Bedienungsanleitung

PROFESSIONAL – 3000 – BOARD

von

Harms Computer – Systeme

Bremen Februar 1991

© by Harms Computer – Systeme

Autoren: Oliver Pophanken, Oliver Harms

Vorwort

Wann immer Sie Ihren Amiga ausschalten, warten Sie auf jeden Fall mindestens 30 Sekunden, ehe Sie ihn wieder einschalten. Diese Zeit ist nötig, damit die RAM-Bausteine alle Informationen verlieren.

Wenn Sie Ihren Amiga nach kürzerer Zeit wieder einschalten, kann das zur Folge haben, daß eine " Guru Message" auf Ihrem Bildschirm ausgegeben wird. Diese kann durch drücken der linken Maus-Taste zurückgesetzt werden.

Sämtliche Bemerkungen , die sich auf den Amiga 2000 beziehen sind anhand der Benutzerhandbücher nachzuvollziehen, die mit zum Lieferumfang von Commodore gehörten. Selbiges gilt auch für die Anwendung von CLI - Befehlen oder die Benutzung der Workbench.

Der Inhalt dieses Handbuches kann ohne Ankündigung geändert werden und ist nicht als eine Garantieerklärung anzusehen.

Inhaltsverzeichnis

1 Das PROFESSIONAL – 3000 – BOARD

Das PROFESSIONAL – 3000 – BOARD ist eine Erweiterungskarte für den Amiga 2000, um die Rechenleistung des Computers zu erhöhen.
Der Amiga 2000 wird von Commodore mit dem Mikroprozessor 68000 von Motorola ausgeliefert. Dieser MC 68000 ist ein Prozessor der zu seiner Umgebung 16 Bit breite Daten- und Adressleitungen unterhält. Im Laufe der Zeit gesellten sich noch weitere Prozessoren der gleichen Serie dazu. Das sind MC 68010, MC 68020 und der MC 68030. Seit kurzem ist auch der MC 68040 erhältlich. Durch die Verwendung des PROFESSIONAL – 3000 – BOARD wird Ihr Amiga mit dem MC68030 betrieben.
Der Vorteil eines 68030- gegenüber einem 68020- Prozessors besteht hauptsächlich darin, daß er asynchron getaktet werden kann. Das bedeutet, daß der Prozessor nicht den selben Takt besitzen muß wie beispielsweise die Customchips (Agnus, Blitter, Denise, ...) des Amiga. Eine Eigenschaft, die der FPU (68881 und 68882) auch gegeben ist. Neben dem asynchronen Takten besitzt der 68030 ein Instruktion-, Daten-Cache und einen sogenannten Burst-Mode, bei dem sich die Speicherzugriffe pro Langwort enorm verkürzen. Der 68020 hat hingegen nur ein Instruktion-Cache. Neben diesen Änderungen gesellt sich noch eine weitere, die Memory Management Unit 68851 (MMU). Sie ist bereits im 68030 integriert. Damit man auch Software betreiben kann, die nur auf dem 68000 läuft ist das PROFESSIONAL – 3000 – BOARD so konzipiert worden , daß eine Softwareumschaltung möglich ist.
Durch gleichzeitiges Drücken der beiden Maustasten während eines Resets zeigt das PROFESSIONAL – 3000 – BOARD ein Menü, in dem man die Betriebsart der Karte einstellen kann. Weitere Änderungen an der Karte sind durch Jumper auf dem PROFESSIONAL – 3000 – BOARD einstellbar. Die Jumper sind auf Seite 7 beschrieben.

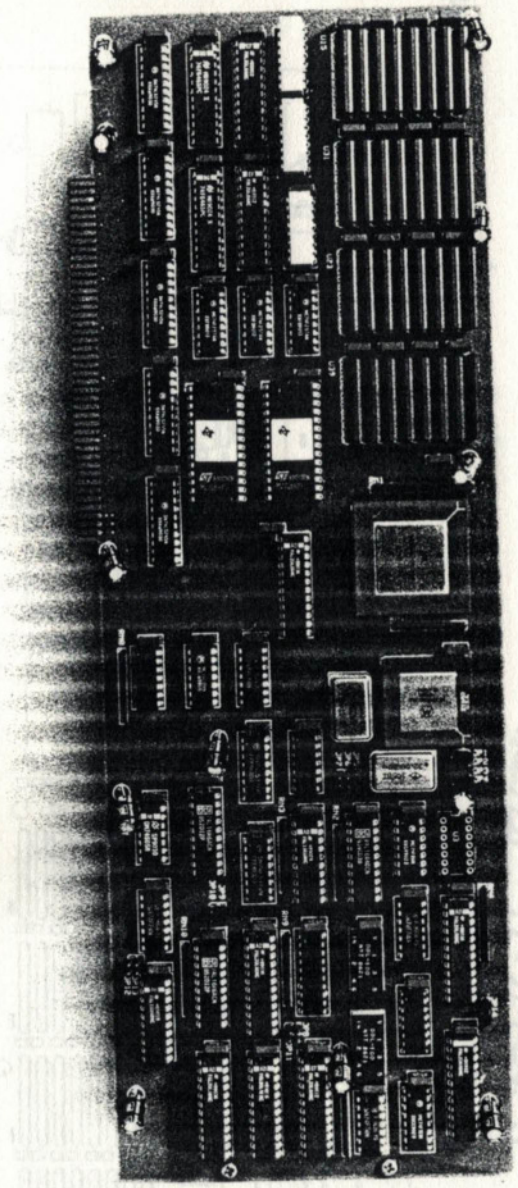
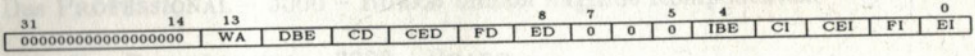


Abbildung 1.1: Das PROFESSIONAL – 3000 – BOARD

Abbildung 1.2: Der PROFESSIONAL – 3000 – BOARD Bestückungsplan

Cache Control Register



WA=Write Allocation
DBE=Data Burst Enable
CD=Clear Data Cache
CED=Clear Entry in Data Cache
FD=Freeze Data Cache
ED=Enable Data Cache
IBE=Instrucion Burst Enable
CI=Clear Instruction Cache
CEI=Clear Entry in Instruction Cache
FI=Freeze Instruction Cache
EI=Enable Instruction Cache

Jumperstellungen

Jumper	Position	Kommentar
JP 1	1-2 X	FPU Async OSC Q1
	2-3	FPU Sync
JP 2	1-2 X	CDU Async OSC Q2
	2-3	CPU Sync
JP 4	1-2 X	Highspeed
	2-3	14 MHz
JP 5	open	Sync Normal
	close X	Sync Highspeed if JP 4 Highspeed
JP 8	open X	AmigaDOS
	close	Other OS
JP 9	open X	2MByte
	close	4 MByte
JP 10	open X	Amiga 2000 B
	close	Amiga 2000 A
JP 11	open X	Autoconfig Memory
	close	Not Autconfig Memory
JP 12	close offen	Nur wenn JP 10 geschlossen (Only if JP 10 closed !!!)
JP 13	close offen	Nur wenn JP 10 geschlossen (Only if JP 10 closed !!!)
JP 14	1-2	+0 Wait
	2-3	+1 Wait

1.1 Lieferumfang

Das PROFESSIONAL – 3000 – BOARD umfaßt folgende Komponenten:

1. Das PROFESSIONAL – 3000 – BOARD
2. Eine Utilitydiskette
3. Eine Anleitung
4. Eine Registrierungskarte

2 Einbau in den Amiga

Vor dem Öffnen des Amigas sind die Garantiebedingungen von Commodore zu beachten.

Weiterhin trennen Sie den Amiga vom Netz. Es sollte auch ausreichend Platz, beim Aufschrauben des Rechners zur Verfügung stehen. Hierzu ist es manchmal sinnvoll den Amiga von den Peripheriegeräten zu lösen.
Für das Öffnen des Amigas benötigt man einen großen Kreuzschraubendreher.

Zuerst werden jeweils zwei Schrauben aus den Seitenwänden gedreht. Auf die gegebenenfalls vorhandenen Unterlegscheiben ist zu achten, da sie bei einem möglichen Fall in den Computer erheblichen Schaden anrichten können.
Die letzte Schraube, welche zum Abnehmen des Gehäusedeckels gelöst werden muß, sitzt an der Rückseite des Rechners (siehe in Bild (2.1) makierte Schraube).

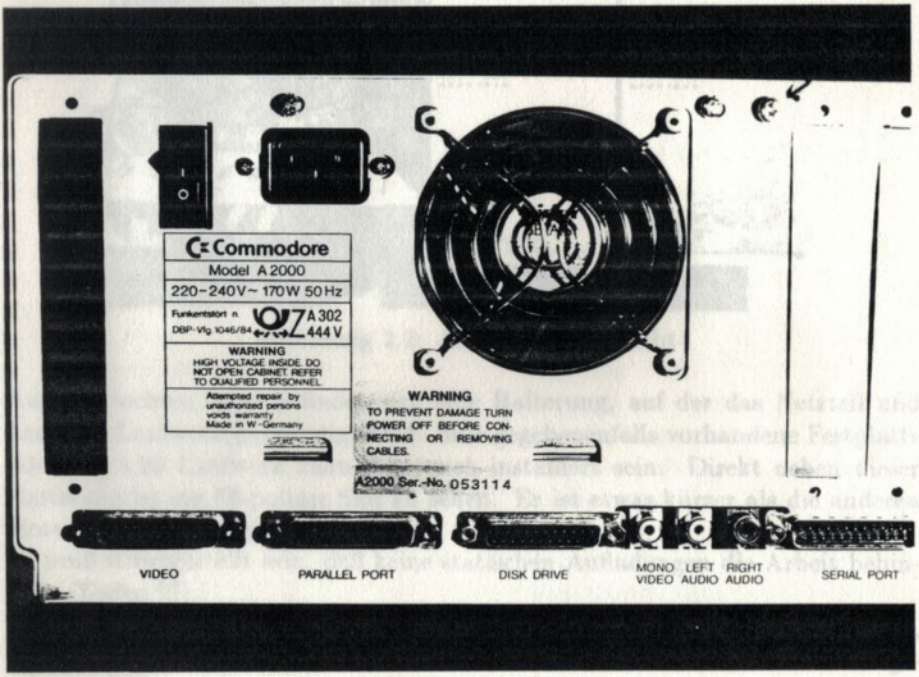


Abbildung 2.1: Amiga 2000 Rückseite

Nachdem die letzte Schraube entfernt ist kann der Gehäusedeckel nach vorne abgezogen werden.

In der Aufsicht kann man die Aufteilung im Amiga gut erkennen (Bild (2.2)).

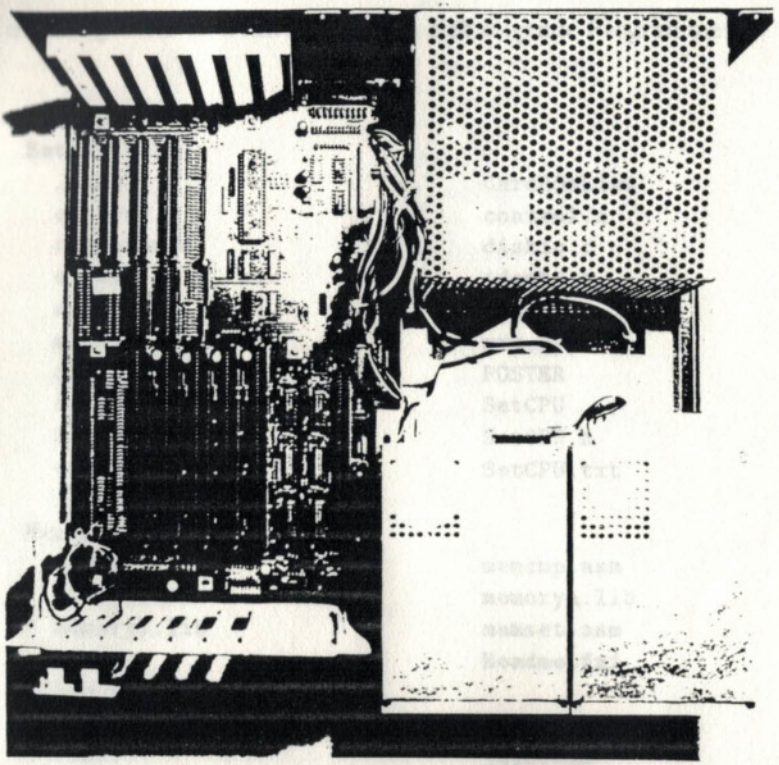


Abbildung 2.2: Amiga 2000 Aufsicht

Auf der rechten Seite befindet sich eine Halterung, auf der das Netzteil und das (die) Laufwerk(e) befestigt sind. Eine gegebenenfalls vorhandene Festplatte oder ein 5.25 Laufwerk kann dort auch installiert sein. Direkt neben dieser Halterung ist ein 86-poliger Slot zu sehen. Er ist etwas kürzer als die anderen Slots auf dem Motherboard.
Es muß sichergestellt sein, daß keine statischen Aufladungen die Arbeit behindern (Erden !!!).
Das PROFESSIONAL – 3000 – BOARD wird, mit der Bauteilseite zum Netzteil, in dem Slot gesteckt. Dabei ist zu beachten, daß die Karte in der dafür vorgesehenen Führung läuft. Die 68000 CPU muß nicht entfernt werden. Mit dem Zusammenbau beginnt man, indem man den Gehäusedeckel von vorne nach hinten über den Amiga geschiebt. Zuerst zieht man die Schraube an der Rückseite an, um den genauen Sitz des Gehäusedeckels zu gewährleisten. Abschließend werden jeweils die zwei Schrauben an den Seiten eingeschraubt.
Der wieder am Netz angeschlossene Amiga ist nun betriebsbereit.

3 Software

Auf der mitgelieferten Diskette befinden sich folgende Programme:

```
Trashcan (dir)
SetCPU (dir)
   .info                          CardROMList
   control.a                      control.i
   coolhand.c                     diskio.c
   expdev.c                       idents.a
   makefile                       memory.c
   misc.c                         mmu.c
   other.a                        POSTER
   reboot.a                       SetCPU
   SetCPU.c                       SetCPU.h
   setcpu.i                       SetCPU.txt
   SetCPU.txt.info
MemRoutines (dir)
   .info                          memcmp.asm
   memcpy.asm                     memorya.lib
   memoryl.lib                    memset.asm
   readme                         Readme.fnf
   Readme.info
JoyLib (dir)
   .info                          joy0.asm
   joy0.o                         JoyDemo
   joydemo.c                      JoyDemo.info
   JoyLib.doc                     JoyLib.Doc.info
   joystick.h                     joystick.library
   joystick_lib.fd                linkdemo.with
   linklib.with                   Readme.fnf
Mathtrans (dir)
   .info                          BenchMark
   BenchMark.c                    BenchMark.info
   Mathtrans.DOC                  Mathtrans.DOC.info
   mathtrans.library              MatLib.asm
   ReadMe.fnf                     ReadMe.fnf.info
   StartBenchMark                 StartBenchMark.info
AmigaBench (dir)
   .info                          AmigaBench
   AmigaBench.asm                 AmigaBench.doc
   AmigaBench.doc.info            AmigaBench.info
```

```
POSTER                      ReadMe.fnf
.info                       AmigaBench.info
Disk.info                   float.030.881
float.030.A.IEEE            float.030.FFP
float.68000                 float.881
float.A.IEEE                float.FFP
float.m.IEEE                JoyLib.info
Mathtrans.info              MemGauge.info
MemRoutines.info            More
More.info                   MuchMore
MuchMore.info               savage.030.881
savage.030.FFP              savage.030.IEEE
savage.68000.ffp            savage.68000.ieee
SetCPU.info                 sieve
Trashcan.info               whetstone.030.881
whetstone.68000
```

3.1 Testprogramme

Im Rootverzeichnis befinden sich einige Programme, mit deren Hilfe man die Funktionen des MC68030 und der FPU 68882 testen kann. Es sind Leistungsvergleiche zwischen den verschiedenen Systemkonfigurationen möglich. Durch Systemänderungen hat man die Möglichkeit auszutesten, bei welcher Konfiguration das System die größte Leistung bringt. Anhand der Dateiendungen, kann man die Art der Compilation erkennen. So ist z.B die Datei float.68000 für den MC68000 compeliert und die Datei float.030.881.IEEE für den MC68000 unf FPU68881 unter Verwendung der IEEE Mathe-Library. Das .a. steht fü die Amiga-Library und das .m. für die Manx-Library.

3.1.1 Float

Mit diesem Test werden Leistungsdaten von Fließkommaoperationen der unterschiedlichen Prozessoren ermittelt. Der Code liegt in mehreren Variationen vor:

1. Für den MC68030 mit FPU68881/2 Unterstützung
2. Für den MC68030 mit IEEE Mathe-Library
3. Für den MC68030 mit MANX Mathe-Library
4. Für den MC68030 mit Motorola Fast Floating Point-Emulation
5. Für den MC68000

Source - Listing des Programms Float. Durch unterschiedliche Optionen am Compiler und Linker erhält man den Code für die verschiedenen Prozessoren.

```
/*********************************************************/
/*                        Float.c                        */
/* Tests speed at which a system does double precision */
/* floating-point multiply and divide instructions. A  */
/* total of 140,000 Double Precision FP operations.    */
/*                                                       */
/*                         Flt.c                         */
/* Tests speed at which a system does double precision */
/* floating-point add, subtract, multiply, and divide  */
/* instructions.  Roughly 1 million DP FP operations.  */
/*********************************************************/

#include <math.h>
#include <stdio.h>
```

```
/********************/
/*     Float.c      */
/********************/
#define CONST1 3.1415970E0
#define CONST2 1.7839032E4
#define COUNT  10000

main()
{
   double a,b,c;
   long i,j,loops;
   long  starttime,stoptime,benchtime;
   long  nulltime,timeticks();       /* timeticks() is specific to */
                                     /* the Amiga. Replace with    */
                                     /* with your system timer     */
   starttime = timeticks();
   stoptime  = timeticks();
   nulltime  = stoptime - starttime;

   printf("\n   Float Benchmark\n");
   printf("   Start %d Loops \n",COUNT);

   starttime = timeticks();              /* Start timer. */

   a = CONST1;
   b = CONST2;

   for(i=0;i<COUNT;++i)
      {
      c = a*b;
      c = c/a;

      c = a*b;
      c = c/a;

      c = a*b;
      c = c/a;

      c = a*b;
      c = c/a;

      c = a*b;
      c = c/a;

      c = a*b;
      c = c/a;
```

```
        c = a*b;
        c = c/a;
        }

    stoptime  = timeticks();                /* Stop timing. */

    benchtime = stoptime - starttime - nulltime;
    a = (double)benchtime/50.0;

    printf("   Done.\n");
    printf("   Benchtime (sec) = %lf\n\n",a);


/***************/
/*    Flt.c    */
/***************/

    loops = 256000;
    printf("   FLT Benchmark\n");
    printf("   Start %ld loops.\n",loops);

    starttime = timeticks();                /* Start timing */

    for( i=1 ; i<=loops ; i++ )
    {
    j = loops - i;
    a = (double)i;
    b = (double)j;
    c = b / a;
    a = b - c;
    b = c * a;
    c = b + a;
    }

    a = c + b;

    stoptime  = timeticks();            /* Stop timing. */

    benchtime = stoptime - starttime - nulltime;
    a = (double)benchtime/50.0;

    printf("   Done.\n");
    printf("   Benchtime (sec) = %lf\n\n",a);
}
```

```
/*******************************************************/
/* Function Returns ticks ( 50 per sec) since midnight.*/
/* Amiga specific, Replace with your own routine.      */
/*******************************************************/

long  timeticks()
{
   struct   ds {
      long  days;
      long  minutes;
      long  ticks;
   }  ds;
   DateStamp(&ds);
   return(ds.ticks + (ds.minutes * 60 * 50));
}
```

3.1.2 Sieve-Test

Das Programm Sieve ist ein reiner CPU–Test. Dieser Test läuft ohne Unterstützung des mathematischen Coprozessors.
Source - Listing des Programms Sieve. Durch unterschiedliche Optionen am Compiler und Linker erhält man den Code für die verschiedenen Prozessoren.

```
#include <stdio.h>

#define TRUE 1
#define FALSE 0
#define SIZE 8190
#define ITER 100

char flags[SIZE+1];
/*************************************************************/
long sync_ticks()
{
struct TIME
{
long days,minutes,ticks;
} t;
long ticks;

DateStamp(&t);
ticks = t.ticks;

do
{
DateStamp(&t);
} while (t.ticks == ticks);

return(3000*t.minutes + t.ticks);
}
/*************************************************************/
long cur_ticks()
{
struct TIME
{
long days,minutes,ticks;
} t;

DateStamp(&t);
return(3000*t.minutes + t.ticks);
}
/*************************************************************/
```

```
print_ticks_as_seconds(ticks)
long ticks;
{
printf("%ld.%02ld seconds", ticks/50, (ticks%50)*2);
}
/*************************************************************/
main(argc,argv)
int argc;
char **argv;
{
int i,k;
int prime, count, iter;
long start, end;

printf("%d iterations:\n", ITER);

start = sync_ticks();

for (iter = 1; iter <= ITER; iter++)
{
count = 0;
for (i = 0; i <= SIZE; i++)
{
flags[i] = TRUE;
}
for (i = 0; i<= SIZE; i++)
{
if (flags[i])
{
prime = i + i + 3;
for (k = i + prime; k <= SIZE; k += prime)
{
flags[k] = FALSE;
}
count++;
}
}
}

end = cur_ticks();

print_ticks_as_seconds(end - start);
printf(" seconds, %d primes.\n", count);

return(0);
}
```

3.1.3 Savage-Test

Dieses Programm gibt es in drei verschiedenen Ausführungen:

1. Savage.030.881 Ist ein reiner Floatingpointtest des mathematischen Coprozessors.

2. Savage.68000.ieee Ist ein Floatingpointtest nach ieee Standard. Hierzu werden die neuen Libaries der Workbench 1.3 benötig. Diese Libaries unterstützten auch den mathematischen Coprozessor. Den Vorteil, den man durch die Verwendung dieser Libary hat, liegt darin begründet, daß der Code mit und ohne Coprozessor läuft.

3. Savage.68000.ffp Ist ein Floatingpointest nach Motorola Fast-Floating-Point Standard.

4. Savage.030.ieee Hier wird die Architektur des 68030 mit zur Optimierung mit ausgenutzt

5. Savage.68030.ffp Auch hier wird die Architektur des 68030 mit ausgenutzt.

Source - Listing des Programms Savage. Durch unterschiedliche Optionen am Compiler und Linker erhält man den Code für die verschiedenen Prozessoren.

```
#include <stdio.h>

#define ILOOP 2500

extern double tan(), atan(), exp(), log(), sqrt();

/*************************************************************/
sync_ticks()
{
struct TIME
{
long days,minutes,ticks;
} t;
long ticks;

DateStamp(&t);
ticks = t.ticks;

do
{
DateStamp(&t);
} while (t.ticks == ticks);
```

```
return(3000*t.minutes + t.ticks);
}
/************************************************************/
cur_ticks()
{
struct TIME
{
long days,minutes,ticks;
} t;

DateStamp(&t);
return(3000*t.minutes + t.ticks);
}
/************************************************************/
print_ticks_as_seconds(ticks)
int ticks;
{
printf("%d.%02d seconds", ticks/50, (ticks%50)*2);
}
/************************************************************/
main()
{
int i;
double a;
int start, end;

printf("%d iterations:\n", ILOOP);
start = sync_ticks();

a = 1.0;

for (i=1; i<ILOOP; i++)
a = tan(atan(exp(log(sqrt(a*a))))) + 1.0;

end = cur_ticks();

printf("a = %20.14lf\n", a);
printf("Done in ");
print_ticks_as_seconds(end - start);
printf(".\n");
}
```

3.1.4 Whetstone-Test

Source - Listing des Programms Whetstone. Durch unterschiedliche Optionen am Compiler und Linker erhält man den Code für die verschiedenen Prozessoren.

```
/*************************************************************************
 *                                                                       *
 *  Whetstone benchmark in C.  This program is a translation of the      *
 *  original Algol version in "A Synthetic Benchmark" by H.J. Curnow     *
 *  and B.A. Wichman in Computer Journal, Vol  19 #1, February 1976.     *
 *                                                                       *
 *  Used to test compiler efficiency, optimization, and double           *
 *  precision floating-point performance.  This version is specific      *
 *  to the Turbo-Amiga and Amiga but it can be easily adapted to         *
 *  other systems by replacing the timeticks() routine with your own.    *
 *                                                                       *
 *************************************************************************/

#define ITERATIONS   10        /* 1 Million Whetstone instructions    */
/* #define POUT  */            /* Leave as is for 'Official' result.  */
#define MTEST                  /* define for Module timing results    */
                               /* only.  Leave as is for 'Official'   */
                               /* result.                             */
                               /* These includes are specific to my   */
                               /* Turbo-Amiga/Amiga system.  Replace  */
                               /* with your own set.                  */
#include <stdio.h>
#include <math.h>
double atan(),acos(),asin(),tan(),cos(),sin();
double log(),exp(),sqrt();
double   e1[4],e2[4];
double   t, t1, t2, x, y, z, x1, x2, x3, x4;
long     i, j, k, l;
long     n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11;
long     m, m1, m2, loops, KWhets;
long  starttime, stoptime, nulltime;
long  sumtime, benchtime,timeticks(); /* Replace timeticks() with */
                                      /* your own system timing   */
                                      /* routine.                 */
main()
{
   printf("\n   Whetstone C Benchmark V1.0\n");

   starttime = timeticks();          /* Calculate timer call delay */
   stoptime  = timeticks();
```

```
nulltime  = stoptime - starttime;

/*************************/
/* initialize constants */
/*************************/

t   =    0.499975;
t1  =    0.500250;
t2  =    2.0;

/************************/
/* Set Module Weights. */
/************************/

m = 10;                    /* m = 10 is used to obtain better timing  */
loops = m * ITERATIONS; /* accuracy only.  Slow systems should use */
n1  =   0 * loops;         /* m = 1.                                  */
n2  =  12 * loops;
n3  =  14 * loops;
n4  = 345 * loops;
n5  =   0 * loops;
n6  = 210 * loops;
n7  =  32 * loops;
n8  = 899 * loops;
n9  = 616 * loops;
n10 =   0 * loops;
n11 =  93 * loops;

                        /* Start Timing the Whetstone here. */

starttime = timeticks();

/*********************************/
/* MODULE 1:  simple identifiers */
/*********************************/

x1 =  1.0;
x2 = -1.0;
x3 = -1.0;
x4 = -1.0;

if( n1 > 0 )
{
  for(i = 1; i <= n1; i++)
  {
   x1 = ( x1 + x2 + x3 - x4 ) * t;
```

```
      x2 = ( x1 + x2 - x3 - x4 ) * t;
      x3 = ( x1 - x2 + x3 + x4 ) * t;
      x4 = (-x1 + x2 + x3 + x4 ) * t;
    }
  }
#ifdef MTEST                          /* Module 1 Run Time */
  stoptime  = timeticks();
  benchtime = stoptime - starttime - nulltime;
  m = 1;
  x = (double)benchtime/50.0;
  printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x);
#endif

#ifdef POUT
  Pout(n1, n1, n1, x1, x2, x3, x4);
#endif

  /*****************************/
  /* MODULE 2:  Array Elements */
  /*****************************/
#ifdef MTEST
  starttime = timeticks();
#endif

  e1[0] =  1.0; /* Start at element 0 in C, vice 1 in Fortran */
  e1[1] = -1.0;
  e1[2] = -1.0;
  e1[3] = -1.0;

  if( n2 > 0 )
  {
    for (i = 1; i <= n2; i++)
    {
      e1[0] = ( e1[0] + e1[1] + e1[2] - e1[3] ) * t;
      e1[1] = ( e1[0] + e1[1] - e1[2] + e1[3] ) * t;
      e1[2] = ( e1[0] - e1[1] + e1[2] + e1[3] ) * t;
      e1[3] = (-e1[0] + e1[1] + e1[2] + e1[3] ) * t;
    }
  }

#ifdef MTEST
  stoptime  = timeticks();
  benchtime = stoptime - starttime - nulltime;
  sumtime   = benchtime;
  m = 2;
```

```
   x = (double)benchtime/50.0;
   printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x);
#endif

#ifdef POUT
   Pout(n2, n3, n2, e1[0], e1[1], e1[2], e1[3]);
#endif

   /*********************************/
   /* MODULE 3:  Array as Parameter */
   /*********************************/
#ifdef MTEST
   starttime = timeticks();
#endif

   if( n3 > 0 )
   {
     for (i = 1; i <= n3; i++)
     {
      pa(e1);
     }
   }

#ifdef MTEST
   stoptime  = timeticks();
   benchtime = stoptime - starttime - nulltime;
   m1 = benchtime;
   m = 3;
   x = (double)benchtime/50.0;
   printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x);
#endif

#ifdef POUT
   Pout(n3, n2, n2, e1[0], e1[1], e1[2], e1[3]);
#endif

   /*********************************/
   /* MODULE 4:  Conditional Jumps */
   /*********************************/
#ifdef MTEST
   starttime = timeticks();
#endif

   j = 1;

   if( n4 > 0 )
```

```
    {
      for (i = 1; i <= n4; i++)
      {
        if (j == 1)
            j = 2;
        else
            j = 3;

        if (j > 2)
            j = 0;
        else
            j = 1;

        if (j < 1 )
            j = 1;
        else
            j = 0;
      }
    }

#ifdef MTEST
    stoptime  = timeticks();
    benchtime = stoptime - starttime - nulltime;
    sumtime   = sumtime + benchtime;
    m = 4;
    x = (double)benchtime/50.0;
    printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x);
    starttime = timeticks();
#endif

#ifdef POUT
    Pout(n4, j, j, x1, x2, x3, x4);
#endif

    /**********************/
    /* MODULE 5:  Omitted */
    /**********************/

#ifdef MTEST
    benchtime = 0;
    m = 5;
    x = 0.0;
    printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x);
#endif
```

```
#ifdef POUT
   Pout(n4, j, j, x1, x2, x3, x4);
#endif

   /*********************************/
   /* MODULE 6:  Integer Arithmetic */
   /*********************************/
#ifdef MTEST
   starttime = timeticks();
#endif

   j = 1;
   k = 2;
   l = 3;

   if( n6 > 0 )
   {
    for (i = 1; i <= n6; i++)
    {
     j = j * (k - j) * (l -k);
     k = l * k - (l - j) * k;
     l = (l - k) * (k + j);

     e1[l - 2] = j + k + l; /* Remember we started at e1[0]. */
     e1[k - 2] = j * k * l; /* l-2 in C, vice l-1 in Fortran */
    }
   }

#ifdef MTEST
   stoptime  = timeticks();
   benchtime = stoptime - starttime - nulltime;
   sumtime   = sumtime + benchtime;
   m = 6;
   x = (double)benchtime/50.0;
   printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x);
#endif

#ifdef POUT
   Pout(n6, j, k, e1[0], e1[1], e1[2], e1[3]);
#endif

   /*************************************/
   /* MODULE 7:  Trigonometric Functions */
   /*************************************/
#ifdef MTEST
   starttime = timeticks();
```

```
#endif
   x = 0.5;
   y = 0.5;

   if( n7 > 0 )
   {
     for(i = 1; i <= n7; i++)
     {
       x = t * atan(t2*sin(x)*cos(x)/(cos(x+y)+cos(x-y)-1.0));
       y = t * atan(t2*sin(y)*cos(y)/(cos(x+y)+cos(x-y)-1.0));
     }
   }

#ifdef MTEST
   stoptime  = timeticks();
   benchtime = stoptime - starttime - nulltime;
   sumtime   = sumtime + benchtime;
   m = 7;
   x1 = (double)benchtime/50.0;
   printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x1);
#endif

#ifdef POUT
   Pout(n7, j, k, x, x, y, y);
#endif

   /*****************************/
   /* MODULE 8:  Procedure Calls */
   /*****************************/
#ifdef MTEST
   starttime = timeticks();
#endif

   x = 1.0;
   y = 1.0;
   z = 1.0;

   if( n8 > 0 )
   {
     for (i = 1; i <= n8; i++)
     {
       p3(x, y, &z);
     }
   }
```

```
#ifdef MTEST
   stoptime  = timeticks();
   benchtime = stoptime - starttime - nulltime;
   m1 = m1 + benchtime;
   m = 8;
   x1 = (double)benchtime/50.0;
   printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x1);
#endif

#ifdef POUT
   Pout(n8, j, k, x, y, z, z);
#endif

   /*******************************/
   /* MODULE 9:  Array References */
   /*******************************/
#ifdef MTEST
   starttime = timeticks();
#endif

   j = 1;
   k = 2;
   l = 3;

   e1[0] = 1.0;
   e1[1] = 2.0;
   e1[2] = 3.0;

   if( n9 > 0 )
   {
     for(i = 1; i <= n9; i++)
     {
      p0();
     }
   }

#ifdef MTEST
   stoptime  = timeticks();
   benchtime = stoptime - starttime - nulltime;
   m1 = m1 + benchtime;
   m = 9;
   x = (double)benchtime/50.0;
   printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x);
#endif

#ifdef POUT
```

```
   Pout(n9, j, k, e1[0], e1[1], e1[2], e1[3]);
#endif

   /*********************************/
   /* MODULE 10:  Integer Arithmetic */
   /*********************************/
#ifdef MTEST
   starttime = timeticks();
#endif

   j = 2;
   k = 3;

   if( n10 > 0 )
   {
     for(i = 1; i <= n10; i++)
     {
      j = j + k;
      k = j + k;
      j = k - j;
      k = k - j - j;
     }
   }

#ifdef MTEST
   stoptime  = timeticks();
   benchtime = 0;
   m = 10;
   x = 0.0;
   printf("   End Module %ld.  Benchtime(sec) = %lf\n",m,x);
#endif

#ifdef POUT
   Pout(n10, j, k, x1, x2, x3, x4);
#endif

   /*********************************/
   /* MODULE 11:  Standard Functions */
   /*********************************/
#ifdef MTEST
   starttime = timeticks();
#endif

   x = 0.75;

   if( n11 > 0 )
```

```
    {
      for(i = 1; i <= n11; i++)
      {
       x = sqrt( exp( log(x) / t1) );
      }
    }

#ifdef MTEST
    stoptime  = timeticks();
    benchtime = stoptime - starttime - nulltime;
    sumtime   = sumtime + benchtime;
    m = 11;
    x1 = (double)benchtime/50.0;
    printf("   End Module %ld.  Benchtime(sec) = %lf\n\n",m,x1);
#endif

#ifdef POUT
    Pout(n11, j, k, x, x, x, x);
#endif

/*********************************************************/
/* MODULE 12:  Array as a Parameter.                     */
/* Same as Module 3 except Subroutine overhead removed. */
/*********************************************************/

#ifdef MTEST

    starttime = timeticks();
    if( n3 > 0 )
    {
      for (i = 1; i <= n3; i++)
      {
       j = 0;
         lab:
       e2[0] = (  e2[0] + e2[1] + e2[2] - e2[3] ) * t;
       e2[1] = (  e2[0] + e2[1] - e2[2] + e2[3] ) * t;
       e2[2] = (  e2[0] - e2[1] + e2[2] + e2[3] ) * t;
       e2[3] = ( -e2[0] + e2[1] + e2[2] + e2[3] ) /t2;
       j++;

       if (j < 6)
          goto lab;
      }
    }

    stoptime  = timeticks();
```

```
    benchtime = stoptime - starttime - nulltime;
    m2    = benchtime;
    m = 3;
    x1 = (double)benchtime/50.0;
    printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x1);

#endif

#ifdef POUT
    Pout(n11, j, k, x, x, x, x);
#endif

/*************************************************************/
/* MODULE 13:  Array as a Parameter.                         */
/* Same as Module 8 except Subroutine overhead removed. */
/*************************************************************/

#ifdef MTEST

    starttime = timeticks();
    x = 1.0;
    y = 1.0;
    z = 1.0;

    if( n8 > 0 )
    {
      for (i = 1; i <= n8; i++)
      {
       x = t * (x + y);
       y = t * (x + y);
       z = (x + y) /t2;
      }
    }

    stoptime  = timeticks();
    benchtime = stoptime - starttime - nulltime;
    m2    = m2 + benchtime;
    m = 8;
    x1 = (double)benchtime/50.0;
    printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x1);

#endif

#ifdef POUT
    Pout(n11, j, k, x, x, x, x);
#endif
```

```
/*************************************************************/
/* MODULE 14:  Array as a Parameter.                         */
/* Same as Module 9 except Subroutine overhead removed. */
/*************************************************************/

#ifdef MTEST

   starttime = timeticks();
   j = 1;
   k = 2;
   l = 3;

   e1[0] = 1.0;
   e1[1] = 2.0;
   e1[2] = 3.0;

   if( n9 > 0 )
   {
     for(i = 1; i <= n9; i++)
     {
      e1[j] = e1[k];
      e1[k] = e1[l];
      e1[l] = e1[j];
     }
   }

   stoptime  = timeticks();
   benchtime = stoptime - starttime - nulltime;
   m2 = m2 + benchtime;
   m = 9;
   x1 = (double)benchtime/50.0;
   printf("   End Module %ld.   Benchtime(sec) = %lf\n",m,x1);

#endif

#ifdef POUT
   Pout(n11, j, k, x, x, x, x);
#endif

   /*************************/
   /* End of Whetstone Tests */
   /*************************/

#ifdef MTEST
   m1 = m1 + sumtime;
```

```
    m2 = m2 + sumtime;
    x1 = (double)m1/50.0;
    x2 = (double)m2/50.0;

    printf("Standard Whetstone Result\n");
    printf("   Benchtime(sec) = %lf\n",x1);
    x  = 100.0 * (double)loops / x1;
    KWhets = (long)x;
    printf("   KWhets/sec     = %ld\n",KWhets);

    printf("Result with subroutine call replaced with subroutine code\n");
    printf("   Benchtime(sec) = %lf\n",x2);
    x  = 100.0 * (double)loops / x2;
    KWhets = (long)x;
    printf("   KWhets/sec     = %ld\n",KWhets);

#else

    stoptime  = timeticks();
    benchtime = stoptime - starttime - nulltime;
    x1 = (double)benchtime/50.0;
    printf("   Benchtime(sec) = %lf\n",x1);
    x2 = 100.0 * (double)loops / x1;
    KWhets = (long)x2;
    printf("   KWhets/sec     = %ld\n\n",KWhets);

#endif

    exit(0);

}

/*******************/
/* Subroutine pa() */
/*******************/

pa(e)     /* Exactly as in the Algol 60 version, but we */
          /* could do away with that 'goto'.            */

double e[4];

{
    int j;

    j = 0;
      lab:
```

```
    e[0] = (  e[0] + e[1] + e[2] - e[3] ) * t;
    e[1] = (  e[0] + e[1] - e[2] + e[3] ) * t;
    e[2] = (  e[0] - e[1] + e[2] + e[3] ) * t;
    e[3] = ( -e[0] + e[1] + e[2] + e[3] ) / t2;
    j ++;

    if (j < 6)
       goto lab;
}

/************************/
/* Subroutine p3(x,y,z) */
/************************/

p3(x, y, z)

double x, y, *z;

{
    x  = t * (x + y);
    y  = t * (x + y);
    *z = (x + y) /t2;
}

/********************/
/* Subroutine p0() */
/********************/

p0()
{
    e1[j] = e1[k];
    e1[k] = e1[l];
    e1[l] = e1[j];
}

/**********************/
/* Subroutine Pout() */
/**********************/

#ifdef POUT
Pout(n, j, k, x1, x2, x3, x4)

long  n, j, k;
double x1, x2, x3, x4;

{
```

```
    printf("%5ld %5ld %5ld   %11.3le %11.3le %11.3le %11.3le\n",
        n, j, k, x1, x2, x3, x4);
}
#endif

/*************************************/
/* Amiga Time Ticks ( 50 * seconds ) */
/*************************************/

long timeticks()
{
      struct   tt {
         long  days;
         long  minutes;
         long  ticks;
      } tt;
      DateStamp(&tt);
  return (tt.ticks + (tt.minutes * 60L * 50L ));
}

/*----------- End Of Whetstone C Source Code ----------*/
```

3.1.5 AmigaBench

AmigaBench ist ein für den Amiga angepaßte Version des allgemein bekannten Benchmark-Tests Dhrystone. Es beinhaltet mehrere Optionen. Diese Optionen bekommt man durch anhängen des Fragezeichens an den Befehl aufgelistet. Das Programm wurde geschrieben von Al Aburto, von dem auch folgende Anleitung stammt:

AmigaBench

Al Aburto
'ala' on BIX
05 Aug 1989

Hello Fellow Amigeans!

AmigaBench is a program in which I intended to include 680X0 Amiga assembly optimized versions of the more popular so-called 'benchmark' programs.

I've only made it to the Dhrystone so far, but I thought the results are interesting enough to upload what I got now. Besides I think someone like 'Jez' could find even more optimizations to do :-).

AmigaBench will run from Workbench or the CLI/AmigaShell. I included a dumb ICON, executable file, and the assembly source code. Its all public domain of course.

Just type '?' and hit the RETURN key and a list of the program commands will be printed to the screen.

1. The 'ca' command just shows you where in memory the program was loaded (_main), the addresses of some of the routines, and the location of the global data (_A4Ref).

2. The 'cs' command prints out the Cache Status if you have an 020 or 030 CPU.

3. The 'wc' command allows you to Write to the Cache (if you have an 020 or 030 CPU). For example '51' will turn the 'Write Allocate' feature ON, and '50' will turn it OFF. The various options are shown.

4. The '1' command runs what I call the 'Standard 68000 Assembly Version' of the Dhrystone. The run time in TICKS and the Dhrystones/sec are printed out. Also your CPU/FPU type is printed out. I don't check for the 68030 and 68882 in the program (I have another independent program in my startup-sequence that I use). So if the ExecBase flags are not set right AmigaBench will not report the 68030 or 68882 even if they are really installed. I'll fix this later.

5. The '2' command runs an 'Optimized 68000 Assembly Version' of the Dhrystone.

6. The '3' command runs an 'Optimized 68020 Assembly Version' of the Dhrystone. This should be the fastest version on those 020 and 030 systems.

That's it for now.

I'm getting about 5400 Dhrystones/sec from my CSA 14.32 MHz 68020 board with 32-bit fast static RAM. I'm sure Ronin will be faster and the GVP 030 really fast at 25 MHz or more. Post or send me results please.

Al Aburto
05 Aug 1989

3.2 Utility - Programme

Auf der Diskette befinden sich noch weitere Programme. Diese sind thematisch in Unterverzeichnisse gegliedert.

3.2.1 SetCPU V 1.6 Originalanleitung

SetCPU V1.60
by Dave Haynie
June 15, 1990

SetCPU V1.60 is a program designed for identification and modification of system parameters roughly related to different versions of the Motorola 68000 family processors. The program will identify the various types of processors and coprocessors in any 680x0 system. It also makes an attempt to correctly identify an incorrectly designed but still possibly functional 68020 system, several of which are known to exist as Amiga coprocessor boards. It contains MMU code to locate kernel ROM in write protected 32 bit ROM.

In any case, the syntax of the program is given as follows:

SetCPU [INST|DATA] [[NO]CACHE|[NO]BURST] [CONFIG n] [BITS n] [TRAP n] [KICKROM path|dfN:] [DELAY n] [KEEPEXEC]] [CARDROM path] [VERBOSE] [[NO]FASTROM [path] [KEYPATCH n] [HEAD] [NOSTACK]] [ROMBOOT]
[CHECK 680x0|68851|6888x|MMU|FPU|MMUON|MMUROM| MMUALIEN]

where "[ ]" indicates an optional parameter, "|" indicates a choice of parameters. Typing "SetCPU ?" will retrieve this same syntax diagram. Typing SetCPU alone will result in the SYSTEM configuration being send to the console, my current system returns this:

SYSTEM: 68030 68882 FASTROM (INST: CACHE NOBURST) (DATA: CACHE NOBURST)

This indicates I have a 68030/68882 system, I've previously installed the FASTROM translation, and both caches are turned on. Issuing the command "SetCPU FASTOM CACHE" would recreate such a setup. Note that any parameters that don't make sense to the real system configuration, such as asking to modify the data cache on a 68020 system or install the FASTROM translation on a 68000 system are just ignored.

1. CHANGES

Since the V1.50 release of SetCPU, the following changes have been made to the SetCPU program:

- FASTROM now supports loading of a ROM image file.
- The patch manager has been removed, except for the KEYPATCH option.
- ROMs beyond 256K are now supported, from both file and special Kick-Start disk.
- ROM images are supported assembled for various memory locations, so that Commodore's developer files assembled at $00F00000, for example, will work. SetCPU determines the size, base, and jump address of a ROM independently.
- ExecBase can be erased on KICKROM resets to prevent Chip RAM sizing problems and other OS switching errors.
- A programmable KICKROM DELAY option allows KICKROM to work on better on some machines with extremely slow 8520 startup times.
- Caching for any Bridge Card memory will always be disabled.
- The system stack, if found in Chip memory, is relocated to Fast memory for FASTROM translations.
- Some 2.0 functions are activated when run under the 2.0 OS, so that most if not all of the basic SetCPU functions operate properly.
- The MMU code is intelligent enough to avoid trashing an MMU setup not generated by SetCPU.

2. DISTRIBUTION

This program is placed in the public domain, and may be used or distributed as you like.

3. CPU IDENTIFICATION

There are two basic types of functions performed by SetCPU. The first of these is CPU system identification and cache control. SetCPU will tell about the type of CPU setup in your machine, which consists of the CPU itself and sometimes FPU or MMU coprocessors. If the CPU supports caches, SetCPU will let you switch these caches, and associated cache line burst mode, on and off. Finally, SetCPU can be used in a Startup-Sequence or other script to make decisions based on the system that's running. This is quite useful with accelerator cards like the Commodore A2620 that let you boot the machine with either 68020 or 68000 in charge. The individual CPU group commands are given below in detail:

NO CACHE

This command will switch on or off 68020 and 68030 caches. If not qualified, it'll act on both instruction and data caches of the 68030.

NO BURST

This command will switch on or off the burst cache line fill request of the 68030. If not qualified, it'll act on both instruction and data caches.

INST

This qualifies a CACHE or BURST operation to restrict its application to the instruction cache only.

DATA

This qualifies a CACHE or BURST operation to restrict its application to the data cache only.

CHECK

This option lets you check for the existence of a particular CPU system component in a script. It works like this:

```
SetCPU CHECK 68020
If WARN
    echo "No 68020 here!"
Else
    echo "Sho nuff got a 68020 here!"
Endif
```

The arguments to CHECK can be any of:

68000	Matches the obvious
68010	"
68020	"
68030	"
68040	"
68851	"
68881	"
68882	"
FPU	Matches 68881, 68882, or 68040
MMU	Matches 68851, 68030, or 68040
MMUON	Matches any case in which the MMU is enabled
MMUROM	Matches an active SetCPU ROM translation
MMUALIEN	Matches any MMU setup no created by SetCPU

If any cache parameter doesn't apply to the system in use, it'll just be ignored. Use the data cache and all burst modes with caution. Some 68030 systems aren't designed to correctly support the data cache, so switching it on may cause an instant system crash. Even on systems that correctly support the 68030 data cache, some device drivers, especially those for DMA devices, may not work properly with the data cache enabled. You may wish to check with your system vendors to make sure before using the data cache in your standard system setup. The Commodore A2091's device driver does correctly support

data caching. However, the use of the data cache is not recommended without an MMU setup, such as FASTROM or one of the KICK setups, invoked. The Amiga OS uses memory that's the same in both Supervisor and User modes of the 680x0. This requires the setting of the 68030's Write Allocate bit for safe operation, and SetCPU will always insure that Write Allocate is set. However, this mode causes the data cache to be updated on longword writes even for locations that are driven noncacheable in hardware. So data caching without a proper MMU setup can cause problems with some I/O devices. With the MMU setup, SetCPU will map the standard Amiga I/O regions as noncachable.

SetCPU may report a "FPU Logic Error" on certain 68020 systems. This is indicating a hardware problem with that board's floating point coprocessor decoding, which results in the FPU responding to the MMU addresses as well as its own. SetCPU knows how to handle such a board, but future software using the MMU may not, so it's a good idea to report this problem to the board vendor for repair.

4. ROM TRANSLATIONS

The second thing that SetCPU V1.60 manages are ROM translations. Using the MMU on systems so equipped, it can locate the Kernel ROM in the much faster 32 bit wide memory provided on many 32 bit systems. It can also boot a ROM based system with an alternate version of KickStart. Most of the options here relate to MMU translation setup and various modifications of the basic translation premise.

As of this release, SetCPU's MMU configurations will support memory outside of the 68000's 24 bit address space, when it is present. Such memory will be automatically recognized and supported by SetCPU if it is linked into the free memory lists when SetCPU builds its FASTROM or KICKROM. Alternatively, the number of significant bits of address in the system can be specified by the BITS command, and SetCPU will build the appropriate MMU table for such a system.

Another feature of this release is support for ROM images of either 256K or 512K in size. KickROMs may be assembled for locations other than the $00FC0000 or $00F80000 base used by physical systems. SetCPU will compute the size, base address, and start address for any KickROM image. ROM images can now be loaded from disk for FASTROM translations as well as KICKROM translations, though a FASTROM image must be the same KickStart release as the current ROM in the system. This facility's main purpose is to support loading of patched ROMs with the necessity of rebooting via KICKROM. As a result of this, the only patch now done by SetCPU itself is the optional KEYPATCH, which may be required for proper operation of the accelerated ROM code on some systems.

(a) **[NO]FASTROM** This activates the FASTROM translation on or off an MMU equipped system. When switching on, it first allocates at least 256K of memory for the ROM image, then at least 512 bytes of memory for the MMU table. It copies the ROM into the image area, then applies the translation by pointing the MMU at the table and activating it. The NOFASTROM option will switch off the MMU and reclaim the memory

used for the ROM image and MMU table. If any other program set up the MMU for something, invoking this option could be a very bad thing to do. In general, until there's some level of OS support for the MMU in Amiga systems, you're really safe using only one MMU tool at a time. If you have an A2620 or A2630 system, this option will always get 32 bit memory for you; if not, you'll have to make sure that your 32 bit memory is the first MEMF_FAST memory in the memory list for it to be used for the ROM image. Also, that ROM image will be allocated as far back on that memory list as possible unless the "HEAD" option is specified. The SetCPU "SYSTEM" line will report this setup as a "FASTROM" setup.

Suboptions are:

file Specifying a file with a valid ROM image will load that ROM image instead of the system's physical ROM image. The one restriction is that the disk-loaded ROM image must be the same ROM revision as the current system ROM. If they aren't the same revision, the KICKROM option can be used instead, but that'll require a reboot. This option can be used to load a patched version of the current ROM without reboot in most cases.

KEYPATCH n This will patch the keyboard scanning routine for machines that have Cherry keyboards (small function keys). The "n" parameter allows a variable delay between 1 and 100 to be specified; the delay depends on the keyboard, but should be pretty independent of CPU speed.

NOSTACK This will prevent the attempted translation of supe visor stack into 32 bit memory. By default, such translation will be done if the supervisor stack is found in Chip memory.

(b) CARDROM path

When used in conjunction with the FASTROM option, this allows ROMs from expansion cards to be located in fast memory as well. The path should reference a file containing lists of expansion cards that should be translated if found. It's necessary to read this from a user-defined file, rather than from the expansion environment itself, since an expansion device's ROM could be located close to that device's registers; there's no way for SetCPU to know it's safe to translate a card ROM image unless you tell it. On my system I read a file called CardROMList, which currently contains the single line:

0x202 0x01 0x10000 0x8000 0x4000 CBM_2090A_Disk_Controller

All the numbers given are in C language hex format. The parameters are, in order, the device's manufacturer code, product code, the device's size (in bytes), the ROM's offset from the configured board's base address (in bytes), and the size of the ROM area to be translated (in bytes). The final item is text string to identify the device; this'll be displayed by the VERBOSE option if the ROM translation does in fact take place. The "_" characters in the name will be translated to " " characters. Note that the CardROM translations are currently based on 16K chunks, and SetCPU

will ignore requests for translations of less than 16K, and round down to 16K boundaries for larger translation requests.

(c) HEAD

This option causes the SetCPU memory allocator to attempt memory allocation for its translated objects from the start of 32 bit memory instead from the end, as it usually does. Allocation from the end usually results in less fragmentation than from the start (due to the alignment restrictions of MMU objects), though this option is useful when dealing with merged memory lists. It is ignored when the ROM image and tables are in chip/$00C00000 memory.

(d) KICKROM path|dfN:

The KickROM option allows the system to be restarted with an alternate ROM image. This can be from a KickStart disk in a specified floppy drive, or from a given file name. If the ROM image is accessible, this command will cause the system to be immediately rebooted into the new OS. Note that pre-1.3 versions of the Amiga operating system will probably have some trouble with expansion cards, especially autoboot cards. For that reason there's the CONFIG 0 option, which is explained later.

The KICKROM command will reboot the machine with the new OS, but that ROM image will be physically located in either memory at $00C00000, if it's available, otherwise it'll use Chip memory. Once the new OS has started up, issuing either "SetCPU FASTROM ..." or "SetCPU KICKROM..." will cause that image to be moved into fast memory, and the slow memory will be given back to the system. The SetCPU "SYSTEM" line will report a 16 bit KICKROM image as a "SLOWKICK", and a 32 bit KICKROM image as a "FASTKICK". A machine running from a SLOWKICK kernel can't be re-KICKROMed, but can be from a FASTKICK kernel.

Suboptions are:

DELAY n This option sets the delay after reset before any code is run. The parameter may be set from 0 to 100, where 0 sets no delay. The default value is 10, which sets the delay value used in SetCPU V1.50. The need for a delay depends on the machine you're on. If your machine hangs after SetCPU loads KickStart, there's a real good chance that you need a longer delay.

KEEPEXEC Normally, KICKROM will clear the ExecBase pointer, to cause a the new version of the ROM to be rebuild from a cold boot condition. This option prevents that clearing action.

(e) CONFIG n

This option controls if and how expansion devices are recognized on a KICKROM boot. At the default configuration level, level 2, the expansion cards are left alone, allowing the new Kernel to try and configure them. Since some older operating system will choke on autoboot devices, this option will allow suppression of them for the rebooting process. When

requesting a KICKROM boot, a CONFIG level of 0 or 1 will prevent the devices from being recoginzed.

Once rebooted in the new OS, moving from a SLOW to a FAST Kick image, as described above, the CONFIG status will be honored. If the devices weren't suppressed, nothing special happens. If they were, they'll stay suppressed, and you very likely won't have the memory to support a FAST Kick image. Specifying a CONFIG level of 2 at this point will attempt to configure the devices without autobooting. At level 1, the devices will be made visable to the system again, but nothing will be done with them.

As of the latest release, CONFIG 0 appears to be required with the 1.2 operating system, at least if there's any autoboot device, even if you're attempting to move from a slow to fast kick image. The next release will attempt to allow 1.2 to configure non-autobooting devices at this point.

(f) TRAP n

This option controls the level of error trapping handled for you by the SetCPU system. The numeric parameter is actually optional for compatibility with SetCPU V1.4. If no TRAP is specified, the default level 2 is enacted. If the TRAP command is given without a parameter, trap level 0 will be setup.

Trap level 0 causes the MMU to look at all 32 bits of address; access to any memory outside of the 24 bit space will result in an exception, which if unhandled, results in a GURU 2. Trap level 1 will set up the MMU to only look at 24 bits of address space. Trap level 2 works like level 1, but additionally sets up a trap handler for the Bus Error exception (which usually surfaces as a GURU #2). For normal operation (eg, running other people's code), Trap level 2 is probably what you want. For final testing of your own code, levels 0 or 1 can catch things which would go unnoticed on a 68000 machine, such as writing to ROM space or out of the 24 bit address space.

The exception handler used for level 2 trapping catches things like writes to protected areas of memory. It just tells the bus machine not to complete the write, and signals no error. There's a slight chance that this won't be enough repair for a program doing something really outlandish – at that point, running at level 1 will let the GURU happen, which might help if you're debugging your own code. Other that that, there's probably nothing you can do to get such a program working with the MMU turned on, other than having it fixed. The other thing to consider is that this exception handler could conflict with another system-level handler installed by a GOMF-like program. That shouldn't cause a big problem, since you'll the one that was installed later, both of which presumably trap the error, but it's something to be aware of.

Under V1.3 and earlier releases, a DOS bug can cause invalid accesses, which cause the exception, when running the EndCLI or NewCLI/NewShell programs; running at level 1 or 2 will avoid gurus with these commands.

(g) BITS n

This option forces the MMU table for KICKROM or FASTROM to be built to support a specific number of bits, regardless of the actual bits apparently used by the system. Valid significant bits range from 24 through 32.

(h) ROMBOOT

This option forces a reset to physical ROM without hanging the system, even when the MMU is active.

(i) VERBOSE

This option more fully describes the system translations.

5. ROM FILE FORMATS

The 256K KickStart disk the KICKROM option will look for is the standard Commodore KickStart format, which is a standard format floppy with the work "KICK" at the start of the disk, followed by 512 blocks of 512 bytes each, a plain dump of the ROM image. 512K KickStart disks look just the same, only, of course, with 512K of ROM. The KickStart loader will actually check the first 32 blocks of disk for the start of ROM.

The size of the KickStart image is determined by the first longword of the file. Optionally, disk files can contain two extra longwords at the beginning of the file, the first being a $00000000, second the expected size of the ROM image. That expected size will be compared with the expected based on the first longword in the ROM image and the actual length of the loaded file, in the case of a disk file rather than a KickStart. The base address and starting address are determined from the ROM image, and SetCPU will attempt to use them. Some ROM images can cause a conflict with other system resources.

6. CREDITS

While this program is an entirely original work, nothing happens in vaccuum, this one included. I'd like to mention folks who, directly or indirectly, helped make this thing happen, by providing example MMU code, suggestions, and incentives. These folks include Neil Katin, Jez San, GVP Inc., Dale Luck, Bryce Nesbitt, Andy Finkel, and the other Commodore-Amiga software people, and the Commodore-Amiga Technical Support folks.

7. POTENTIAL BUGS AND OTHER NOTES

I should point out here that much of what SetCPU does is of a rather dubious nature. Everything that's possible to do correctly under the 2.0 release of the OS, including CPU/FPU identifications and cache control, is done via the approved 2.0 methods when running in 2.0, and via my own tricks when run under 1.3 or earlier releases.

While it's impossible for an application to correctly use the MMU under 1.3 or 2.0, SetCPU attempts to be intelligent about its use of the MMU. It will check for the use of the MMU by an agent other than SetCPU, and refuse to modify the current MMU setup if such an alien MMU setup is found. For systems with an unused MMU, SetCPU will do it's best to be safe about the modifications it makes to the memory map. There may be problems with this

program's MMU code on the Amiga 3000, but based on the SuperKickStart and CPU programs that are shipped with the first A3000s, this should not be of immediate concern, since the A3000 comes with equivalent functionality. Should an A3000-safe version of SetCPU eventually become needed, I'll make the attempt to track down any A3000 bugs I can find. Just because you help design a machine doesn't necessarily mean you have one to code and test on at home.

Finally, if you wish to contact me regarding bug reports, new releases, contributions of cash or macadamia nuts, or pretty much anything else, I can be reached at the below addresses.

-Dave Haynie

Logical Address:

PLINK: D-Dave H
bix: hazy
usenet: {uunet,rutgers}!cbmvax!daveh

Physical Address:

284 Memorial Drive
Gibbstown, NJ
08027

3.2.2 Memroutines

A few months ago, when I was doing some serious debugging, I accidentally discovered that the Lattice C functions memcpy, memcmp, and memset, as well as the Aztec C functions movmem and setmem, all deal with data one byte at a time. With a 68000 processor, this doesn't matter much, especially if you take advantage of Lattice C's capability to deal with these functions as "built-in". However, Amigas with 68020 and 68030 processors, and 32-bit wide memory, are starting to proliferate. With this sort of hardware, you can definitely improve performance by processing data a long word at a time, instead of a byte at a time.

The three functions in this package, memcpy(), memcmp(), and memset(), are "plug-compatible" replacements for the Lattice functions of the same name.

```
char *memcpy(),*memset(),*to,*from,*s1,*s2,*toaddr,ch;
long int memcmp(),count;
toaddr = memcpy(to, from, count);
```

from	=	pointer to data to be copied
to	=	pointer to area that data is to be copied to
count	=	number of bytes to be copied
toaddr	=	pointer to area that data was copied to

```
x = memcmp(s1, s2, count);
```

s1, s2	=	pointers to data to be compared
count	=	number of bytes to be compared
x	=	0 if s1 and s2 are identical; Contains a negative value if s1 $\prec$ s2, or a positive value if s1 $\succ$ s2

```
toaddr = memset(to, ch, count);
```

to	=	pointer to area that is to be set
ch	=	value to set area to
count	=	number of bytes to set
toaddr	=	pointer to area that was set

Note that with Aztec C, the "count" parameter must be "long".

Two libraries are provided, memorya.lib, the Aztec version (sorry, 3.6a), and memoryl.lib, the Lattice version. When linking, this file must be included before the normal library (c.lib for Aztec, lc.lib for Lattice). If you are using Lattice C, "#include string.h" statements should be removed or commented out.
Using these routines with Aztec C should never cause a performance penalty, even with a 68000 processor, unless you call them a disproportionately large number of times with parameters that are not word-aligned. With Lattice C, you may find a degradation in performance if most of your calls to the functions have a value of less than 8 for "count". This is because using the built-in versions of the functions means less overhead entering and exiting the functions.
This software is public-domain.

Robert Broughton 328-1027 Davie St. Vancouver, BC
V6E 4L2 Canada USENet: a1040@mindlink.UUCP

3.2.3 JoyLib

Bei dieser Software handelt es sich um eine sehr schnelle Routine für die Joystickabfrage. Sie sit so geschrieben, daß sie auf jedem 680x0 Prozessor läuft. Die Software liegt in zwei Versionen vor. Die erste ist eine Standart".o"-Routine für alle Sprachen, die in der Lage sind "_joy0/1" zu verwenden. Diese Routine ist kompatibel zum BLINK-Format. Der andere Teil ist eine Version, der Modula und anderen Sprachen zugedacht ist, die ".fd" zur Bildung eines Interfaces zu einer anderen Sprache benötigen.

```
JoyLib
######

Hello World! This is a new joystick routine with a special algorhythm
I'd call "Advanced Hufmann Word Shift Comparision" (just kidding).

There are two versions of the routines: 1) standard ".o" linkage for
any language able to call "_joy0/1" and compatible to BLINK format,
2) a shared library (thanks to BLINK) for such nifty things like Modula
and all... A .fd for building interfaces to *any* languae is provided.

The routines will return a single argument in D0, which bits represent
joystick directions:

    UP = 4

      LEFT = 1    RIGHT = 2

   DOWN = 8

   FIRE = 16

Simple enough, isn't it?

This stuff is Public Domain; use it, inflame it, molest it...

Olli

****->

Snail to: Oliver Wagner  Lansberge 5  4322 Sprockhoevel  West Germany
```

3.2.4 FPU-Mathtrans.Library V1.1

FPU-Mathtrans.Library V1.1 28.1.1990
(c) by Heiner Hückstädt

```
This library was written for all people, who own the Motorola
floating point unit MC 68881, MC 68882 or MC 68040 (haha) and
never use it, because most of the software, written on the
Amiga make use of the Fast Floating Point format, which is
absolutely undigestible for the numeric coprocessor. (It's a shame !!)
Only a few software developers spread a recompiled FPU version, too.

This library is to increase the performance of that software products,
who take much use of the FFP-transcendental functions (Sin, Cos, Pow, etc.)
like Distant Suns (very very good program, but where is the 68020/881
version) or others.

How to use this library
-----------------------

You must have a 68020/68030 - 68881/68882 Turboboard !
If so, copy this library to your LIBS: directory
The old mathtrans.library will be deleted

Realisation
-----------

The FPU MC68881/82 knows for every function
(expect SPTieee, SPFieee and SPPow) a hardware
instruction like fsin.x, fcox.x, flogn.x ,etc.
But all inputs for the mathtrans.library and the
mathffp.library come in this #$-?{\copyright}?? FFP-format,
so every input (and output, too) has to be converted
from FFP to IEEE single precision and back using SPTieee
and SPFieee. (Excuse me Commodore, but these functions were
stolen from your original mathtrans.library.)

Profit
------

On my configuration (68020 7Mhz, 68881 25Mhz, no 32 bit-ram)
my genius BenchMark shows a speed increase of 7.1 overall.
This may differ due to your money, megahertz and megabytes.
```

```
THIS PROGRAM IS IN THE PUBLIC DOMAIN

Permission is hereby granted to distribute this program,
source, and documentation for non-commercial purposes,
so long as the copyright notices are not removed.
This program may not be distributed for profit without the
permission of the author.

Comments, questions, donations, bugs due to this library may
come to
```


Heiner Hückstädt
Gellertstraße 12
D-5090 Leverkusen 1

That's all, hope you can use it

4 Lizenzvertrag

1. Urheberrecht:

   Das Programm und die damit verbundenen Unterlagen unterliegen dem Urheberrecht. Das Programm bzw. die Unterlagen oder jegliche Kopien dürfen nicht verwendet, kopiert, abgeändert oder übertragen werden, sofern nicht ausdrücklich in diesem Vertrag vorgesehen.

2. Lizenz

   Sie haben das einfache Recht, das beiliegende Programm lediglich auf einem einzelnen Computer zu verwenden. Das Programm darf materiell von einem Computer auf einen anderen übertragen werden, vorausgesetzt, daß das Programm jeweils nur auf einem Computer verwendet wird. Das Programm darf nicht elektronisch über ein Netz von einem Computer auf einen anderen übertragen werden. Es dürfen keine Kopien des Programms oder der Begleitunterlagen an andere verteilt werden. Das Programm bzw. die Unterlagen dürfen nicht geändert oder übersetzt werden.

3. Sicherungsunterlagen und Übertragung

   Sie dürfen lediglich eine (1) Kopie des Programms für Sicherungszwecke anfertigen. Die Urheberrechtanzeige muß reproduziert und der Sicherungskopie beigelegt werden. Das Produkt darf nur an eine andere Partei übertragen und konzessioniert werden, wenn die andere Partei den Bedingungen und Kozessionen dieses Vertrages zustimmt und einen Meldeschein ausfüllt und diesen an Harms Computer – Systeme zurücksendet. Wenn sie das Programm übertragen, müssen sie gleichzeitig die Unterlagen und die Sicherungskopie übertragen oder die Unterlagen Übertragen und die Sicherungskopie vernichten (löschen).

4. Bedingungen

   Diese Lizenz ist bis zu ihrer Beendigung wirksam. Sie können sie beendigen, indem Sie das Programm und die Unterlagen und sämtliche Kopien davon vernichten. Diese Lizenz läuft ebenfalls ab, wenn Sie irgendeiner Bedingung oder Kondition dieses Vertrages nicht nachkommen. Sie erklären sich einverstanen, bei einer solchen Beendigung sämtliche Kopien des Programms und der Unterlagen zu vernichten.

5. Haftungsausschluß für das Programm

   Das Programm wird "in gegenwärtigen Zustand" bereitgestellt ohne jegliche Garantie irgendeiner Art, sei es ausdrücklich oder stillschweigend inbegriffen, einschließlich, jedoch ohne Beschränkung darauf, der stillschweigend miteingeschlossenen Garantien der Marktfähigkeit und Eignung für einen bestimmten Zweck. Das gesamte Risiko bezüglich der Ergebnisse und der Leistung des Programms wird von Ihnen getragen, sollte sich das Programm als schadhaft erweisen, übernehmen Sie (und nicht Harms Computer–Systeme) die gesamten Kosten für sämtliche erforderlichen Wartungs-, Reparatur- oder Korrekturarbeiten. Weiterhin übernimmt Harms Computer–Systeme keinerlei Garantie, Gewährleistung bzw. gibt keinerlei Garantieerklärungen bezüglich der Anwendungsergebnisse des Programms hinsichtlich Richtigkeit, Genauigkeit, Zuverlässigkeit, Gültigkeit oder sonstigem; und Sie vertrauen auf das Programm und die Ergebnisse ausschließlich auf eigenes Risiko.

6. Beschränkte Gewährleistung für die Diskette

   Harms Computer–Systeme garantiert dem ursprünglichen Lizenznehmer lediglich für einen Zeitraum von neunzig (90) Tagen ab dem Datum des ursprünglichen Kaufes, daß die Diskette(n), auf der (denen) das Programm aufgenommen ist, keine Material- oder Verarbeitungsfehler aufweist (aufweisen). Falls ein durch diese Gewährleistung gedeckter Fehler während dieser Garantiezeit (90 Tagen) auftritt und Sie spätestens fünf (5) Tage nach Ablauf dieses Zeitraumes von 90 Tagen an Harms Computer–Systeme zurückgegeben wird, so wird Harms Computer–Systeme die Diskette entweder reparieren oder ersetzen. Diese Garantie steht an Stelle von sämtlichen sonstigen ausdrücklichen oder gesetzlichen Garantien, und einer jeglichen stillschweigend inbegriffenen Garantie, einschließlich, jedoch ohne Beschränkung darauf, der stillschweigend miteingeschlossenen Garantien der Marktfähigkeit und Eignung für einen bestimmten Zweck, wird hiermit auf den besagten Zeitraum von 90 Tagen eingeschränkt.

   Die Haftung von Harms Computer–Systeme beschränkt sich ausschließlich auf die Reparatur oder auf den Ersatz des Schadhaften Produktes in ihrem allgemeinen ermessen und umfaßt in keinem Falle Schadenersatz für Verwendungsverlust oder Verlust von erwarteten Gewinnen oder Vorteilen oder sonstigen Neben- oder Folgekosten, Ausgaben oder Schäden, einschließlich, ohne Beschränkung darauf, jedlicher Daten oder Informationen, eventuell verloren gehen oder falsch wiedergegeben werden, selbst wenn Harms Computer–Systeme von der Möglichkeit solcher Schäden in Kennntnis gesetzt worden ist.

Einige Länder lassen keine Beschränkung der Dauer einer stillschweigend miteingeschlossenen Garantie zu, so daß die obenstehende Beschränkung möglicherweise auf Sie nicht zutrifft. Einige Länder lassen keinen Ausschluß bzw. keine Beschränkung von Neben- oder Folgeschäden zu, so daß die obige Beschränkung bzw. der obige Ausschluß möglicherweise auf Sie nicht zutrifft. Diese Garantie gibt Ihnen bestimmte gesetzliche Rechte, und Sie haben möglicherweise sonstige Rechte, die von Land zu Land verschieden sind.

7. Dieser Lizenzvertrag unterliegt den Gesetzen der Bundesrepublik Deutschland.

Hardware Haftung und Garantie

Harms Computer–Systeme garantiert dem ursprünglichen Käufer für einen Zeitraum von sechs Monaten ab dem Datum des ursprünglichen Kaufes, daß das PROFESSIONAL – 3000 – BOARD keine Material- oder Verarbeitungsfehler aufweist. Falls ein durch diese Gewährleistung gedeckter Fehler während dieser Garantiezeit (sechs Monate) auftritt und Sie spätestens fünf (5) Tage nach Ablauf dieses Zeitraumes von 6 Monaten an Harms Computer–Systeme zurückgegeben wird, so wird Harms Computer–Systeme das PROFESSIONAL – 3000 – BOARD entweder reparieren oder ersetzen.

Diese Garantie steht an Stelle von sämtlichen sonstigen ausdrücklichen oder gesetzlichen Garantien und einer jeglichen stillschweigend inbegriffenen Garantie, einschließlich, jedoch ohne Beschränkung darauf, der stillschweigend miteingeschlossenen Garantien der Marktfähigkeit und Eignung für einen bestimmten Zweck, wird hiermit auf den besagten Zeitraum von sechs Monaten eingeschränkt.

Die Haftung von Harms Computer–Systeme beschränkt sich ausschließlich auf die Reparatur oder auf den Ersatz des schadhaften Produktes in ihrem allgemeinen ermessen und umfaßt in keinem Falle Schadenersatz für Verwendungsverlust oder Verlust von erwarteten Gewinnen oder Vorteilen oder sonstigen Neben- oder Folgekosten, Ausgaben oder Schäden, einschließlich, ohne Beschränkung darauf, jedlicher Daten oder Informationen, eventuell verloren gehen oder falsch wiedergegeben werden, selbst wenn Harms Computer–Systeme von der Möglichkeit solcher Schäden in Kennntnis gesetzt worden ist.

Einige Länder lassen keine Beschränkung der Dauer einer stillschweigend miteingeschlossenen Garantie zu, so daß die obenstehende Beschränkung möglicherweie auf Sie nicht zutrifft. Einige Länder lassen keinen Ausschluß bzw. keine Beschränkung von Neben- oder Folgeschäden zu, so daß die obige Beschränkung bzw. der obige Ausschluß möglicherweise auf Sie nicht zutrifft. Diese Garantie gibt Ihnen bestimmte gesetzliche Rechte, und Sie haben möglicherweise sonstige Rechte, die von Land zu Land verschieden sind.

Fehler, die durch unsachgemäßen Gebrauch oder falschen Einbau der Boards verursacht wurden, unterliegen nicht der Garantie.

Falls trotz unserer regelmäßigen Qualitätskontrollen Mängel an dem Produkt zu verzeichnen sind, sind diese schriftlich innerhalb 2 (zwei) Wochen bei uns anzuzeigen.

5 Fehlerbehandlung

5.1 Allgemeine Fehler

Fehler im direkten Zusammenhang mit dem PROFESSIONAL – 3000 – BOARD sind nicht bekannt.
Sollten trotzdem Fehler auftauchen sind sie meistens darauf zurückzuführen, daß das Programm spezielle Hardwareeigenschaften des MC68000 nutzt, die nicht mit dem MC68030 konform laufen. Diese Art der Programmierung wird gelegendlich bei Spielen angewandt und kann dort zu Fehlern führen. Weitere Ursachen können falsch angeforderter Speicher sein. Zum Beispiel wenn ein Programm 512 kb Chip-Memory erwartet und 1 Mb zur Verfügung stehen. Diese Fehler haben nichts mit dem PROFESSIONAL – 3000 – BOARD zu tun, sondern sind genereller Art. Fehler können in Verbindung mit Hardware von Fremdherstellern auftreten, die sich bei der Entwicklung nicht an die Richtlinien von Commodore gehalten haben. Falls Probleme auftauchen sollten, dann entfernen sie diese Hardware und probieren nur den Amiga mit dem PROFESSIONAL – 3000 – BOARD . Durch einzelnes Hinzufügen der Fremdprodukte läßt sich meist das Teil identifizieren, welches den Fehler verursacht. Das Wechseln der Hardware wird im ausgeschalteten Zustand vorgenommen. Die Garantiebedingungen der einzelnen Hersteller sind dabei zu beachten.

5.2 68030 Exception Error List

Vector Number(s)	Vector Offset Hex	Vector Offset Space	Assignment	*Status* Asserted
0	000	SP	Reset Initial Interrupt Stack Pointer	–
1	004	SP	Reset Initial Program Counter	–
2	008	SD	Bus Error	Yes
3	00C	SD	Adress Error	Yes
4	010	SD	Illigal Instruction	No
5	014	SD	Zero Divide	No
6	018	SD	CHK, CHK2 Instruction	No
7	01C	SD	cpTRAPcc, TRAPcc, TRAPV Instruction	No
8	020	SD	Privileg Violation	No
9	024	SD	Trace	Yes
10	028	SD	Line 1010 Emulator	No
11	02C	SD	Line 1111 Emulator	No
12	030	SD	(Unassigned, Reserved)	
13	034	SD	Coprocessor Protocol Violation	No
14	038	SD	Format Error	No
15	03C	SD	Uninitialized Interrupt	Yes
16 Through 23	040 – 05C	SD	Unassigned, Reserved	
24	060	SD	Spurious Inerrupt	Yes
25	064	SD	Level 1 Interrupt Autovector	Yes
26	068	SD	Level 2 Interrupt Autovector	Yes
27	06C	SD	Level 3 Interrupt Autovector	Yes
28	070	SD	Level 4 Interrupt Autovector	Yes
29	074	SD	Level 5 Interrupt Autovector	Yes
30	078	SD	Level 6 Interrupt Autovector	Yes
31	07C	SD	Level 7 Interrupt Autovector	Yes
32 Through 47	080 – 0BC	SD	TRAP #0-15 Instruction Vector	No – No
48	0C0	SD	FPCP Branch or Set on Unordered Condition	No
49	0C4	SD	FPCP Inexact Result	No
50	0C8	SD	FPCP Divded by Zero	No
51	0CC	SD	FPCP Underflow	No
52	0D0	SD	FPCP Operand Error	No
53	0D4	SD	FPCP Overflow	No
54	0D8	SD	FPCP Signaling NAN	No
55	0DC	SD	Unassigned, Reserved	No
56	0E0	SD	MMU Configuration Error	No
57	0E4	SD	Defined for MC68851 not used by MC68030	
58	0E8	SD	Defined for MC68851 not used by MC68030	
59 Through 63	0EC – 0FC	SD	Unassigned, Reserved	
64 Through 255	100 – 255	SD	User Defined Vectors (192)	

6 Technische Daten

Das PROFESSIONAL – 3000 – BOARD wird mit folgenden Daten, je nach Ausführung, ausgeliefert:

- 2, 4 MB 32-Bit Memory autokonfigurierend
  erweiterbar auf 8 bzw. 16 MB On-Board
- MC68030 30 MHz Taktfrequenz (optional 56 MHz)
- Prozessoren umschaltbar von MC68030 nach MC68000
- Co-Prozessor MC68882 oder MC68881 (optional bis 60 MHz) einsetzbar
- schnelles Memory voll 16/32 Bit DMA fähig und mit 100% I – D Caching
- Booten von MC 68030, MC 68000 oder AMIX softwaremäßig über Bootmenü
- Einstellbare Waitstates für variablen Takt
  Dynamische Buscycle Anpassung
- Separater Takt für Prozessor und Co-Prozessor möglich
- 100% MMU Unterstützung, jede Kickstartversion ladbar
- I – D DMA-Caching im Amiga Adressraum

Technische Änderungen vorbehalten.