in

COLLABORATORS			
	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 27, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 in **1**

1.1 Table of Contents . . . 1
1.2 Legal/Distribution Information . . . 1
1.3 Introduction . . . 2
1.4 The System CPU . . . 2
1.5 Coprocessor Devices . . . 3
1.6 Bus Interfaces . . . 3
1.7 I/O Devices . . . 4
1.8 System Memory . . . 4
1.9 Cache Memory . . . 5
1.10 Clock Ratings . . . 6
1.11 System Entirety . . . 7
1.12 The Amiga . . . 7
1.13 asyscpu . . . 8
1.14 Amiga Coprocessors . . . 10
1.15 Amiga Custom Chips . . . 11
1.16 Amiga Bus Layouts . . . 12
1.17 Summary . . . 14
1.18 AIBB Overview . . . 14
1.19 System Requirements . . . 15
1.20 Getting Started . . . 16
1.21 Main Screen . . . 20
1.22 perfgraph . . . 20
1.23 testinfo . . . 20
1.24 basemachind . . . 20
1.25 compinfo . . . 21
1.26 basicinfo . . . 22
1.27 testgads . . . 23
1.28 mainmenus . . . 23
1.29 menu1 . . . 23

1.30 menu2 24
1.31 menu3 25
1.32 menu4 26
1.33 sysinfodisp 29
1.34 testoverview 33
1.35 compsystems 39
1.36 notes 40
1.37 credits 41

Chapter 1

in

1.1 Table of Contents

A.I.B.B.
Amiga Intuition Based Benchmarks
A system performance evaluation utility for the Amiga
Program Release Version 6.2

Table of Contents

Legal/Distribution Information
AIBB/Benchmarking Introduction
Intro to the Commodore Amiga
Overview of AIBB
System Requirements
Getting Started
Main Screen Description
The System Information Display
Overview of Included Tests
Included Comparison Systems
Notes and Summary
Credits and Acknowledgements

1.2 Legal/Distribution Information

Legal and Distribution Information

This software is provided as is. No warranty as to the performance or validity of data obtained within is stated or implied. Bug reports and suggestions for improvement are welcomed, and every effort will be made to evaluate such reports.

AIBB is freely distributable provided no fee other than a moderate fee for disk copying charges is made for its acquirement. It may be distributed across any electronic network, provided no fee is charged specifically for it's download. A broad-based download fee is acceptable provided it is charged universally for all such file downloads. All associated files included with the distribution archive of AIBB are to remain intact and unaltered. BBS listing notices and the like may be included in the archive provided no alterations are made to the actual distribution files themselves.

This program, and all accompanying files are not public domain. They are copyright material and may not be used for commercial purposes without

permission from the author. In most circumstances such permission will be granted, but the author must be contacted before any distribution with a commercial product.

AIBB is not shareware, as no donation or usage fee is required. However, any donations are always appreciated, and can only encourage further development of the program. This is an ongoing project, and will continue to be so as long as interest in it is shown.

1.3 Introduction

Introduction{UB}

AIBB is a utility primarily designed to assist in the evaluation of system performance on a basic level. It consists of a series of performance tests , the results of which are evaluated against other systems and the displayed for comparison purposes. It should be noted that care must be taken when making a definitive evaluation of the performance of any system, as much more is involved in making a thorough determination than the data which can be provided by AIBB alone.

System performance evaluation, commonly referred to as "benchmarking", is the rather dubious science of trying to determine which system or system architecture is "fastest". Unfortunately, all to often it is not completely clear what is meant by which system is "fast".

Computer systems in general usually consist of a number of devices interconnected to form a whole. These individual devices can be on one circuit board, such as the case with certain coprocessor devices, etc... or even as seperate entities completely, physically connected in some external fashion, such as with expansion boards. All of these devices will have certain advantages and disadvantages with respect to performance levels. Combined together, it is generally the overall use of the system in general which determines how much of an effect is seen in these factors when observing overall system performance. Before delving into these factors further, it is necessary to first clarify a few of the key components and factors which are main players in the performance game, which include:

The system CPU
Coprocessor Devices
Bus Interfaces
I/O Devices
System Memory
Cache Memory
Clock Speed
System Entirety

1.4 The System CPU

The system CPU.

The CPU (Central Processing Unit) of a computer is often the focus of most performance discussions. This unit is generally responsible for the non-specific portion of any computing task. It’s duties involve general

program instruction execution, and in many cases it is the device responsible for 'mastering' the system and coordinating the system effort as a whole. Note that this is a generalization. Systems do exists which are distributed; their CPU is not as readily defined, or consist of multiple processing units each coordinated as a whole. However, in the context of this discussion a single primary device will be assumed.

Since the CPU of any system does often receive a great deal of the overall responsibility for program execution and task organization, it is thus a very key part in the overall performance of the system as a whole. However, often times it is considered solely as the factor which determines the "speed" a computer can perform a particular operation. This assumption is not always valid, and must be thought out carefully. Many other factors may affect the efficiency of the CPU itself in performing it's operations, which is why the system as a whole must be evaluated towards a particular job which it is to be given. But before this relationship becomes clear, the other components which are factors must first be recognized.

1.5 Coprocessor Devices

Coprocessor Devices

A coprocessor is any system processing unit which works in conjunction with the primary processor (CPU) in the actions of the system. Such devices are often subsystem-specific, and are responsible for a particular set of computing tasks. For example, a system may include a FPU, or Floating Point Unit to take on the task of floating point computations. These processors are generally fine-tuned to that specific task, and thus are more efficient at it than the main processor would be if it were to do the same job.

Thus, the primary use of coprocessors is to alleviate some of the total system computing load from the CPU. These devices may be directly coupled to the CPU, thus being closely tied to the performance of the master processor, or may be of a loosly coupled variety. This latter type of coprocessing unit is tied to the CPU only when it requires data and information from the main processor, and in some situations may be capable of accessing and modifying system memory without going through the CPU at all. Although this concept is not unique to coprocessors alone, it is relevant, and thus will be explained here. Such memory accessing capabilities denote a Direct Memory Access device (DMA). These devices do not necessarily rely on the CPU to transfer data to them, and thus are often 'decoupled' from the CPU in such a way as to have a different performance ratio from the CPU itself. Even non-DMA devices are often afforded a level of concurrent, or simultaneous operation with the main CPU, so as to provide a more efficient method of task completion. However, DMA devices are more closely tied with another set of subsystems to be considered when dealing with system performance, I/O devices.

1.6 Bus Interfaces

Bus interfaces

This is often a confusing topic. The term 'bus' is used a great deal,

but all to often it is not clear what is meant by it. As stated before, a computer system consists of a number of devices integrated together to form the whole. A bus is, simply put, a communications pathway between devices. Over these pathways control, address, and data signals are transferred to devices which are required to perform a portion of any particular task. Most systems contain more than one bus in which this communication takes place. Usually, a primary bus or combination of specific primary buses is responsible for the majority of data transfer and communications between all devices in general, with lesser buses used as specific pathways between certain devices. Buses are often 'sized', or given in terms of bit-bandwidth. Basically, this is a determination of the maximum size of a single data transfer across the pathway between devices. For example, an 8-bit bus can transfer an 8-bit quantity of data across it at once, while a 32-bit bus can transfer 32 bits at a single time (Where a bit is defined as an electrical signal value representing a binary number, either 0 or 1 [Logical FALSE or TRUE, which orientation depending upon the design of the system] for each bit). Although there are other sizing factors which come into play, this is a general idea, and suitable for the discussion at hand.

As any system relies on the coordinated efforts of all its components, the efficiency and effectiveness of communication between each device is of importance when considering the overall performace of the computer. A bus which is not up to par with the capabilities of the devices it interconnects will hinder the system while one which is capable of handling the individual components will allow for a more efficient setup. More of this relationship will be given later after the other component members are introduced.

1.7 I/O Devices

Input and Output (I/O) Devices

This is a lose subset of devices collectively describing such units as storage media devices (disk/tape drives, etc...), external communications devices (serial and parallel communications to external units), and specific control input units, such as keyboards and other data input means. While the latter of these devices is generally not considered to be of much influence in system performance, the former members, such as storage devices, can have a great impact on performance levels.

Storage devices are in general the slowest of data transfer devices on any system. For this reason they are often considered to be a 'bottleneck' in system performance evaluation. However, many advances have been made in the design of such units, including the use of DMA access from storage device control units to the system main memory, which helps by alleviating the CPU's responsibility in data transfer from these devices.

Generally, I/O devices are more important to systems requiring a great deal of access to large quantities of data, or ones involved in data transfer as their primary mechanism of use.

1.8 System Memory

System Memory.

This subsystem has been mentioned in passing previously, but until this section not given full attention. System memory resources also play a big part in overally system performance evaluation. Memory can affect a system's performance in many ways. Depending on the speed of other devices, utilizing memory subsystems which are slower (requiring the addition of 'wait states - periods of time in which the data requesting device waits for the data to be available - to properly interface to the system) can cause any data accesses to occur at a slower rate than the rest of the system could otherwise handle them. Many memory subsystems do indeed utilize wait states, as other devices are too fast for such memory and the memory access speeds required for zero-wait-state access would make for prohibitively expensive systems. Although a completely zero-wait state system is often not feasible, methods are available to system designers to try and reduce the overall memory latency periods. One widely used method is the use of cache memory.

1.9 Cache Memory

Cache Memory.

Cache memory is a memory storage medium which is usually designed for the fastest possible access to frequently used resources, usually microprocessor instructions and/or data. This area is generally small compared to the size of an entire system memory complement, and thus can be implemented at a cost lower than that of employing very fast components for all memory. The general operation of most memory caches is to store the most recently accessed instructions or data within the cache, then make a check for them there upon the next memory access call. In this sense, if the instruction or data is in the cache, it can be accessed almost immediately, rather than having the processor fetch the required data from the system's main memory resources. A cache 'hit' is the term used to indicate the processor did indeed find the data within the cache, and did not have to fetch from main memory, whereas a 'miss' denotes when the processor was forced to get the needed data or instructions from the main system memory. When a miss occurs, the cache will usually be updated with this new data in the case it is called for again, thus keeping the data in the cache fresh.

The main theory behind such caches is that many programs spend a great deal of time within the confines of a definable event loop. Therefore, depending on the size constraints, part or all of such a loop can be held within the cache, decreasing execution time. Caches can be found both external to the microprocessor or, increasingly, within the microprocessor itself. They may be seperated such that they only instructions or data are held individually, or may be set up such that both types of memory accesses are kept within one cache. There are tradeoffs to both types of design, but in general the cache in any form is a useful mechanism for increasing system performance. One must be cautioned however, as the cache can also lead to a misrepresentation of system performance comparisons. Benchmarking tools are often small segments of programs, and as such may be easily completely cached on systems equipped with such. Thus, a benchmark result may not accurately depict the true system performance with a real-world application which would not be

entirely housed within a such a cache.

1.10 Clock Ratings

A word on clocks and clockspeed ratings.

"Clockspeed" ratings of devices are in actuality frequency measurements. Almost all digital devices operating in a computer system today require some sort of timing input to coordinate their internal and external responses. Generally, this is provided by a clock signal fed to that device, and in some cases the device itself may be responsible for the generation of additional clock outputs to other devices.

Clock frequency ratings for system components are usually today given in terms of MegaHertz (MHz). This is a cyclic frequency rating indicating a the number of cycles per second an oscilating periodic signal undergoes. As an example, a rating of one MegaHertz indicates a frequency of one million cycles per second.

As indicated earlier, almost all digital system components require some form of clock input. To see where this is important, take the case of the CPU. Generally, instruction execution timing is stated in terms of the number of clocks a given instruction takes to complete. A faster clock means that although an instruction takes the same number of clocks to finish, more clock input edges occur in a given time frame, and thus afford a faster response. In this sense, faster clock rates generally indicate faster devices. The system bus, and other devices are also managed in terms of clock inputs signals. These may or may not be the same input as given to the CPU, or the CPU itself may control them itself. Thus, differences in clock ratings between subsystems can be a source of bottlenecking, if one faster clocked subsystem is forced to wait to synchronize with a slower subsystem in order to transfer data and control signals.

Let it not be thought that clock input frequency is the sole governing force in determining component speed, however. In many cases, other effects cause similarly clocked devices doing the same task to finish in differing amounts of time. One way this can happen is if one device has been enhanced in such a way as that it’s internal operations are more efficient, thus requiring fewer clocks to complete. Therefore, this factor must be weighed as well as clockspeed in even single device evaluations. Device designers are constantly using both increased clock rates, as well as increased internal efficiency to advance the performance of system components.

It should be noted here that the term "bus cycle" is often confused with the concept of of clockspeed, because of the term cycle. A bus cycle is related to the clock cycle rate, but not usually identical. Bus cycles are the time required for the CPU or other device to access data and complete an external bus operation on it. For example, the MC68000 CPU runs a 4 clock memory access cycle in general (asynchronous memory transfers), requiring 4 CPU clocks to access a given memory operand. This is assuming a no-wait state operation. Wait states are additional clock periods added to this cycle time in order for the data to be validly returned from the accessed device, and are placed in the bus cycle period when a device is incapable of responding to the data transfer request within the normal 4 clock period. This is only given as a particular example; other CPUs and architectures have differing bus cycle timing layouts (i.e, the MC68020, MC68030, etc... run 3 clock asynchronous bus cycles normally at zero wait

states).

1.11 System Entirety

Putting it all together.

Many factors are involved in the evaluation of a system's performance. But just as a computer is the sum of its parts, these factors cannot be considered alone. They must be put together and seen in entirety in order to get a whole picture. Moreover, the intent of the system in use is important in weighting these factors towards which are more influencing for any particular task.

As an example, consider a system primarily intended for data processing tasks. One might expect that it should have a relatively fast CPU in order to work through the data at a reasonable pace. However, if the system's memory resources are such that they require the addition of many wait states into their accesses, then some of the effect of having a fast CPU is offset. Even ↩ further, what type of data is being processed?
Then again, if the data is of a floating-point varieny, then a very fast CPU might not necessarily be as effective as a moderately fast Floating Point coprocessor added to the system. Another important factor might be the amount of data which needs to be continously accessed from storage devices. In the case where a great deal is being pulled from such devices, and they are slow in providing the data to the system, then no blazingly fast component elsewhere is going to be able to make that system setup mark high in it's environment as the data is only able to get to the 'fast' devices as fast as the 'slow' storage devices can provide it.

It is obvious that care must be taken in evaluating any system's performance in order to properly take into account all factors involved. This includes determination of the usage of the system, and how individual components may affect this speed.

1.12 The Amiga

The Commodore Amiga

The Commodore Amiga is a particularly interesting system as a whole to evaluate, as it houses a fairly complex architecture for its relative price range. It includes aspects of multiprocessing within it's design, as well as a multitude of different system layouts to consider. However, only subsystems relevant to the type of testing performed by AIBB will be considered here, these being the 'core' elements of the system, discounting I/O devices and external communications units. Of primary interest in this discussion are the:

Amiga System CPU ,
Coprocessing devices ,
Custom Graphics Chips ,
and Amiga Bus Layout .

A Summary of this and all benchmarking issues discussed so far is given at the end of this section.

1.13 asyscpu

Primary Amiga system processors.

The Motorola M68000 series of microprocessors are utilized as the main CPU in all Amigas in production today. Various models of Amigas exist which utilize all of the primary variants of this microprocessor family, with third-party add-on accelerator units providing an upgrade path for many systems originally borne with earlier 68000 series CPUs. An overview of the various M68000 microprocessors and their main uses in Amigas is as follows:

MC68000/MC68HC000

The MC68000 was the CPU the Amiga was born with, utilized in the Amiga 1000 first, and subsequently in the A500 and A2000 stock system models. This CPU is characterized by a 24-bit address bus, giving it a 16 megabyte addressing capability, and a 16-bit data bus. This microprocessor is classified as being a 16/32 bit device. Its external data pathways are 16 bits in size, while internally it supports a 32-bit model by containing full 32-bit register implementations.

In all stock Amiga models utilizing this CPU, the device is clocked at the rate of the system bus, approximately 7.15 MHz for NTSC based systems, and about 7.09 MHz for PAL systems. Certain add-on accelerators do exist which are built around this CPU, replacing the stock motherboard component with an add-on board which runs the CPU at 14.28 MHz, or in some designs, 16.0 MHz.

Recently, the MC68HC000 variant of the 68000 has been introduced into the Amiga market on an accelerator board. The 68HC000 is a standard 68000, but manufactured in CMOS technology. This design of the part allows it to run at higher clock rates, and with less power consumption than the standard 68000. Aside from this, the 68HC000 is identical to the 68000 stock device.

MC68010

This CPU has not seen wide use in Amiga systems, although it can be found occasionally. The MC68010 is pin-compatible with the MC68000, allowing for simple drop-in replacement in any system utilizing the latter. Most systems do not see a tremendous performance boost while utilizing the 68010 as it’s improvements over the 68000 are not a tremendous leap.

The MC68010 includes various internal microcode enhancements over the MC68000, allowing for faster instruction execution in some circumstances, as well as the addition of a specialized programmer-transparent ’loop mode’ which enhances CPU performance in tight program loops by allowing said loops to be latched into the CPU instruction prefetch queue where external bus cycles are not necessary for the loop code proper. As indicated earlier though, this CPU has not seen a great deal of use in Amiga systems, and is mostly found in circumstances where owners of 68000-based Amigas have chosen to replace their stock CPUs with this device directly.

MC68020

A major upgrade to the line, the MC68020 includes a great many advances over the previous members of this microprocessor

family. The MC68020 is the first fully 32-bit capable microprocessor of the M68000 series, incorporating full 32-bit address and data buses, as well as a 256 byte instruction cache, in order to keep program code sections used often within a fast-access medium. The MC68020 is a major step above the MC68000 or MC68010, with an architecture more capable of handling larger demands upon its resources.

The 68020 is utilized in earlier acclerated Amiga systems, including as the main processing engine of the first A2500 series of machines which housed the CBM A2620 accelerator unit. Many acclerators using this CPU were produced by third-party manufacturers, including low-cost units found in some A500 units, as well as in the A2000 line. In most designs, this CPU is clocked at approximately 14.28 - 16.0 MHz, with a few of the lower-cost accelerators running the CPU at the ~7.15 MHz (NTSC) / ~7.09 MHz (PAL) system clock of the Amiga.

MC68030

Improvements were made to the MC68020, including the addition of a 256-byte data cache to complement the existing instruction cache, and the inclusion of an on-board memory management unit (MMU) in order to produce the MC68030. Additional improvements exist internally to this CPU over the MC68020 to give it a stand against its generation of competing microprocessors. The 68030 can be viewed as an incremental improvement to the 68020, adding additional features but not being a tremendous architectural change from its predecessor.

The MC68030 is found as the accelerated CPU of the later A2500 series of Amigas, as well as being the main processor of the Amiga 3000 line. This microprocessor has also been widely implemented in accelerator units for all models of Amigas and is used at a wide variety of clock frequencies ranging from 16.0 MHz to 50.0 MHz.

MC68040

Currently found in a variety of accelerators, and as the main processor for the A4000/040, the 68040 is a generation leap over the previous MC68030 model and incorporates a great many advances over all previous models in this series of microprocessors. Both instruction and data caches found in the MC68030 are present, but their size has been increased to 4K bytes each. In addition, the data cache of this processor now supports a 'CopyBack' mode of operation, providing for faster data access times by allowing memory writes to be deferred to the cache until an update of memory contents is absolutely required. On-chip MMUs exist for both data and instruction streams within the CPU, and the internal pipelines have been further optimized for increased performance. A subset Floating Point Unit (FPU) is also included on-chip for floating-point calculations.

The 68040 is at present found in only 25 and 33 MHz rated varieties at this writing, though this will likely change in the future. Unfortunately, it does seem to be a developing trend in the Amiga community to somewhat overclock the 68040, an action neither sanctioned nor recommended by Motorola.

There are several variants of these primary microprocessor models in production. The newest such variants are the Motorola "EC" series of

M680x0 parts, and "LC" series of MC68040 parts. The "EC" (Embedded Controller) series are characterised by changes from the standard part ranging from simple packaging to the removal of certain internal features. This latter option is what has been taken with the MC68020, MC68EC030, and MC68EC040 parts. The MC68020 is given by a 24 bit address range, as opposed to the normal 32 bit address range of the standard 68020 part. Aside from this difference, it is identical to the 68020. The MC68EC030 is characterized by the lack of an on-chip MMU. It functions identically to the standard MC68030 with this exception. The MC68EC040 and MC68LC040 are similar to each other except that the on-board MMUs of the normal 68040 are preserved, in the LC part, with just the FPU not functional on the unit, while the EC part removes both the FPU and MMU units from the chip.

At this point it is of interest to bring up a point of common interest with accelerated Amiga systems; that of asynchronous vs. synchronous accelerator designs.

Synchronous designs were the first accelerators to appear for the Amiga. These are generally found in the MC68020 based accelerator units, and also in many of the low-cost MC68000-based accelerators. A synchronous design is one in which the devices present on the accelerator are clocked at a rate which is absolutely synchronized to the main system clock signals. For the A500 and A2000, this means the clock rate of such accelerators must be an even multiple of the ~7.15MHz (NTSC) / ~7.09 MHZ (PAL) system clock rate. Because of the difficulties involved in maintaining synchronicity at high clock rates, generally these accelerator units are restricted to about 14 MHz, or double the system clock rate.

Asynchronous designs, on the other hand, have no such restrictions. These units are somewhat more difficult to design, but in general the accelerator components may be operated at nearly any clock input, provided they are themselves capable of performing at the given frequency. This operation mode is what all MC68030-based accelerator designs for the A500 and A2000 utilize, thus giving the wide range of clock rates found in these accelerators.

It must be noted however that an ambiguity exists in the terms synchronous and asyncronous. The 680x0 microprocessor series is characterized by normally running asyncronous bus cycles. This simply means the processor initiates a read/write action, and it is up to the external device to terminate (acknowledge) the cycle, thus completing it. This behavior is NOT related to accelerator design as might be confused by the use of the same terms. In accelerator design terms, asyncronous and synchronous are designating how the accelerator state machine relates to the main system clock, and NOT how individual bus cycles are run by the CPU in general.

1.14 Amiga Coprocessors

Primary Amiga System Coprocessors.

Many accelerated Amigas also utilize an FPU for floating-point math intensive operations. The main FPUs in use by the various Amigas available, and the add-on accelerators in use on the Amiga, are manufactured by Motorola as well, either as seperate coprocessor devices, or as in the case of the MC68040 are embedded within the main CPU itself. An overview of the various FPUs in use is given below:

MC68881

This is a seperate floating point coprocessor device which provides fast hardware-supported floating-point operations to any system software which supports it's use. This unit does provide a certain level of concurrancy, giving it the abililty to perform certain instructions at the same time the main CPU is performing other operations. Support for this coprocessor is provided either by a built-in hardware microcode interface, found on the MC68020 and MC68030, or by software trap interfacing for the MC68000 and MC68010. The latter method is used in but a few early Amiga accelerator boards, while the preferred interface, that to the MC68020 or MC68030, is supported by virtually all accelerators utilizing those CPUs.

The MC68881 may be run asynchronous to the CPU clock input, meaning it need not run at the same clockspeed as the CPU itself. Thus, a faster FPU may be used to give somewhat of a boost to floating-point operations. The MC68881s in use in Amigas today are found mostly running at clock frequencies ranging from 12-20 MHz.

MC68882

The successor to the MC68881, this unit incorporates the same interface and operations as the former device, but with certain internal enhancements. The microcode for many operations has been optimized for faster response, and support for further multiple floating point instruction concurrency was added. In general this FPU will perform at about 1.5 times the speed of the MC68881 at the same clock input frequency. The MC68882 is primarily operated at clock rates of 12-50 MHz, depending on the accelerator or system utilizing it.

MC68040

The MC68040 CPU incorporates an FPU within the processor itself. This FPU unit is a basic subset FPU of the MC68882, eliminating mainly the transcendental (sin, cos, etc...), and complex functions found in microcode on the former. Nevertheless, the optimized nature of the existing FPU instructions provided allow for emulation of the missing functions in such a way as to give faster execution than the MC68882 for almost all operations.

1.15 Amiga Custom Chips

The Amiga's Custom Graphics Chips.

In addition to the main processing units, the Amiga also incorporates a number of custom designed devices, known collectively as the Amiga's custom chips. Their primary purposes are varied, but they are generally in charge of such things as DMA access and arbitration to various memory areas, and graphics/sound generation and effects. These custom chips are:

Agnus/Alice

Probably the most talked about custom chip, Agnus is found in a number of flavors, ranging from the original device, to the 'super' version found in the A3000. Aside from minor internal changes, the main differences between these different versions is the amount of memory they can directly access. Agnus is

responsible for for control of 25 system DMA channels, generation of all system clocks in the A500 and A2000, and provides control and addressing for CHIP RAM, which is the memory accessable by these custom chips. The size of this memory region is determined by the Agnus in use, and is either 512 KBytes, 1 Megabyte, or 2 Megabytes in range. As the custom chips are utilized primarily for graphics and sound coprocessing tasks, all such data must be located in this CHIP RAM area.

Agnus also contains within it what is referred to as a Blitter. This internal device is a fast memory copy unit designed to move areas of memory as efficiently as possible, and has the capability to also perform specific logic manipulations to the data in the process.

Finally, Agnus also contains Copper. Copper is the system's Display Synchronized Coprocessor. This device assists with screen refreshes and display building, and is a major factor in the Amiga's graphics engine.

Alice is the successor to Agnus, and part of the AGA graphics chip found in the latest Amiga models. Containing the same 16 bit data bus interface to CHIP RAM, Alice is nonetheless capable of directing 32-bit fetches to RAM, as well as take advantage of double CAS page mode cycles, providing for a larger bandwidth to memory, and increased performance.

Denise/Lisa

The Denise custom chip is primarily responsible for color generation and display resolution modes. This chip also contains the eight hardware display sprite controllers used in the system.

Lisa, part of the AGA custom chip set, is the replacement for the aging Denise. This new chip is implemented in full CMOS technology, and incorporates the ability to handle up to 24-bit RGB video, as well as do double 32-bit fetch cycles to memory which increase its data bandwidth rate to 64 bits per cycle, or four times that of the earlier Denise chip.

Paula

Paula is a more or less diverse device. It controls sound generation, contains the system floppy disk control circuitry, and houses the I/O control circuitry for the disks as well as external control ports. Paula also contains an interrupt control system for various system operations.

The custom chips of the Amiga and the coprocessors associated with them are designed in such a way as to alleviate the main CPU of many intensive tasks, such as graphics operations and sound generation. They support a concurrent level of operation, allowing the main CPU to continue with non-specific computing tasks while the custom chips handle their respective operations. The devices are capable of DMAing directly into the CHIP RAM area, freeing the CPU completely from task responsibility in those respects.

1.16 Amiga Bus Layouts

Amiga Bus Layout.

The seperation of operations and the definition of the CHIP RAM memory area is further accentuated by the fact that the Amiga utilizes two buses along these lines. The CHIP RAM bus is a seperate entity from the main bus utilized by the CPU and other devices, but is accessable by the CPU as well. The seperation can even be greater given the fact that the CHIP RAM bus can be decoupled from the CPU bus completely under certain circumstances.

The CHIP RAM bus is primarily utilized by the custom chips, with the CPU being given access to it on an interleaved cycle basis (every other bus cycle can be a CPU access cycle). The custom chips have priority in this domain, and this is where the idea of bus contention arises. If a great deal of bus activity is in progress by the custom chips, they may 'lock out' the CPU, forcing it to wait if it needs data or information from this bus' memory space. This is where the touted 'FAST RAM' comes in.

FAST RAM is memory not on the CHIP RAM bus, but rather on the main system bus or expansion bus. This memory is not accessable by the custom chips, and thus no contention for it's access occurs between them and the CPU. Due to the seperate nature of the buses, it is possible for the CPU to be processing instructions and data utilizing FAST RAM while the custom chips are concurrently operating in the CHIP RAM area. This parallel operational status allows the Amiga to perform a great variety of graphics operations in such a way as to done on a bus which is not operated at a great speed.

The CHIP RAM bus on all Amigas is operated at a clock frequency of approximately 7.15 MHz. On the A500 and A2000, this is the main system clock frequency. For those machines, the CHIP RAM bus is accessed via a 16-bit wide bus port, while on the later A3000/A4000 systems the bus port for external accesses is a full 32-bit interface, affording larger data transfer sizes at the same clock rate.

Because of bus contention, a system containing only CHIP RAM may very well have slower operations than one which contains FAST RAM as well. The FAST RAM equipped machine will be capable of having the CPU operate concurrently on information on that bus, while the custom chips operate on their tasks. The CHIP RAM only system is going to have circumstances where the CPU will be forced to wait to access data, as the custom chips may be utilizing the CHIP RAM bus heavily.

FAST RAM in the A500 and A2000 series of machines can be located on many devices, from standard expansion card extenders which exist on the system expansion bus and operate at the system clock frequency, to other methods of RAM addition which have been devised that do not directly use the common Amiga expansion routes. FAST RAM located along the standard expansion backplane on these systems operates at the system bus clock rate (7.15 MHz), and is accessed accordingly. On A3000 machines, FAST RAM is generally located on the system motherboard, and is accessed according to the system clock rate of those machines, which on stock models may be 16 or 25 MHz.

It should be noted that some systems utilizing only 512K of CHIP RAM have in their memory lists a region of RAM which is called FAST, but in fact is on the same bus as CHIP RAM. This is generally the memory found on the A2000 motherboard for 512K CHIP RAM machines, or on the A501 expansion card for A500s. This memory will suffer from the same bus contention that CHIP RAM is exposed to, and thus it is generally advisable to be sure that program code is not put here unless it has to be (e.g, if true FAST RAM exists, it should be prioritized). The utility program "FastMemFirst" supplied by CBM is meant to do just that.

FAST RAM located within the domain of an accelerator is not limited to the system bus clock rate. It may be operated at such, but in general can

be accessed at a clock rate much different, usually at the accelerator's CPU clock. Systems utilizing accelerators benefit from this setup, as an accelerator does not change the system clock rate, and therefore in order for an accelerator's CPU to use system resources, it has to synchronize with the system clock, and may even have to contend with a narrower bus interface. Such is often the case on the A500/A600 or A2000 when utilizing MC68020 or MC68030 based accelerators, which are best suited for 32-bit bus ports. Since those processors take a performance hit when accessing narrower bus ports, as well as a hit from the possibly slower clock rate of the system bus, accelerators often are equipped with their own RAM resources which is designed to operate at the CPU clock frequency and utilizes a more efficient bus port size (32-bit). The case with the A3000/A4000 is slightly different.

The A3000 and A4000 utilize a 32-bit bus for their memory resources already, therefore this is not a problem with accelerators for those machines. However, the bus on the A3000/A4000 is clocked at 16 or 25 MHz (depending on the model), and if a faster CPU is used in an accelerator it may be profitable for the unit to contain it's own RAM resources in order to lower access delays to a minimum. The A3000/A4000 does include provisions for an accelerator to supply it's own clock signal to the motherboard, but as of this writing, this has not been employed by any devices.

1.17 Summary

Summary and Overview.

There is a great deal to be visualized when trying to make a comparison of system performance levels. A great many factors come into play when trying to determine just what system is best and quickest for the task at hand. Various factors can determine how efficient an accelerator is on a particular system, or how efficient a system is in general. Interface efficiency, accelerator or general system design, and intended use all play a part in determining which setup is the "winner" in the speed race. Indeed, there may not be a winner, except in a particular task category, and this must always be remembered.

No benchmark or performance test can possibly hope to test all of these categories, and the others which also play roles. Thus, it is necessary to utilize data obtained from any set of benchmarks as only a portion of the picture to be analyzed, and not as a rock-solid performance indication. System design has improved to the point where many benchmarks can be fooled into giving higher performance measures than would be found in any typical application. As benchmarks are typically small pieces of code, they must be evaluated as such. They can indeed give clues as to the performance level of a system, but certainly not a definitive answer.

1.18 AIBB Overview

Overview of Amiga Intuition Based Benchmarks

Amiga Intuition Based Benchmarks (AIBB) is a program primarily designed to test various aspects of system performance at the CPU and

accompanying device level. It does not test such things as I/O efficiency and storage media data retrieval and placement efficiency (storage I/O). The tests contained within AIBB by no means give a complete picture of any system's performance level, but does provide some basic information and comparison data for a variety of systems.

AIBB is divided into a number of sections. Several are simply informative in nature and are designed to give a better picture of the system conditions during the actual testing phases. Other portions of the program allow for a certain measure of system control, giving the ability to somewhat modify the parameters under which tests are performed. It is important to try to pay attention to the parameters and information given by AIBB, as they may in turn give important clues as to the nature of the test results reported.

AIBB is set up to allow a user to perform a number of tests on the host system, and compare those results against a series of other systems. Comparison data is given in both graphical and numerical form. AIBB also allows the entire series of tests to be performed, and the results and system state stored as a "load module" which may later be loaded and used as one of the comparison systems against which a possibly different host will be checked against. Tests may be manipulated by code type and system situation in order to allow a better picture of the system performance criteria being looked at.

1.19 System Requirements

System Requirements

AIBB may be run on any Amiga system utilizing AmigaOS 1.3 or greater, but it should be noted that the tests performed are designed primarily for accelerated systems or fast systems in general. Therefore, tests may be exceedingly long on Amigas utilizing slower CPU units, and the general speed of the program may seem a bit slow on such platforms.

Users of MC68040 based systems must be utilizing AmigaOS 2.0 or greater in order to run AIBB. Modified versions of AmigaOS 1.3 do exist which are patched to somewhat deal with the problems of that OS version and the 68040, but as per CBM's official stance, this is not a supported method of utilizing the 68040 as a system processor. For this reason, AIBB will abort if it detects a 68040 and the system OS version is less than 2.0.

AmigaOS 1.3 users with accelerators must be sure to be using the latest SetPatch routines for those OS versions. (SetPatch v1.34) SetPatch corrects a problem with FPU code with those OS versions, and is necessary for proper operation of AIBB. AmigaOS 2.0x also is shipped with a SetPatch routine which should be executed in the Startup-Sequence to assure any future OS bug fixes and corrections will be applied.

When AIBB first starts up, it performs a series of system tests to determine the type of system it is being operated on, ascertaining such things as CPU type, FPU type, MMU type, etc. Unfortunately, some low-cost accelerator units may experience a problem here...most notably in the MMU type tests.

The MMU on systems which house the unit as a seperate device (such as 68020 + 688851 systems) is treated by the CPU as an external coprocessor...much like the FPU on such systems is. The MMU or FPU in such a setup responds to an instruction when the instruction coprocessor ID field matches the hardware set ID of the device. This allows more than

one coprocessor in a system (such as both an MMU and FPU). The ID decoding mechanism is handled in hardware...and this is where the problem arises with some accelerators. Such accelerators do not fully decode the coprocessor ID, and thus the FPU may respond as an MMU, etc. Most of the time this causes no problems to the system, but it does for AIBB which is looking for these devices. Unfortunately, AIBB will most likely not work on systems afflicted with this until the hardware bug is corrected by the manufacturer. It should be noted that most systems/accelerators do NOT have this problem, but a few may show up from time to time.

This program does not absolutely have any absolute requirements other than those previously mentioned in order to be operated, but it does have some suggested configurations. In order to utilize the program's file functions, AIBB must be able to find one of the following shared libraries in the libs: directory on your system disk:

1. asl.library (AmigaOS 2.0 systems only)
2. kd_freq.library (library version 3.0 or greater)
3. req.library (library version 2.0 or greater)
4. reqtools.library

AIBB will search for these libraries in this order, and utilize the first one found. Primarily, the library need is for file requester utilizing functions within AIBB. AIBB will still operate without finding one of these libraries, but it will block access to the file-requesting functions it normally provides.

This will be the last version of AIBB to include support for AmigaOS versions below 2.0. At this time, more effort is being placed into compatibility with later AmigaOS generations, and this will be the mode of support emphasized.

1.20 Getting Started

Getting Started with AIBB

AIBB may be started from either the CLI/Shell or WorkBench. If the latter method is used, it is imperative that the icon used (if not the supplied one) have it's STACK value set to 20000. AIBB invocations from the CLI/Shell have no special requirements or stack settings as AIBB will perform the necessary set-up in this environment. It is recommended that careful attention be paid to the existing system memory resources before starting AIBB. AIBB is quite large, and if you wish it and it's test code to be loaded into a certain memory medium (generally a fast medium if possible), then enough contiguous memory must exist in that memory region. AIBB will give information as to where exactly it's code is located, but if you are interested in loading AIBB in a certain region, this must be taken into account BEFORE starting the program.

Several options are available from the command line when invoking AIBB from the CLI/Shell, or equivalently through the icon TOOLTYPES array when starting from the WorkBench. These options are listed below:

CLI/Shell Options: These options must be preceded by a dash ('-'), with no spaces between the dash and the option. The argument following the option is listed below as <arg> and should be formatted as such: -<option><arg>, such as -m0.

-c<arg>: Sets the CPU type AIBB will use for the host system. Available arguments are:

```
0  :  68000 CPU
1  :  68010 CPU
2  :  68020 CPU
3  :  68EC020 CPU
4  :  68030 CPU
5  :  68EC030 CPU
7  :  68040 CPU
8  :  68EC040 CPU
9  :  68LC040 CPU
```

Any other value will be ignored.

-f<arg>: Sets the FPU type AIBB will use for the host system. Available arguments are:

```
0  :  NO FPU
1  :  68881 FPU
2  :  68882 FPU
3  :  68040 FPU (Internal)
```

Any other value will be ignored.

-m<arg> Sets the MMU type AIBB will use for the host system. Available arguments are:

```
0  :  NO MMU
1  :  68851 MMU
4  :  68030 MMU (Internal)
7  :  68040 MMU (Internal)
```

Other values will be ignored.

-cs<arg> Sets the CPU clockspeed aibb will show/use for the host system. The argument field should be a valid clockspeed rating, such as 25.0 for a 25MHz rating.

-fs<arg> Sets the FPU clockspeed aibb will show/use for the host system. The argument field should be a valid clockspeed rating, such as 25.0 for a 25MHz rating.

-b This option accepts no arguments. Supplying it on the command line turns off the 'Click' sound AIBB makes when a gadget is pressed.

WorkBench options: These options mimic the ones given above for the CLI/Shell, with the exception that they are contained within AIBB's icon TOOLTYPES field. The options available are:

CPU=<arg>:
Sets the CPU type AIBB will use for the host system. The CPU type may be specified as:

```
                              68000
                              68010
                              68020
                              68EC020
                              68030
                              68040
                              68EC030
                              68EC040

                    For example, to specifiy a 68EC030 CPU, the option
                    to give would be CPU=68EC030.

          FPU=<arg>:
                    Sets the FPU type AIBB will use for the host system.
                    The FPU type may be specified as:

                              NONE
                              68881
                              68882
                              68040

                    For example, to specifiy no FPU, the option to give
                    would be FPU=NONE.

          MMU=<arg>:
                    Sets the MMU type AIBB will use for the host system.
                    The MMU type may be specified as:

                              NONE
                              68851
                              68030
                              68040

                    For example, to specifiy no MMU, the option to give
                    would be MMU=NONE.

          CPUSPEED=<arg>:
                    Sets the CPU clockspeed aibb will show/use for the
                    host system.  The argument field should be a valid
                    clockspeed rating, such as 25.0 for a 25MHz rating.
                    For example: CPUSPEED=16.0 would set a CPU speed of
                    16.0MHz which AIBB will then use internally.

          FPUSPEED=<arg>:
                    Sets the CPU clockspeed aibb will show/use for the
                    host system.  The argument field should be a valid
                    clockspeed rating, such as 25.0 for a 25MHz rating.
                    For example: CPUSPEED=16.0 would set a CPU speed of
                    16.0MHz which AIBB will then use internally.

          NOBUTTONBEEP:
                    Using this tooltype option turns off the click sound
                    AIBB uses when a gadget is depressed.

IMPORTANT:
    The CPU/FPU/MMU options given above are for special circumstances only!
Normally, AIBB will determine all of the above independently, and tampering
```

with these values will be detrimental. However, these options can come in very handy under certain circumstances.

Some accelerator models on the market suffer from a hardware bug: They do not properly decode the coprocessor ID in hardware for systems with such devices. The end result is attempted accesses to an MMU may end up with the FPU on the system erroneously responding instead. Now, since AIBB relies on an 'exception' occuring when no MMU exists in its efforts to ID the system MMU, this becomes a problem if the FPU responds instead. The result of this is that AIBB may fail to function properly on such systems, and this is where the above options come in.

When the options above are specified, AIBB will take them at face value. No further testing of the system is attempted. Therefore, by specifying various values, the problem above can be circumvented as AIBB will not perform the internal checks which may cause errors. If you suspect your system is one with such a hardware bug, try manually setting the system CPU, FPU, and MMU types to see if this cures the problem. You should not have to set the device clockspeed ratings manually, as AIBB will still be able to perform this.

ONCE AGAIN, do not take the CPU/FPU/MMU command options lightly! If false values are given, it may very well result in program errors within AIBB, or possibly a system failure. Under most circumstances, you will NOT need to use these options AT ALL, and can allow AIBB itself to determine the system configuration.

Under some circumstances, AIBB may request that the processor type be supplied manually by the user. This is primarily in situation where AIBB can't positively determine whether a 68EC030 or 68030 exists, or in the case of 68LC040/68EC040 determination. If AIBB requests this information, please supply the correct processor type, as failing to do so can result in serious problems on occasion. This is especially true in the case of the 68EC030 vs. the standard 68030. AIBB may not be able to determine the exact processor in this case if for some reason the MMU enabled bit is set in the processor's Translation Control (TC) register. Both the 68EC030 and 68030 have valid TC registers, even if with the EC part the MMU is non-functional. Since AIBB attempts to parse MMU tables if the MMU is active (for locating system structures), fooling AIBB into thinking that an EC part is a standard 68030 in the case of a seemingly active MMU can result in AIBB attempting to parse a non-existant MMU table. This can be very problematic, and in extreme cases result in a system failure.

Once AIBB loads, a few moments may be needed by the program while it evaluates the system it is being operated on, the exact time depending on the relative speed of the host system in question. A screen displaying a message of that sort will be given while this is in progress. Following this evaluation, you will be presented with AIBB's main program screen .

1.21 Main Screen

Main Screen Description

AIBB's primary screen consists of several informational areas designed to provide information about test operations and basic system information. These areas are divided into the following areas:

Performance Graph
Test Result/Information
Base Machine Indication
Comparison Information
Basic System Information
Test Activation Gadgets

AIBB also has a series of Main Screen Menus which provide further control of its testing environment.

1.22 perfgraph

Performance Graph

The performance graph is a bar graph display of the comparisons made after each test is performed. Ratings are given in reference to the base machine for comparisons, with the highest performing system having it's bar displayed in a seperate color from the other systems. Note that although numerically two machines may have the same results out to 2 decimal places, AIBB may still show one as the performance leader. This is due to rounding, and the fact that the one highlighted machine does in fact have a higher rating if a few more decimal places were shown numerically. However, such small quantities should not be taken literally, as far too many variables exist to use such values in accurate comparisons.

1.23 testinfo

Test Result/Information

This area provides several pieces of data. First, it gives the name of the test last whose information is being displayed currently. The numerical result of the test performed is given here, as well as the memory node reference number where the test code and any test data is located. To reference these node numbers, please see the section on the System Information Display .

1.24 basemachind

Base Machine Indication

Below the Test Result/Information area is a small reference

which lists the current comparison system being utilized as the base for all comparisons performed.

1.25 compinfo

Comparison Information

This section provides several key pieces of information about test performance. It gives the numerical ratings of all systems utilizing the base machine as a reference. These values are the same as those used to generate the performance graph.

The system headers here which label the machine in each row are in fact gadgets that when pressed will move AIBB to its
System Information Display , showing data on the system selected.

In addition, this area houses the test code type gadgets/indicators. Selection of code options for the host system causes AIBB to perform any tests utilizing those options. Selections under the comparison systems result in AIBB using the figures for that code type (previously obtained when the comparison data was generated) when making comparisons. Note that not all options will be available, depending on system capabilities.

The gadgets allow for seperate selection of CPU and floating point code models. Floating point code selections will only have effect on tests which use such operations, while the CPU code model will be in effect across all tests. Thus, when performing a non-floating-point test, the current floating point code model selection is ignored.

The gadgets are cyclic in nature; repeated selection will move them through all available code models. The currently available CPU code types are:

Standard 68000 Code
: Having this item selected sets the code type to that which is compatible with all MC680x0 series microprocessors. Note that this means no advantage is taken of the capabilities or code optimizations available on later-generation microprocessors of this series, but it is a good base selection as it can be utilized on all existing Amiga systems.

68020+ Code
: This item selects code compatible with later generation MC680x0 series processors. It will not be compatible under most circumstances with earlier (MC68000 or MC68010) based systems, but will take advantage of some of the more advanced capabilities of these later processors in the series.

The currently available floating point code options are given below. As indicated earlier, they will affect only tests which utilize floating-point math in nature.

Standard Math Code
: Using this option sets the code type to use software emulation of floating point routines. This is compatible with all Amiga systems in use, as it is not hardware specific.

In-Line Coprocessor Code
: This option sets the test code type to that which uses

faster in line FPU instructions for floating point operations. As not all systems will have a coprocessor available, this option is not universally available on all systems.

68040 Enhanced Math Code

For use with 68040-based systems, this option allows the use of FPU code which is more optimized for 68040 processors. Such processors do not have hardware-assisted transcendental functions and this option will set up for in-line emulation of such, alleviating the need for trap-based libraries such as 68040.library or similar vendor supplied code.

1.26 basicinfo

Basic Information

Located just below the performance graph, this area provides key pieces of information about the current state of the host system. The system CPU type, FPU type, and MMU type in use are displayed, as well as the current operational status of the MMU. Also displayed are the approximate CPU and FPU clock speed ratings, as calculated when AIBB first evaluated the host system on startup.

This area also contains the system cache status indicators/gadgets. These show the current state of any CPU caches which may exist, and also allow their condition to be changed by selecting the cache parameter desired. Clicking on a particular parameter toggles it through both its "ON" and "OFF" states.

A lot of confusion tends to exist about the CPU cache modes, and the MC680x0 cache BURST mode (supported on the MC68030 and MC68040) is often not understood. BURST mode operations are a special form of cache filling (updating the contents of the cache) where an entire "line" of cache data may be filled sequentially and faster than the single-entry mode of cache filling. A cache "line" in this case is a series of 4 longwords (32 bits each) arranged simplistically as:

```
entry:   1    2    3    4

line 1  ---- ---- ---- ----
line 2  ---- ---- ---- ----
                ...
```

where each entry is one longword. The MC68020 and MC68030 utilize cache sizes of 16 lines, giving 256 bytes of cache storage. The MC68040 increases this to give a total of 4K of cache space for each of the data and instruction caches.

BURST mode is essentially a compromise in performance. Average-case CPU performance is enhanced at the cost of worst-case performance. The latter effect is true because during BURST mode operations the CPU bus controller is committed to a memory fetch sequence for a longer period of time than with single-entry mode.

The mode enhances average and best case performance by allowing the CPU to sequentially fetch 3 additional longwords from memory faster than normally done by the usual asynchronous single-fetch bus cycle. Once it has fetched the first longword, the next 3 are clocked into the cache line utilizing only 2 clocks per fetch, thus filling one cache 'line' in 9

clocks (assuming a zero-wait state initial fetch) rather than 15 clocks. The theory behind this is that the data/operands sequentially surrounding the initial fetch will most likely be needed soon in any case, and placing them in the cache leads to their eventual faster access.

BURST mode operations are not universally applicable to all systems however. Generally, the memory controller on the system (or particular memory board) must be capable of supporting BURST mode operations, or the BURST request by the CPU will not be fulfilled. In systems not capable of these modes, activating them will not be detrimental, but will go unnoticed in performance terms.

The CPU will request BURST fills when it deems appropriate, but the memory controller will not acknowledge the request and thus simply force the CPU to do single-entry fetches as in standard operation.

1.27 testgads

Test Activation Gadgets

These are located in the lower right-hand corner of the screen and serve several purposes. Normally, they are utilized to start a test, but this is dependent upon the mode of operation AIBB is currently in. See the section on "Review Mode" for further information of this nature.

Activation of a gadget in standard mode starts a test with the current code parameters and general settings, as detailed in the appropriate sections later. Tests are divided into two groups: "Standard" and Floating-Point. Standard test types, denoted with WHITE lettering, are more general to the system, and represent code more often found in operational situations. Floating-Point tests, given YELLOW lettering, utilize a great deal of floating-point math to test the system's performance across that domain. See the Test Descriptions for more detailed information on the tests available within AIBB.

1.28 mainmenus

Main Screen Menus

AIBB's primary screen has attached to it a number of menu items which give even more options and control over program operation than the gadgetry supplied. There are 4 menus each controlling a different aspect of AIBB:

Menu 1: General
Menu 2: System
Menu 3: Test Options
Menu 4: Special

1.29 menu1

Menu 1: General

This menu provides general control of and information about AIBB. The

subitems included here are:

About AIBB
This option presents a requester giving credits and information about this version of AIBB.

Load Module Prefs
AIBB allows the use of alternate systems other than those contained internally in order to make comparisons against the host system. This menu item will bring up a requester-like arrangement which will allow the paths to load modules to be used in place of the internal defaults to be specified. To replace an internal module at startup for comparisons, simply enter the full path name to the alternate load module in the respective entry in this requester. Leaving an entry blank informs AIBB to use it's internal default for that system. Note that this configuration will take effect when AIBB is next started, and the the next menu item, "Save Configuration" as detailed below, must be selected to save the choices made here.

Color Settings
The colors AIBB uses for its main screen displays are user selectable, and may be changed if personal taste desires. This menu option will bring up a color requester which will allow AIBB's palette to be modified to suit. This may be particularly useful for users of monochrome monitors which can only display levels of grey, rather than color. Under such circumstances some of AIBB's normal colors may map to grey shades so similar as to be indistinguishable on the screen. Use of this option can correct such a situation.
Use of the "Save Configuration" menu item will save the color palette chosen with this option to file, and AIBB will use that palette in subsequent invocations.

Save Configuration
This saves the current state of AIBB's menu item selections, as well as the current order of the comparison machines as they are placed. For more information on these regards, see the section on loading new comparison modules from the default systems within AIBB. AIBB currently saves this data to a file called "aibb.prefs", which may be located in an assigned directory called AIBB:, or your system S: directory. This file will be searched for, in that order, when AIBB is first invoked, and the values contained within will set AIBB's startup options. If AIBB cannot locate a preferences configuration file, it will notify you and use internal default values.

QUIT
This item forces termination of AIBB.

1.30 menu2

Menu 2: System

The system menu allows for control over AIBB's interaction with the rest of the system while it is running. The one item currently residing here is given below:

AIBB Task Priority

A submenu-endowed item, this selection allows for the changing of AIBB's task priority. This is primarily for running tests while still allowing multitasking to occur, while examining the effects of different task priority levels. For information on disabling multitasking during test operations, see the "Disable Multitasking" entry under the Test Options menu descriptions.

1.31 menu3

Menu 3: Test Options

This menu controls various settings available with AIBB's tests. The items listed here are:

Disable Multitasking

When this item is selected, it indicates AIBB should perform all tests in such a way as to disable all system multitasking during the run of any test. This allows a figure to be generated which indicates the system performance FOR THAT TEST more accurately, as there is no task context switching during the test runs. Note that all comparison system figures are generated with this option enabled, so this should be selected in order to compare the systems on an even par. When this item is utilized, the previously mentioned ability to set AIBB's task priority will have no impact on test performance, as no task switching will occur, and thus the task priority level becomes meaningless.

It should be noted that when using this option, it is a good idea NOT to be running much in the background. The Amiga's operating system is a near-real-time setup, requiring in many cases fast response to system conditions. Use of this option can affect certain other operations adversely, most notably that of serial communications and the like.

Screen Overlay

Using this option results in AIBB putting a one bitplane (two color) low-resolution screen over it's main screen during every test. AIBB's normal screen is a high-resolution 4 bitplane (16 color) screen, and on CHIP RAM only systems, and for some tests even on FAST RAM equipped systems this may result in a great deal of bus contention on the CHIP RAM bus. Subsequently, performance levels may be adversely affected for the test. The use of this option attempts to alleviate some of this problem by utilizing a screen overlay which minimizes bus contention on the CHIP RAM bus by limiting the required DMA activity by the custom chips to display it

while it is the topmost screen. Again, all comparison data for the other systems is obtained with this option enabled, so in order to keep comparisons on par this option should be enabled, which it is by default values.

Note that for graphics-related tests this option will not be activated as it would be detrimental to what those tests are indeed trying to analyze. It is advised that if this option is enabled while multitasking is permitted that screens not be shuffled while a test is in progress. The uppermost screen is the cause of the CHIP RAM bus display DMA effects, and to shuffle to another screen during a test could nullify the advantage of using this option.

Set Gfx Test Display Mode

AIBB allows all graphics tests to be run on any system supported display mode, and this option allows the user to select the display resolution and depth (number of colors) to use when running such tests. Selection of this menu item brings up a screen mode requester via the asl.library requester functions. As versions of asl.library which support the screen mode requester are required for this to function, the host system must be running AmigaOS 2.1 or greater.

Once a particular screen mode is selected, any graphics tests run will be done in that mode. This is particularly useful for comparing the effects of differing resolutions and display depths on graphics performance levels. One must be careful to take note of the modes used for the other systems as well, else improper conclusions as to how well a system does in these tests could be drawn. For this reason, AIBB will post a warning if the screen mode of either the host system, or a comparison system does not match the modes in use on the other machines. If simple, fair and straightforward checks are desired, all systems should be compared using the same screen mode.

View Comparison System Gfx Modes

As AIBB does allow differing screen modes to be used for graphics tests, through this function it also allows browsing though the various modes in use on the host/comparison systems. Selection of this item brings up an interactive requester which allows movement through various systems, and comparison of the various display parameters in each.

Set Comparison Base

This item contains the names of the comparison systems in a submenu area. Selecting one of these submenu items sets the current comparison base system to that machine. The comparison base is the system utilized as the 'base' value for test results when computing performance ratings. All percentages shown are given as percentages of the base system, with a 1.0 value for a system indicating a performance equal to the base system.

1.32 menu4

Menu 4: Special

This menu has a number of special functions AIBB includes for comparison convienience among different machines. These are:

Enter/Exit Review Mode

Entering Review Mode gives a method for reviewing previously performed tests and their comparisons. When this mode is active, selecting a test gadget, or setting a comparison option (code type, etc), will result in the display of the results last obtained for that test. If no test results for the host system are available, the information for the comparison systems currently in use will be shown, and the host system will data will be marked with a "N/A" indicating the information is not available. The ability to display the comparison system data without running the actual test on the host system is provided to allow a quick view of the performance of said comparison machines before running the test(s) on the host.

Code type options may be manipulated here, and if a test result is available for those settings, it will be displayed. For example, if you were to have the Matrix test as the current test you are viewing, and you want to see the results of the test under 68020+ code, selecting that item under the "This Machine" code type selection will show the Matrix test results utilizing this code type (if they were previously performed, making the data available).

Start/Stop Log File

AIBB has the ability to keep a "log file" of test activities. This option allows you to start this logging operation, or stop it once in progress. The log files contain basic information, in text form, about each test as it is performed, as well as essential system information.

Starting a log file involves selecting a file name to which AIBB will save this data. If the file is an existing one, AIBB will check for the words "AIBBLogFile" at the start of the file. If this is not found, you will be warned and given the option of aborting the use of this file as a log file. Heed this...AIBB WILL write into any file if told it is acceptable, including executable load files. This checking is done in order to prevent accidental file damage or destruction.

All Tests | Make Module

This is a rather important option. As indicated earlier, AIBB has the ability to create a "load module" of comparison results in order to utilize them later in other runs as a comparison system. This selection allows the generation of just such a load module. Selecting this menu item will result in a requester being displayed which warns that this option may take considerable time, and that multitasking will not be functional during it’s operation. At this point, the operation may be cancelled if it is not desired at that time.

When performing all the tests, the options "Disable Multitasking", and "Screen Overlay" previously mentioned are

automatically enabled in order to give consistancy to all such generated modules which may be utilized in AIBB. Using this option, all tests are performed in all possible code combinations available on the host system configuration, in order that later comparisons will have as much data to go by as possible.

Upon completion of all the tests, a requester will be displayed informing you if the tests completed successfully, and asking if you wish to create such a load module at that time. If you choose to do so, a file requester will appear asking for the name of the file to save the module under. Following this, a smaller requester will appear asking for the name to use with the module under the graph display for it. This defaults to the first 8 characters of the filename, but may be changed as desired. Note that only names of up to 8 characters are supported at this time.

If "Cancel" is selected in reference to the module creation requester, AIBB will go back to it's normal operations, and other tests may be performed. In this manner, it is possible to use this option simply to perform all possible test combinations for later review. If you wish to review the tests done before making a module, this is possible by not saving the module at the time, and entering "Review Mode" upon finishing. If no further tests are performed (which would invalidate the consistancy of the module's data), then selecting "All Tests | Make Module" again after reviewing the data will result in a requester informing you that the data for a module is still valid and will ask you if you wish to create one now.

It should be noted that comparison options and settings are not in effect during the performance of the tests with this option. AIBB will merely do all tests with all code types possible, and keep the results (if desired). Comparison options are only effective (and necessary) when viewing the information present, and are not important when generating a load module.

Once all module options are completed, AIBB will present an analysis of the overall system performance with respect to the various comparison modules currently in use. This analysis consists of averages in Integer, Graphics, and Floating-Point performance when put against each comparison machine in turn. This average gives somewhat of an "all around" look at the host system's performance levels.

Show Aggregate Results

Once a load module has been performed on the host system, this item becomes available for selection. When activated, a requester displaying combined totals for the host system in terms of Grapics, Integer, and Floating Point performance will be shown. These totals are given as figures against the currently loaded comparison systems. Additional tests may be run after the original load module creation to see any effects may take place in different configurations (cache, etc...). Rerunning tests under the same situations as the module run uses will most likely not affect these figures significantly.

1.33 sysinfodisp

System Information Display

AIBB's System Information Display is a seperate display which is brought up when the Main Display gadgets for individual systems are selected. This display gives various information about the state of the system selected, and is also the location from which other load modules to enter as comparison systems may be selected.

The display here is broken into several sections, giving modular information areas pertaining to various system data. If the host system is the system being viewed, the data represents the current state of the host system. If a comparison system's information is being viewed, then the data is representative of the system state when that machine's module was created for further comparisons.

The upper portion of the display consists primarily of CPU/FPU/MMU data and state information which is fairly self-explanatory. Other information given in this section includes the display type in use, Agnus and Denise custom chip revisions of the system, and several items of particular interest:

System Stack Memory Location

The system stack (or "Supervisor Stack") is the memory region reserved for use by the processor while operating in what is known in M680x0 terms as "Supervisor Mode". Supervisor mode is the CPU mode of operation most often associated with operating system use, and various system maintenance operations. Supervisor mode is characterized primarily by the fact that it allows unhindered access to certain CPU operations which are of primary interest only to system-level operating system functions. User Mode is the operational status in which almost all applications function, and said CPU operations are considered "off limits" in this mode. This is to protect the integrity of the system from runaway programs and the like, and to more easily facilitate multiprocessor/multiuser system environments. It is a characteristic of the M68000 microprocessor series and serves to allow a seperation between operating system priviledges and user program priviledges.

The system stack is where much CPU state information is stored during operating system activities, and thus it is important to recognize it's location in memory. Depending on the memory type where this stack is located, it may affect certain operation speeds, and it's location is thus given here to allow this to be taken into account when evaluating system performance. It should be noted that although this is an important item of interest, it is generally not going to have much effect on the greater majority of AIBB's operational modes and testing.

AIBB Process Stack Memory Location

This item is probably of more interest than the System Stack location. AIBB's process stack is a memory region which is assigned to AIBB (and any user program) when it is invoked. Certain program variables and data are stored on the stack during operations, and thus it's location can affect performance levels. This should be taken into account carefully, as some of the testing AIBB does utilizes this stack for data, and thus results will be affected if it is located in a slower memory medium than optimal

for the system configuration.

Operating System Version
This field identifies the operating system version in use on the system in question. Certain versions may have different features, and may affect certain of the test performance levels.

Operating System Location
On certain MMU equipped accelerated systems, or on such system with special hardware setups, the operating system ROM image may be relocated to a faster memory medium. ROM access times are generally slower than that of RAM resources, and in the case of an A500 or A2000 with an accelerator which is more at home with a 32-bit data bus than those systems' normal 16-bit 7.15 MHz bus, it is extremely advantageous to move the operating system kernel code to such a faster accessed memory region. Often times, this relocation is done by using a system's MMU (Memory Management Unit), which allows for address translation of memory "pages".
Translation occurs by mapping a certain memory region such that accesses to it are rediverted to an alternate location in this kind of setup. Programs such as Dave Haynie's SetCPU and the CPU program which comes with AmigaOS 2.0 and above allow this type of operation. AIBB is capable of determining the actual memory location of the ROM code image by checking through the MMU translation tables, and will report where the code resides.
Some accelerators allow for translation of the ROM image without utilizing an MMU. Such units utilize a custom hardware arrangement, and at this time AIBB cannot accurately determine the memory location of the ROM image for these systems. In these cases, it is recommended that such translations be noted for further reference if comparisons are to be made against other systems utilizing a module or log file results so that no confusion about the system setup occurs.

The lower portion of the System Information Display contains provisions for examining system memory node, expansion board, or pertinent sytem library information. Three gadgets to the right of this area provide the means to select the desired display. The list of nodes or boards can be moved through using the 'Next' and 'Previous' gadgets located below the selection gadgets, while the library information is static. The information given for memory nodes is:

Memory Node Index
This is an index value corresponding to which node is currently being viewed, and how many total nodes exist. This value directly relates to the main screen's "Code Loc" and "Data Loc" test information values and can be used to determine where AIBB's test code and data is located.

Memory Node Name
This is simply the name of the given memory node.

Memory Node Address Range
The address range for the current memory node is displayed here in a hexadecimal form. Both the starting address, and ending address are given.

Memory Node Total Size
The total usable memory within the given node is displayed here.

Memory Node Priority
Memory on the Amiga is prioritized for allocation. This means that memory of a higher priority is given precendence over other memory regions when an allocation request is attempted. For example, a memory region of priority 5 will be scanned first for a suitable memory chunk for a given allocation request before attempting other regions. If there is not enough memory in this region, the next priority region is tried, and so on. The main item of note otherwise is that this is true for GENERAL memory requests. Memory requests which specifically ask for CHIP memory will have the allocation attempted there, regardless of priority.

Memory Node Bus Port Width
This is the bus width of a given memory region. A 16 bit bus corresponds to a data path width of 16 bits, 32 meaning a 32 bit data path width, etc. For 68020+ systems, memory port widths of 32 bits will have the advantage over 16 bit ports for efficiency reasons, as the 68020 and above have 32 bit data paths, whereas the 68000/68010 have 16 bit data paths.

Memory Node Type
Whether the given node is FAST memory or CHIP memory is displayed here.

Custom Chip Bandwidth
This will only be seen when examining a CHIP memory node, and only under AmigaOS 3.0 or greater, and indicates the bandwidth specified for CHIP memory on the system. Note that at present the only differences here will be seen between AGA chipset equipped systems and non-AGA equipped machines.

CPU/Memory Access Latency Index
This figure represents the latency between a memory cycle, and when another cycle can be performed. Lower ratings indicate better response times for a particular memory node, with the unattainable goal of 0.0 indicating that no latency occured at all. Basically, this gives information as to the relative efficiency of various memory nodes (eg, one with a rating of 5.0 would be more efficient, and hence faster than one with a rating of 7.0.). Note that this can only be used as a valid comparison across different systems if other factors such as processor type, clockspeed, and bus width are also taken into account. This figure is most useful in comparing two different memory regions on similar systems, such as two memory boards on a 68030 based system against each other for relative efficiency. Note that this figure will only be given for FAST RAM memory regions.

When Expansion Board information is selected, information about the system AutoConfig® boards will be shown. The given fields will be as follows:

Board Index
The index value for this board, and the total number of expansion boards for which information is available is shown here.

Board Address
This is the configuration address of the given board. For memory boards, this will generally reflect the starting address of the memory region it occupies.

Board Size
The total byte size reqirements for the board will be displayed here. This shows the amount of memory this board will take up when configured onto the system. Note that with memory boards, this will generally reflect the size of the memory available on the expansion board.

Board Manufacturer ID
Commodore-Amiga assigns all valid AutoConfig® board manufacturers a unique ID code. This field contains the ID of the manufacturer of the given board being shown.

Board Product ID
Manufacturers have the option of assigning a product ID to their boards. This shows the product ID given to a particular expansion board.

Board Type
At this time, this field will simply designate whether the given board is a memory board, or some other type of peripheral expansion.

Board Attributes
This field basically gives information as to whether the expansion device is configured as a valid Zorro-II or Zorro-III setup.

Ident
AIBB contains a number of expansion board identifications internally, and will attempt to match the board found with one of these in the lists. If no match is found, the statement "No Information Available" will be given to indicate this. If you see this message, and wish the board in question to be listed in AIBB's lookup tables, please let me know by way of providing me with the expansion board Product ID, Manufacturer ID, and the identity of the device.

When Library node information is selected, information about pertinent system libraries will be shown. Note that not all system libraries currently in use are displayed. Only selected ones which are of interest when determining performance factors are recorded. Currently these are:

exec.library
graphics.library
intuition.library
layers.library
expansion.library

The information given is of the following form:

Library Name
This is simply the name of the library in question.

Library Version
This field gives the version and revision of the system library being displayed. This may be important when looking at performance statistics of tests which make use of system kernel calls (such as graphics tests).

Library Base:
This is the base address of the library, and indicates where in memory it is located. Again, this may be of interest when examining the performance of tests which make use of system kernel calls.

The System Information Display also includes a number of menu options which are explained below:

Select Other:
A submenu attached to this item allows you to switch to viewing another system's attributes from within this display.

Load New:
This is the option to utilize if you wish to load a comparison module in place of the ones alread in use. The loaded module will replace the currently displayed system's location in the comparison systems. This option is not available when viewing the host system's data. Subitems attached to this menu item allow you to select the type of module to load. These are:

From File:
This should be selected if you wish to load a previously saved module in file form. A requester will be displayed asking for the file name to load. AIBB will attempt to load the module, and if all data consistancy checks are valid, it will place this data in the location of the previously displayed system.

Under this option is a list of the internal default modules AIBB contains. This allows the rearranging of the order of the default systems as they appear on the graph in the Main Display, and also allows a default system's values to be re-loaded if one is superseded by a file-based module at an earlier time. Note that the order of the system default modules is one of the items saved in the AIBB.prefs file, so you may choose any ordering of the internal startup default systems which suits you best.

Return to Main:
Returns you to the Main Display portion of AIBB.

1.34 testoverview

Overview of AIBB's Included Tests

The tests AIBB incorporates are described below. The type of test, and it's basic operations are given in the descriptions, as well as the amount of memory each test may need to allocate external to AIBB itself. The "standard" tests are as follows:

A. WritePixel

The WritePixel benchmark will open a screen/window combination and fill it completely with a given color pattern. The work is done one pixel at a time, utilizing the operating system routines SetAPen() (sets the current RastPort primary pen color) and WritePixel() (which sets a pixel to the current primary pen color).

The test is basically a benchmark of the time needed to call these routines, and for them to execute. For the most part, this it will be primarily useful for evaluating the effective ROM image access time for systems which differ from the conventional ROM access method found on the Amiga 500/600 and 2000, namely accessing the ROM over those systems' normal 16 bit bus. As these routines also result in many accesses to the CHIP RAM bus, it can also give a hint as to the efficiency of a system's CHIP RAM bus interface.

WritePixel reports its results in pixels per second drawn. Please note that this is NOT the maximum pixel rate of any particular system, as there are more efficient methods of doing this kind of work. This is the effective pixel rate of the system when the methods and routines used by this test are employed.

Memory Usage: No direct memory resources external to AIBB are allocated. CHIP memory is utilized for the screen and window.

B. Dhrystone

This test should be fairly familiar to most people, as it has been utilized on many different system for benchmarking purposes. It is a test which attempts to put conditions upon the system which more closely simulates a possible applications program section. It returns, not run-time in seconds, but rather a rating of Dhrystones per second, where in this case, the larger number indicates better performance.

Memory Usage: No memory resources external to AIBB are allocated.

C. Matrix

A matrix manipulation benchmark utilizing 3 50x50 integer matrices. The test simply performs a series of matrix operations (addition/subtraction, multiplication, transposition, etc) upon these matrices. The test is set up in such a way that a great amount of time is spent moving data, as well as performing arithmetic operations upon it. Therefore, this could be thought of as also testing memory manipulation efficiency. The test is an indicator of how well a processor/memory combination handles memory accesses to data and operations on such, as the test does not allow the processor to simply perform the data operations solely within it's registers.

Memory Usage: 30,000 (29.3K) bytes external to AIBB are allocated.

D. MemTest

This test is memory-bound, as its name implies. In essence, it is a memory block movement test, timing the efficiency of memory accesses and transfers using longword (32 bit) sizes. It should be noted that the Data Loc portion of the test result information will supply the node location of the RAM being tested. Systems with FAST RAM will show higher results, as the test will execute quicker, and as can be expected, 32-bit ported FAST RAM will perform better than its 16-bit ported counterpart. Note that this test will use FAST RAM as a memory medium if available, and will report its results in megabytes transferred per second.

Memory Usage: 32,768 (32K) bytes external to AIBB are used.

E. Sieve

Another test which should be familiar to most, the Sieve of Erathosthenes. It uses a fairly simple algorithm to determine prime numbers within a range of numbers. This test simply times your system when implementing this algorithm, which is decribed fully in many textbooks, or one can simply look at BYTE Magazine's benchmarks, which use a similar Sieve test.

Memory Usage: No memory resources external to AIBB are allocated.

F. Sort

A series of 30,000 16-bit integers is sorted from a pseudo-random setup, and the procedure is timed. "Pseudo-random" meaning that the number arrangement is not created in a random fashion, but rather in a mixed fashion so that on each invocation of the test the numbers will be created in the SAME mixed fashion. This is because the sorting algorithm is sensitive to the mixing, and if each time the test was run a different group of values was used, no two tests results could be compared well. The mixing method I used was to insure that the algorithm would be forced to do the most work for each test.

Memory Usage: 60,000 (58.6K) bytes external to AIBB are allocated.

G. IMath

Integer Math. This test performs a wide variety of integer math functions. Included among these operations are the standard functions, such as addition, subtraction, multiplication, division, and a few additional bitwise functions, such as ANDing, ORing, and XORing.

Memory Usage: No memory resources external to AIBB are allocated.

H. TGTest

Text/Graphics test. This test is another one which is dependent upon the efficiency of the system graphics routines' execution speed, as well as the efficiency of the CHIP RAM bus interface on the system.

Memory Usage: No direct memory resources external to AIBB are allocated. CHIP RAM is used indirectly for the screen/window creation.

I. EmuTest

This test is basically a small CPU emulator core running an instruction set simulation (basically a small program). The Amiga seems to have gained a bit of a precedence in CPU emulation, and this test was developed for the purpose of showing various systems' ability to perform such emulation efficiently and speedily. The simulated CPU is a standard 68000, though the results from this can be taken as indicative of other CPU emulators as the basic principle is the same. All instructions and internal operations are completely software emulated. The results for this test are given in Simulated MegaHertz, basically a rating showing how fast the emulation is towards an equivalent hardware-based CPU.

Memory Usage: No memory resources external to AIBB are allocated.

J. InstTest

This test is not affected by the code settings given for any system. It performs a series of the most common CPU instructions in a 6K loop, and times their execution. It then does a percentage average of the instruction makeup, and gives a result in Instructions per Second. THIS IS NOT A STANDARD "MIPS" TEST! Most tests using the "MIPS" scale are very simplistic and for the most part are not very useful whatsoever. A standard "MIPS" scale test will most likely give you numbers much larger than AIBB will. AIBB attempts to make an even spread of 680x0 instruction execution, thus showing a somewhat more even look at things. This test is basically to determine the raw speed of code execution on any given system.

Memory Usage: No memory resources external to AIBB are allocated.

K. EllipseTest

This is a test of an applied graphics operation. The test draws a series of filled, anti-aliased ellipses and times the operation. Anti-aliasing is the technique of "blending" line curves so as to soften their sharper edges.

Memory Usage: No direct memory resources external to AIBB are allocated. CHIP memory is utilized for the screen and window.

L. LineTest

A test of line-drawing primitives. LineTest opens a screen/window combination and draws a series of lines throughout them. The lines are drawn in horizontal, vertical, and diagonal fashion, with emphasis being on the former two. This test reports its results in terms of lines drawn per second.

Memory Usage: No direct memory resources external to AIBB are allocated. CHIP memory is utilized for the screen and window.

The floating-point specific tests implemented by AIBB are given below. Note that these tests are also dependent on any standard code type selections which may be made, as well as the type of floating-point code utilized.

Tests are marked as to their usage of transcendental functions

(sin(), cos(), log(), etc...) for record keeping and comparisons by 68040 users, who should see the appropriate notes in this documentation concerning the built-in 68040 FPU and transcendental functions. The rating scale used below for such usage corresponds to this table:

Level	Meaning
NONE	No transcendental functions are used
LIGHT	5-20% of calculations are transcendental in nature.
MODERATE	21-50% of calculations are transcendental in nature.
HEAVY	Greater than 50% of calculations are transcendental.

M. FMath

Floating Point Math. Similar to the IMath test, with the exeception that Floating Point values and operations are utilized. With this test, no bitwise operations are performed. Single precision floating point operations/values are used here.

Transcendental Usage: NONE.
Memory Usage: No memory resources external to AIBB are allocated.

N. Savage

This is another of the "probably familiar" tests. It is a standard implementation of the Savage test, which makes nested calls to transcendental functions to create a single value. Double precision floating point operations/values are used.

Transcendental Usage: HEAVY; this test is almost exclusively transcendental in nature.
Memory Usage: No memory resources external to AIBB are allocated.

O. FMatrix

The FMatrix test is similar in concept to the Integer Matrix test outlined above. Again, a great deal of data movement is performed, in addition to the operations involved, which are floating point operations in this case. With the matrix operations, the results under Floating Point coprocessor equipped systems can be interesting to note, as the system is not able to keep the data within fast-access FPU registers, and thus must make frequent bus accesses for the data it needs. Double-precision floating point math is used for this test.

Transcendental Usage: NONE.
Memory Usage: 38,400 (37.5K) bytes external to AIBB are allocated.

P. Flops

A common rating of floating-point operations, the term 'Flops' denotes Floating point operations per second. This test takes a composite of operations and reports its results in terms of scalar MFlops, where 1 MFlop is one million floating point operations per second.

Transcendental Usage: NONE.
Memory Usage: No memory resources external to AIBB are allocated.

Q. TranTest

This is a test which is solely transcendental in nature. A series of transcendental functions are performed in a large loop, and timed for speed of operation. This test will tend to show the relative efficiency of a system in performing more complex mathematical functions.

Transcendental Usage: HEAVY (Completely transcendental).
Memory Usage: No memory resources external to AIBB are allocated.

R. BeachBall:

The BeachBall test was originally written by Bruce Holloway of Weitek, and published in the March 1988 issue of Byte Magazine. It is essentially a very math-intensive operation which draws a beachball on the screen, complete with shading. The test opens a 640x400 interlaced 16-color screen, and proceeds to render the picture. This test is closer to a true "application" test, in that it actually does something visible, and produces an output. The system will end up being tested in both the floating point arena, and in CHIP RAM access performance, which is done through standard operating system graphics handling calls (thus will be affected by the speed of such, which in turn can be affected by ROM image re-mapping, etc).

Transcendental Usage: LIGHT.
Memory Usage: No direct memory resources external to AIBB are allocated. CHIP RAM is used indirectly for the screen creation.

S. FTrace:

Another applications-type test. FTrace implements a subset of the calculating functions which are used to perform ray-tracing operations. Ray-tracing is a particularly floating-point intensive art, and this test gives some indication of a system’s performance in this type of operation. No visible result is produced, so in that matter it is not an ’ideal’ test, but it can be used to give some indications in this arena.

Transcendental Usage: LIGHT; Calculations are performed in such a way that transcendental usage is minimized.
Memory Usage: No memory resources external to AIBB are allocated.

T. CplxTest:

This test implements a series of complex-number operations and times their execution. Complex number applications are important in many of the sciences, and are particularly prevalent in such areas as electrical engineering (circuit analysis) and vector analysis to some degree (not specifically "complex numbers" in that case, but the operations are similar). This test utilizes a lot of quick, small memory moves, as well as performing a variety of floating-point operations.

Transcendental Usage: LIGHT TO MODERATE.
Memory Usage: No memory resources external to AIBB are allocated.

1.35 compsystems

Included Comparison Systems

AIBB's internal default comparison systems were selected to give a broad overview of a number of system configurations and hardware types. They represent the broadest base of default type systems (eg, not third-party enhanced). These systems are as shown below:

A600-NF
: An Amiga 600 system with no FAST RAM (NF) complement. This is an all CHIP RAM based machine, and is provided here to give a comparison towards systems utilizing only CHIP RAM. This is a stock machine, with accelerator devices or other additional enhancements. AmigaOS 2.x was the operating system used and was located in ROM.

A1200-NF
: Commodore's low-end AGA machine, the Amiga 1200, was used to gather the data for this system. No FAST RAM was used in this machine, and AmigaOS 3.0 (V39.106) in ROM was the operating system present

A3000-25
: The comparison data here was obtained from a 25 MHz CPU rated system, which utilizes the MC68030 CPU and MC68882 FPU as it's processing engines, and equipped with static-column (BURST mode capable) FAST RAM. AmigaOS 2.x was the operating system in use, and was located in ROM on the system A3000's motherboard.

A4000-25
: An Amiga 4000 utilizing a 25 MHz 68040 CPU (stock configuration) was utilized to obtain comparison data. AmigaOS 3.0 was utilized as the system OS (V39.106) and was located in ROM on the motherboard.

It should be kept in mind that all parameters for each system should be noted when making comparisons by checking the statistics located on AIBB's System Information Display. This is especially true if you are comparing a similar based machine to one contained within AIBB. Various parameters can alter performance significantly and need to be accounted for when making fair checks. For example, no systems here were utilizing such performance enhancements as CPUBlit (A program designed to enhance performance by using the CPU instead of the Blitter for certain operations), or utilizing an OS image contained in RAM (which can be faster than ROM images under certain circumstances).

Operating system versions are also important to keep in mind. AmigaOS 3.0 is significantly more optimized in certain graphics respects than its predecessors, and may show up as a significant performance boost in graphics-related tests over a similarly equipped system running a lower OS version. (All systems save for the A4000 here were utilizing the AmigaOS 2.04 mask ROM as their operating system kernel environment...the latter A4000 was utilizing AmigaOS 3.0 contained in ROM).

One other important aspect of performance regarding the Amiga which has come more seriously to light is the question of display parameter effects on test results. With the advent of the AGA chipset and the new display

modes it contains, a great deal more care must be taken when making system comparisons because of the system bus bandwidth limiting effects some modes may have. Please do make sure to note the display mode used on the default systems contained here when comparing systems. Also, when making modules or test result notes, it is wise to carefully monitor what types of screens are currently in use and displayed when AIBB is performing tests.

1.36 notes

Notes and Summary

It has been indicated before, but it should again be emphasized that no benchmark or even suite of benchmarks can hope to give a complete picture of system performance alone. A full picture of the system resources, as well as an understanding of just what the system in question is being used for is necessary to make any type of evalution. AIBB is merely one small tool which may be used to try to gather a sampling of data when making a performance determination.

When performing tests, it is very important to keep track of just where test code and data is being placed in the system by using the information provided by AIBB, and by using other methods if need be. For example, if you have a 512K CHIP RAM machine, and some SLOW-FAST RAM (sometimes mistakenly thought of as true FAST RAM), this could affect test results in ways not expected. Keeping careful track of these variables can help in determining just what is occuring in the system during performance analysis.

Of some interest in terms of FPU performance is the MC68040's built-in FPU unit. This FPU is a subset of Motorola's previous MC68881 and MC68882 coprocessors, and does not include all functions on-chip which were supported by the previous FPUs. Most notably, the transcendental function such as sine and cosine, etc... are not hardware supported. Rather, the simpler functions such as floating-point multiplication, addition, division, etc.. have been greatly optimized and enhanced. The MC68040 FPU relies on software emulation of the complex functions, and most accelerator vendors, as well as CBM itself, supply a function library to emulate these routines in the form of software 'traps'. Since the complex functions utilize the simpler functions to derive their actions, in theory all functions should still execute faster than on previous coprocessors. However, this may not be the case.

Trap functions such as those supplied in the aforementioned libraries are routines executed when the coprocessor indicates an unsupported function routine is being called. This is a form of 'exception' routine, requireing CPU/FPU internal context saving, and other related actions. This is because the CPU/FPU treats the function call as an error, and calls the error routine appropriate to it. In this case, it will be the math support library, which will execute the proper function and return the value needed. Unfortunately, all this activity results in a performance hit, resulting in timings which are longer than that of the previous coprocessors which emulated these functions in their hardware.

All this might imply that the 68040 is crippled in this respect. However, this is not the case. Applications written to take advantage of of 68040's FPU will function much faster, as they will emulate the required complex functions in forms not requiring the trap functions. The trap functions are there for programs which are using FPU code set up for the

MC68881 or MC68882, which are at this time the more common FPU units.
AIBB includes an option, specified earlier, for more efficient 68040 FPU code. This code emulates the transcendental functions an other functions unsupported by the 68040 within AIBB itself. This will alleviate the overhead involved with trap-based emulation methods if selected.

1.37 credits

Credits and Acknowledgements

As with all large projects, nothing is accomplished entirely by one person. I have many people to thank for their assistance in the development of AIBB. A few of the more influential people who have contributed greatly to this effort are:

Kimberly Polglase
For putting up with me throughout this ordeal :)

Redmond Simonsen
One heck of a nice guy and thought provoking fellow. His help with interface ideas was very much appreciated, and are still instrumental in any upcoming future versions of AIBB.

Dr. J. Scott Thayer
Sysop of AmigaFriends BBS, and a dedicated beta tester extraordinaire. His comments and testing data were key to much of what was done with this program over the course of it's development.

Mathew Rouch
A good friend of mine, and a computer science student at present. His help in several algorithmic coding problems allowed me to solve some difficulties which would have taken a great deal longer to overcome than they did.

Unfortunately, I cannot list everyone who has been of assistance with this project, but to all of them, listed and unlisted, I wish to express my deepest thanks and appreciation.
Comments and suggestions about this program are always welcomed, as I hope to be able to continue its development. Please feel free to make any suggestion you see fit, but do try to be constructive in any criticism so that I may improve AIBB. Bug reports are certainly wanted, and I will do my best to locate and correct such problems.
I can be reached electronically many ways, but the following are probably the easiest methods for those with internet access:

lkoop@tigger.stcloud.msus.edu (GP Acct)
f00012@kanga.stcloud.msus.edu (Engineering Acct)

(Pick your paths :))

I can also be found on BIX as "lkoop", and can be reached there easily as well. For those wishing to correspond by mail, comments may be sent to:

LaMonte Koop

1001 Summit Ave. North #125
Sauk Rapids, MN 56379

As for me, well, I'm an Electrical/Computer Engineering student (currently just a wee bit from done) with an added major in Physics, and an emphasis in systems architecture design. AIBB was originally started as a bit of a hobby, and as time went on became a long-standing project. This particular version is almost a year in the making, and I do intend to continue enhancing the package as long as interest remains in it. Enjoy the program; I hope you find it useful, and that it serves whatever purpose you may need of it.