68000 or mixed source for any combination of 68000, GPU, and/or DSP.
.SCR	Smooth-format 16-bit CRY Cinepak film (Note: This file extension is also used in some cases to designate object files containing music data.)
.SCR	Compiled MIDI score file. This is an object file, the same as .O files, except with a different extension to highlight the idea that they contain MUSIC score information. Files with an .SCR extension are to .MID files as .S files are to .O files. Note: This file extension is also used for some Cinepak Movie Files (Smooth CRY-format).
.SRG	Smooth-format 16-bit RGB Cinepak film
.SYM	Symbol Table File. Created by FILEFIX utility. This is the same basic format as an executable program file, except with empty TEXT and DATA sections. Only the symbol table has information in it.
.TGA	Targa picture file. The Targa format is a popular format for 16-bit and 24-bit RGB true color graphics images. Can be converted into Jaguar CRY-format using the TGA2CRY utility.
.TX or .TXT	Binary image of a program's TEXT segment. Created by the FILEFIX utility. The current version of FILEFIX produces files with a ".TX" extension. However, older versions created files with the ".TXT" extension. Because the .TXT extension is also used for ASCII text files, this was changed to avoid conflicts.

Extension	Filetype
.WAV	Waveform definition. Used by the Jaguar Synth & Music driver.

Please do not use the filename extensions shown above for file types other than those shown. This can be a cause of great confusion. Perhaps the most common misuse of filename extensions is using ".ABS" for ROM image files that should have ".ROM" extensions.

Basic Testing For Jaguar Programs

It is important that your Jaguar programs run at the proper address, start themselves correctly, and do not try to write data at runtime into the ROM address space. With a development system, it is possible for a program to do any or all of these things, and you may not even realize it's a problem until you try to execute your program on a standard retail console. The earlier your programs avoid such problems, the easier the task is.

Below is a short basic test procedure that should be tried frequently with all programs destined to become a cartridge. It is by no means a complete and comprehensive testing procedure, but it will confirm the basic operation of your program.

1) Set the Alpine's memory protection switch to "Write Enable".

2) Download the code & data to the Alpine board. Make sure you are not downloading code or data directly to the console's DRAM (i.e. memory addresses from $200000-down).

3) Set the Alpine's memory protection switch to "Write Disable".

4) Turn off the Jaguar console. Wait for about 20 seconds.

5) Hold down the 'B' button of Joypad #1 and turn the console power on.

6) The standard Jaguar startup screen should appear with the Atari logo and spinning Jaguar cube. Release the 'B' button. Now press and release it again.

7) Your program should now start immediately. If it does not operate as expected, then you have a problem that needs to be solved. This can include: trying to write to ROM, being at the wrong address (your programs must start at $802000), or having bad or incomplete startup code.

8) Hold down the 'B' button of Joypad #1 again, and hit the RESET button on the top of the Alpine board. You should see a repeat of steps 6 and 7.

The steps above should be the first stage of your overall test procedure. Of course, once your program is known to pass this test, you need to subject it to a variety of more complete and more sophisticated tests. No Jaguar program should be released to the public without having first passed a comprehensive testing procedure.

Jaguar Programming Tips & General Procedures

The following is a list of several tips for Jaguar programming. Some might seem obvious to experienced Jaguar programmers, but there are also some new tips that reflect newly discovered bugs or simply better methods of doing things.

1) In order to guarantee proper system initialization, every Jaguar program must start out with the standard startup code supplied in the JAGUAR\STARTUP directory of the standard Jaguar Developer distribution.

2) Every object list must start with two branch objects. The first one should branch to a stop object if **VC < a_vdb**, and the second should branch to a stop object if **VC > a_vde**. The *a_vdb* and *a_vde* variables are calculated by the video initialization routine shown below in item 3.

3) Use the blitter in phrase mode whenever possible (it is much, much faster).

4) Because of a blitter bug, you must always set A1_CLIP to 0 prior to each blit, even if you aren't enabling clipping in the B_CMD register.

5) Don't rebuild your entire object list every vertical blank. Only update the individual fields of the objects that need to be updated.

6) The GPU and DSP may not be reliably stopped once they are running by anybody but themselves. This is a recently documented bug. GPU or DSP code which needs to run most of the time but be stopped occassionally should monitor a semaphore and shut itself down when the semaphore is given the "shutdown" value. (Alternately, a GPU or DSP interrupt could be used to tell the GPU or DSP to shut themselves down.)

7) The YPOS field of GPU Interrupt Objects was misdocumented as existing. This field does not exist. You can use branch objects to simulate the result of that field.

8) In order for GPU or DSP interrupts to be handled, those processors must be running. If no program other program is running and you want interrupts to be handled, leave a small piece of GPU or DSP code running that continuously checks a semaphore to determine whether it is OK to shut itself off. Keep in mind that as long as the semaphore is in internal RAM, this uses no bus bandwidth, so it shouldn't affect the rest of the system at all. Do not put either the GPU or DSP into a tight (i.e. one line) infinite loop.

9) When copying data to GPU or DSP RAM or I/O registers, always copy long words.

10) When the Jaguar console resets, the interrupt stack pointers of the GPU and DSP are in an undefined state. Always initialize these registers as needed.

11) Avoid creating object lists at assembly-time which are used directly from ROM at runtime. The bus access speed for ROM is much slower than for RAM (up to 10x slower), and the amount of time required to process your object list will increase dramatically, and some object lists may not function at all. Always create your object list in RAM (or copy it to RAM before using it).

12) Avoid displaying bitmapped graphics directly from ROM. Because of the greater bus access times required for ROM, a bitmap object with data in ROM will use up the system bandwidth available to the object processor (and therefore the bandwidth available to the rest of the system) very quickly. To save ROM space, compress the images using the JAGPEG or LZJAG utilities, and then decompress them from ROM into a RAM buffer, from which they get displayed.

13) Use the GPU and DSP as much as possible, instead of the 68000. The optimal solution is to use the 68000 to get the program started and load some code into the DSP and/or GPU, and then shut the 68000 down using the STOP instruction.

However, if you are using the 68000 a lot, or are using it for time-critical routines (like a vertical blank handler), copy your code from ROM to RAM and execute it there. That way, the memory accesses done by the 68000 to read instructions will hog less of the system bus, leaving more bandwidth available for the object processor, blitter, DSP, and GPU. Your code will also execute more quickly.

14) To save ROM space, compress your code using the LZJAG utility and then decompress it from ROM to the execution address in RAM.

Ideas To Try

1) If you have a lot of blit operations to be done, especially from different processors or from interrupts, rather than wait around for the blitter to be available each time, when you could be doing other processing, implement a GPU-interrupt routine that reads blit requests off a stack and sets up the blitter registers and starts the blit operation for you. Here are the basic steps involved:

a) Define a structure that contains the values that need to be stuffed into the blitter registers. Also include a pointer to a semaphore variable.

b) When you need to do a blit, set up one of these structures, and stuff a pointer to it into a variable. Clear your semaphore, and then force a GPU interrupt.

c) The GPU interrupt handler will grab the pointer to the structure and stuff it into a stack. If the blitter is currently busy, the interrupt exits. If the blitter is currently free, the GPU interrupt handler pops the pointer back off the stack, reads the structure, and stuffs the blitter registers to start the blit. The interrupt handler will then exit.

d) When the blit is completed, another GPU interrupt will occur (you must set bit 8 of the G_FLAGS register to enable this). The interrupt handler will grab the pointer to the semaphore for the just-completed bit, and stuff a value into it that indicates that the blit is finished. If there are any more blit requests waiting on the stack, the interrupt handler will grab the next one off the bottom of the stack and get it started.

Of course, this is just a rough outline, so the details are glossed over a bit, but you should get the

basic idea. Steps **c** and **d** are done more or less invisibly to the processor and code that requested the blit in the first place. As long as your actual calculations aren't affected by blits that aren't completed yet, you'll never have to wait for the blitter. Also note that using a GPU interrupt to put items onto the stack isn't really necessary if all your blitter requests are coming from the GPU in the first place.

Jaguar Atari-Based Development System Information

This section focuses on the differences between the standard PC/MSDOS-based development system and a development system based around one of the Atari computers.

First of all, with only a few exceptions, the documentation for the tools applies to both the PC/MSDOS version and Atari TOS version. In those instances where there are differences, they are noted.

General Guidelines

A standard component of MSDOS is the command line interpreter COMMAND.COM. On the Atari, there is no corresponding system shell; programs are normally launched through the GEM Desktop, part of the system's GEM graphic user interface.

Without a full-blown integrated development environment of some kind, a command line interpreter is essential for development work. Therefore, for the Atari we provide GULAM, a command line interpreter patterned after the UNIX C-Shell. GULAM is launched from the GEM Desktop like any other program, and once loaded takes over the system with its own text-based screen. GULAM uses UNIX-style commands rather than MSDOS-style, but supports command name aliases, so you can customize this to suit your own preferences. Please see the GULAM-specific documentation for more information.

Also provided for the Atari is a version of MicroEMACS, a popular text editor. The GULAM shell actually has a version of EMACS built-in, but the one we provide separately is more recent and more sophisticated.

Currently we provide the GNU GCC cross compiler that runs on PC/MSDOS systems and generates 68000 code. We do not currently provide the GNU GCC compiler for the Atari computers. However, the standard Atari version of GNU GCC used for building programs for the Atari TOS computers can also be used to generate code for the Jaguar. We consider it likely that developers who prefer the Atari-based development system are going to already have the Atari version of GCC. However, if you do not have the Atari version, and do want to work with it, let us know. Other Atari-based C compilers can also be used to generate 68000 code, provided they can output either DRI or COFF-format object modules.

Please stay in touch with the Jaguar Developer Support people at Atari. We are looking forward to helping you to make your product a software experience that takes the utmost advantage out of the Jaguar's excellent hardware.

Jaguar Development Standards

To insure consistency and to maintain the high quality of Jaguar software, the following standards must be adhered to by all developers: Please ensure that you contact Jaguar Developer Support *before* submitting code for Compatibility Coding if you have any questions regarding these guidelines.

Items shown in *italics* apply to titles published by Atari and must be adhered to by Atari-contracted developers, in addition to the other standards.

1) The title screen must contain all necessary copyright information:

- *The phrase "Licensed to Atari Corp." must follow the copyright information on games licensed to Atari Corp.*
- The phrase "Licensed by Atari Corp." must appear following the title screen on third-party Licensee titles.
- Programming credits may be included as desired, but they cannot replace or precede copyright information.
- *The title screen(s) must be the first visible screen(s).*

2) The "0" button should be used on the title screen to toggle game music off and on; game sounds are unaffected by "0". The default condition of the music (upon boot-up or Restart) should be on. If the "0" button is not used in the game, it should be used to toggle game music off and on during all other game play screens as well.

If the music is toggled off by the "0" button, the music volume slider should go to "0" volume as well. Alternately, the volume slider can remain fixed at the current volume and the message "mute on" can be displayed.

3) The Restart function of simultaneously pressing the "#" and "*" buttons should reset you back to the title screen. The order in which the buttons are pressed should not matter. Reset should occur immediately.

4) When the Pause button is pressed, all game actions must immediately stop and the word "PAUSED" must be displayed in the center of the screen. When the button is pressed again, all game actions should immediately resume and the word "PAUSED" should be erased from the screen. The Pause indicator should be of such color and size that it is easily seen. It is helpful to game magazines if pressing the 1 and 3 keypad keys while paused removes the pause message to facilitate screen captures.

5) Pause and Restart should be allowed anytime during a game with the exception that Pause is not necessary on the title screen.

6) We require a demo mode in all games showing some game action. This should be automatic from the title screen after a brief time of no user action and can also be an option on the option screen. Without a demo mode, retailers are much less inclined to have your game in the machine in their point-of-sale display.

7) Please ensure that any text you may display during the game can be read easily over all backgrounds. Either a contrasting color scheme or an outline around the text is recommended.

8) The "Completion of Game" logic should work as follows: When the game ends, there will probably be a "Congratulations" screen, or a high score screen. No matter what screen is shown, you must construct the end of the game so that the user cannot bypass any "Congratulations" text or High Score screen accidentally. Make the program work such that a Restart is required to return to the title screen from the "Congratulations" screen, OR implement a timer which ignores all input for a period of time (except timer wouldn't restrict Restart) so that the user does not miss any valuable information.

9) For normal "Game Over" screens, allow any fire button press to return you to the title screen.

10) For multi-player networked games, use the Modem/Networking developer guidelines.

11) We recommend that the high score screen displays the current version number on the title screen during final testing. If there is no high score screen, the version number can be displayed in the "Pause" screen. This version number must be removed prior to release of software.

The last digits of the top high score in the default high score table should be the version number of the software.

12) Joystick port 1 is to be used for a one-player game. Joystick port 2 is to be used for the second player in a two-player game. See the Enhanced Joystick/Multi-player Adapter documentation for further details.

13) The "B" button should be used as the primary action button; the "A" button should be used as the secondary action button. The "C" button should be used as the third action button. If a button is not used then it should be used as another "B" button. There must be an option to allow users to reconfigure the default settings.

Buttons must be implemented this way.

14) When the game is paused, pressing the "A" button should bring up a visual indicator and allow the user to adjust music volume via the joypad. Pressing the "B" button should bring up a visual indicator and allow the user to adjust sound effects volume. The "C" button can optionally be used to adjust a specific sound such as engines or voices. The indicator should be removed by the same button that brought it up. The volume level information should be saved when the high score or controller configuration information is written to the cartridge EEPROM.

The visual indicator used for adjusting music volume or sound effects volume should be a horizontal bar.

15) The "Option" button should be used to take the user to the Option screen. There should be an option to reconfigure the default joypad controls. This should also be saved to cartridge.

This option should be allowed during Pause also.

16) The stored information in the EEPROM should be cleared if the user simultaneously presses #, *, and Option at the title or options screens. A message "Cartridge Memory Cleared" should then be displayed.

17) The EEPROM data must be checksummed. If it is invalid or the EEPROM has timed out due to wear or failure, the default settings should be used. The game must never hang due to EEPROM fault.

18) We recommend using the keypad for passwords.

19) The NTSC and PAL versions of a game must both be in the same cartridge.

20) *If a game has a save game feature, it must be allowed only when the game is paused. A message "Game Saved" should be displayed below the paused message when the game save feature is activated.*

21) *Any game of a graphically violent nature must contain a parental lockout code. Default is "Lockout On". The code must be changeable by a parent following instructions in the game manual. Under lockout no extreme violence is displayed. The code should be enterable only on the option screen.*

JAGUAR SOFTWARE EXPERIENCE Approved Manufacturer Production Guidelines Version 1.5, October 18, 1994

I) COMPATIBILITY CODING AND CONTENT VERIFICATION

Publisher will send to Atari:

1. Code on either floppy (for cartridges) or CD master (for CD ROM). ROM image (.ROM file, single contiguous file containing executing at $802000) on floppy must be ZIP'd and spanned across floppies using PKZIP v2.04 or greater.

2. Two sets of blank floppies, EPROMs (150ns or faster) or CD masters so we can return compatibility coded version of title.

3. Completed Code Submission Form and affidavit of Content Descriptor (see section III for information on Content Descriptor).

4. Documentation of testing procedure and proof that the testing procedure has been adequately satisfied.

5. Instructions to play submitted software.

Atari will:

1. Review game to see if it adheres to the Jaguar Development Standards document guidelines (for fire button use, etc.)

2. Perform a hardware compatibility verification.

   If code is accepted for compatibility coding, Atari will compatibility code and return it to the publisher.

   If code is rejected or other problems are found, an anomaly form will be faxed to the publisher. The publisher then can correct the problem(s) and resubmit code. If they need to resubmit code more than once, Atari will charge $250 per additional submission for re-review of the code and compatibility coding.

II) GIFT BOX

Option 1

Please see Jaguar Product Style Guide for Atari recommended box design.

Option 2

Publisher's custom designs are allowed with prior Atari approval.

General guidelines:

The Jaguar logo must appear on the front of the box in dimensions no less than 2.5"w x 1"h. It may not be obstructed by other artwork. Other licensor logos (such as QSound, Cinepak, etc.) appear on the back of the box.

"Interactive Multimedia Cartridge" must appear across the bottom edge of the box front.

The Jaguar Compatibility Assurance Hologram (see section IV) must be affixed to the front of the box

III) CONTENT DESCRIPTOR (Subject To Industry Rating System Proposal)

Atari will not censor content; publishers should make themselves aware of local laws concerning entertainment media content.

Atari does reserve the right to withhold the use of the Atari and/or Jaguar logos to protect the goodwill of the Atari name and contradictory trademarks. The publisher must still properly use the Jaguar Compatibility Assurance Hologram and must adhere to all stipulations set forth in the Third-Party Licensing Agreement.

Upon submission of the Software Experience for compatibility coding and verification, the Licensee must also submit an affidavit stating that the Content Descriptor does accurately reflect the content of the Software.

EXAMPLE CONTENT DESCRIPTORS...

- General Audience Material
- Adult-Oriented Themes
- Adult/Sexual Situations
- Sexual Themes
- (Graphic/Comical/Light) Violence
- Adult Language
- Partial Nudity
- Explicit Sexual Themes

IV) MANUFACTURING

Cartridges & CDs ("Product")

Option 1

Atari will handle all manufacturing, based on Licensee's ROM or CD-ROM master and production-ready film. Atari will charge our cost, plus a 10% handling fee.

Option 2

Licensee can handle manufacturing themselves. The following services are available from Atari-approved sources:

Cartridge Shells (cost: approximately $0.32 each)
Source: Stoesser Industries; Contact: Robert Stoesser; Phone 415-969-3252
Their supplied casings conform to Atari specifications. Publishers can order custom plastic colors, or have their own logo appear in the molding of the cartridge simply by purchasing a low-cost insert from Stoesser.

ROMs
Sharp; Contact: Paul McCartney; Phone: 408-452-6409
Samsung; Contact: Lori Steinthal, I-Squared Mfgr. Rep.; Phone: 408-988-3400, x223
MX Macronix Inc.; Contact: Ray Mak; Phone: 408-453-8088
Goldstar; Contact: Y. Kenneth Kim; Phone: 408-432-1331, x3603

Standard Cartridge PCB's
Atari will supply board layout information; Licensee must submit manufacturing samples to Atari for approval. We will also be happy to provide direct sources for PCB's.

CD-ROM
WEA/Ivy Hill; Contact Atari for sales office for your territory.

Cartridge Turnkey Service
Extron Manufacturing (contact: Thao Nguyen; phone 408-456-0180) has been designated as a fully approved manufacturer under the provisions of the Jaguar Software License Agreement.

Publisher can create their own cart internal (PCB , etc.,) design, but it must be submitted, registered and approved by Atari prior to manufacturing to ensure compatibility with future revisions of Jaguar. This will accommodate Publishers wishing to create special carts with battery-backed up SRAM, etc. Some of these alternative designs may already be available to Licensees; call for availability.

All ROM and CD-ROM duplication must be performed by the Atari-approved vendors. Publisher shall have the right to have a ROM or CD-ROM duplicator qualify as a Manufacturer under this Agreement upon proof of the following:

1. That manufacturer is properly licensed by Philips/Sony for CD-ROM, if applicable;

2. That the manufacturer can maintain reasonable quality assurance standards.

3. That the manufacturer agrees to such reasonable security and reporting requirements to assure that compliance with the royalty provisions of the Jaguar Software License Agreement are implemented and verifiable by providing any information relating to production of Jaguar ROMs or CD-ROMs when requested by Atari; and

4. That the manufacturer agrees to maintain Atari's intellectual property rights.

Atari shall reasonably assist any manufacturer advanced by Licensee to become a manufacturer, however under no circumstances shall Atari have liability for the conduct of the manufacturer. Atari shall inspect the manufacturing facilities prior to approval. Please allow 60 days for the approval process.

Jaguar compatibility assurance holograms (see next section) must be affixed to the front of the point of sale box.

V) COMPATIBILITY ASSURANCE HOLOGRAMS AND ROYALTY

1. Jaguar Compatibility Assurance Holograms must be ordered from Atari via fax (1-408-745-2088). Holograms are ordered on a by-title basis to track royalties via Atari-assigned serial numbers.

2. Holograms are ordered in opening orders of a minimum 2000, reorders are in multiples of 1000. Holograms are 12 cents ($0.12) each.

3. Holograms will be delivered generally within 3 working days.

4. Publisher will be billed for holograms and royalty at time of shipping, to be paid in accordance with the terms of your License Agreement.

Additional Documentation

The following additional documents are also included with the Jaguar Developer's Kit or are available separately:

DB: The Atari Debugger

