AT BIOS KIT

The Comprehensive Guide to Creating an AT Bios in C

A Complete Description of the Bios Including Source Code on Diskette

John O. Foster & John P. Choisser

Annabooks

ISBN 0-929392-02-7

AT BIOS KIT

The Comprehensive Guide to Creating an AT Bios in C

John O. Foster & John P. Choisser

A Complete Description of the Bios Including Source Code on Diskette

Annabooks

AT BiosKit

by John O. Foster and John P. Choisser

PUBLISHED BY

Annabooks
12145 Alta Carmel Court, Suite 250-262
San Diego, CA. 92128

November 1989

Printed in the United States of America

ISBN 0-929392-02-7

Dedication

John O. Foster

This Book is dedicated especially to Patricia, and to the rest of the family for all their help and support.

John P. Choisser

This Book is dedicated to Coleen for her loving support during my career change.

Preface

What started as a diversion has turned into a gratifying nuisance. Publishing the XT-AT Handbook and the XT Bioskit was an exercise in "Is anyone interested?" The Handbook proved there were some people out there, and the XT BiosKit indicated there were a lot of people out there. "Do you publish an AT Bios Book?" was often asked. When we at Annabooks tallied up the interest, the next move was obvious. More and more system designers are moving from 8088 based systems to 286 and 386 based systems. And the fascinating thing we heard was the variety of uses to which the PC architecture was being put...

We heard from those who are involved in automation, robotics, research, education, avionics, consumer and office products, energy management and control, and industries and applications too numerous to mention, both in the U.S. and around the world. Product volumes involved ranged from 5 and 10 per year, to thousands per month.

We even heard that because of the BiosKit, projects that were designed for 286 based systems could be changed to 8088 based systems. Being able to use the BiosKit was important enough to justify changing the product.

For all you who asked for an AT BiosKit, we are happy to present this latest volume in an exciting series which will cover a variety of topics, always in the form of a "working" book.

John O. Foster
Hidden Meadows, California
November 1988

Table of Contents

<u>SECTION E</u> <u>*"THE C PROGRAMS"*</u>

<u>SECTION F "ODDS AND ENDS"</u>

NOTES

SECTION A

About the Bios

This section introduces the Bios, lists some reasons for creating your own customized Bios, explains Ownership and License information, and provides an Overview of the structure of the Bios.

ONE

Introduction

Unseen and unnoticed, the Bios (Basic input/output system) of a personal computer takes control when power is switched on, setting up the hardware devices, running tests, and booting up the operating system from disk. Without a Bios the machine is an unusable, non-operating collection of hardware components. Yet the Bios is a mysterious component for the majority of users. The Bios is a collection of programs stored on a read-only memory chip and is usually considered to be part of the hardware set, but these routines are like any other software. They can be analyzed, understood, modified and re-created as desired.

This Book

This book is a guide to analyzing and understanding what a Bios is, how it works, and how one goes about creating a Bios. The descriptions of the Bios functions are augmented with listings of the programs required to perform those functions. The auxiliary tools and utility programs which are used to create the Bios are also included with an explanation of their purpose and function.

What is the Bios?

Bios is an acronym for Basic input/output system. It is a set of programs that reside in non-volatile memory (often called firmware; programs that are implemented in hardware memory chip(s) and are not lost when power is turned off). The Bios includes the power-on diagnostics, a boot-up program, and device drivers for the keyboard, disk, display, serial ports, printer port, and miscellaneous functions. A Bios is usually written in assembly language, but the BiosKit is done in the C language wherever possible. Some low-level operations require assembly language for memory allocation reasons and some operations are done in assembly language for efficiency. But by providing the bulk of the BiosKit in C, it is much easier to understand, modify, and maintain the code.

Why do your own Bios?

You may want special features, such as special interface card support, SysVue capability, faster bootup, equipment status displays, and so forth. Some of these features are difficult to implement with a supplied Bios since you do not have the capability to modify it. Additionally, a standard Bios may not have the compatibility that you desire. You may also wish to upgrade your machine to newer capabilities that are included in later generations of compatible computers.

If you are a manufacturer or systems integrator, you may have found that your own Bios requires expensive long-term commitments to a Bios vendor that may not be able to respond to your needs in a timely manner. Once you have control of your Bios in your own hands, you will be able to modify and maintain your Bios at your own discretion.

If you are a manufacturer of systems for use in government or commercial applications which require delivery of source code for all software included in a system, BiosKit allows you to easily meet these requirements without cumbersome escrow agreements or additional payment to a Bios Supplier to obtain source code. By including BiosKit Documentation as part of your product, you may satisfy your project requirements. If your project requires special verification or certification activities, Bioskit not only gives you access to source code, it also allows you to configure it as you desire.

TWO

Audience

This book is intended for various groups of people:

* Those who have an unsatisfied curiosity about the internal workings of their personal computer.

* Those who are involved in the hardware and system design details of PC's and need to understand the workings of a Bios.

* Those who are creating special applications of PC systems and find that the typical PC Bios may have some shortcomings or omissions.

* Those who just want to make their own Bios so they can enjoy the satisfaction of having done it.

* And especially for those people that are in more than one just one of the above groups.

More Notes

THREE

About the Programs

This book and the magnetic media included with it provide you with the source code and explanations needed for you to customize your own Bios for an AT type compatible computer. The majority of the source code is in the C language with supporting routines in assembly language. C was chosen for a variety of reasons, including:

* It is well suited to systems level programs because it permits access to machine features.

* It is widely used, understood, and a de-facto standard among system level programmers.

* There are many publications available for reference, tutorials, as well as periodical publications for alternate information.

* It is supported by a variety of affordable compilers.

* Its architecture and structure encourage good programming practices which help to minimize the frustrating aspects of debugging and error-correction.

Because of some restrictive aspects of a Bios program, some of the programs are written in assembly language, as you might expect with systems-level C programs. This is required because:

* Certain locations in the completed program need to be fixed at absolute memory locations (the reset jump).

* Constants need to reside in read-only memory and be allocated to the code segment in fixed locations.

* Some operations such as block moves (used in CRT scrolling, for instance) are much faster than the C counterparts.

Things To Do

FOUR

Tools Required

Building and customizing the Bios requires some tools. Knowledge is the basic tool. Then you will need the physical tools; the hardware and software described below.

Knowledge Tools

Readers should have or be prepared to acquire some degree of skill in the following areas to comprehend and become proficient in the creation and enhancement of a Bios:

* operation of DOS
* operation of a text editor
* assembly language understanding
* understanding of the C language
* understanding of the AT architecture

There are many excellent publications available on these subjects in addition to the manuals that are supplied with the software tools listed below. By visiting your library and book store, and by reading computer periodicals, and especially by talking to computer-interested acquaintances, you will find ways to augment your knowledge. For those who may be apprehensive about tackling a Bios project, remember that knowledge is acquired by bits and pieces. Curiosity, patience, and a positive attitude will be your best ally.

Software Tools

To build the Bios you will need the following tools:

* C Compiler
  Recommended - Microsoft V 5.1 or later
  Other Compilers may require some modification of the Source Code.
* Macro-Assembler - Microsoft V 5.1 or later
* Linker - Microsoft (included with MASM or Compiler)
* Debug program (included with MASM)
* A Make Utility (included with MASM or Compiler)
* An Archive or backup utility is also recommended.

You may chose alternate tools, but they may require some modification of the source code to build and execute successfully.

Hardware Tools

Your development computer should be an XT/AT compatible with 640K of ram, a hard disk, and a printer. Either a color or monochrome monitor is sufficient.

To program Eproms for your computer you will need a Prom programmer. The Sunshine (brand name) programmer (approximately $125.00) is suggested.

To erase Proms, a Walling Datarase UVProm Eraser is suggested ($35 to $55).

Target Tools

The programs in this book were designed for use with an AT compatible hardware set. This means that the original AT or the truly compatible "clone/compatibles" will require little or no Bios modification to operate successfully. There are some areas of difference, however, that may affect your Bios details. Some of these are described below.

Prom/Rom chip types - Some high volume AT's use masked Rom chips for the Bios. These chips may or may not have the same electrical characteristics and pinout as the Prom chips you would normally use for your Bios. You should verify that the pinout of the original Bios chip you intend to replace is compatible with the chip you will be using. If it is not pin compatible, you may be able to replace it by using an appropriate adapter.

The next consideration is the addressing capability of the original Bios Socket(s) in your target system. Depending on the original design, the address mapping may be hard-wired or jumper (or decoding device) selectable to accommodate the Bios Proms you create. You should have the range of F0000-FFFFF available for your Bios. This will probably require two Proms, one for even bytes and one for odd bytes in your target system.

Turbo machines generally interpret a special combination of keystrokes to enter/leave the turbo mode. This decoding may be added to the keyboard handler (Bioskb.c) in the scan code conversion section of the hardware interrupt service routine. Turbo-AT's may use one of the spare output pins of the 8042 keyboard controller chip to change the CPU clock speed. To determine if this method is used, you may manually toggle these bits using Sysvue (Port In and Out commands) to see if this scheme is used to control the Turbo switching. If you include a Turbo switching feature and subsequently have floppy disk failures, the DMA clock speed is to be suspected. Some Turbo motherboards switch the DMA clock to high speed along with the CPU clock. It may be necessary to include an "Unturbo" and "Returbo" sequence in the floppy disk module (Biosdisk.c) to insure that the DMA is operated at the slow clock speed during disk transfers. To do this, save the state of the signal used to implement the speed switch upon entry to the Disk_io handler, and restore it to its previous condition upon exit. A Turbo machine may have a

"Turbo" mode display (typically an LED) to show when it is in turbo mode. This indicator is usually driven from the same speed switch signal that controls the clock.

One method of investigating compatibility is to attempt to run the standard version of DOS on a computer. If a given computer operates only with its customized version of DOS, some hardware incompatibility might be expected.

SysVue

Bonus - BiosKit includes the source code for SysVue, a Prom resident program that provides many of the features of disk resident Debug programs. SysVue is always available with a keystroke sequence. It is extremely handy for checking custom features of your Bios.

Once you have successfully built the standard version of the BiosKit, you will be ready to consider special versions for your particular application.

Things I Should Do Soon

FIVE

Ownership, License, and Warranty

The following paragraphs explain the ownership (right of title) of the Source Code and of the Binary Code generated from it, the license granted to you to duplicate the object code, and the warranty of this published information. This information is important to you. Please read it carefully.

Ownership of Source Code

The Source Code for BiosKit is an intellectual work covered under the copyright laws of the United States and other countries. Title to ownership is retained by FOSCO. Your purchase of the BiosKit includes a license to use the Source Code, but does not convey right of title of said Source Code to you. You may use the Source Code in accordance with the License Agreement as described in the following paragraph.

License to Duplicate

With the purchase of the BiosKit, FOSCO and Annabooks have waived the $4.00 per copy license fee for the first ten (10) copies of the BiosKit Binary code for your own use, or for sale in your product.

You are responsible for remitting to Annabooks the fee of $4.00 per copy for additional copies. This fee remittance should be made payable to Annabooks and annotated as "AT Bioskit License Fee". It should be made on not less than a quarter-year basis and must be payable in U.S. (US$) funds.

Payments should be sent to the following address:
Annabooks
12145 Alta Carmel Court, Suite 250-262
San Diego, CA. 92128

Please abide by the license fee requirements to avoid the need for legal recourse on the part of Annabooks and FOSCO to insure compliance with these requirements.

During the build/customization phases, you may program an unlimited number of Proms until you have completed the Bios. Once you place the Proms in a computer used to run programs, the Proms are included in the ten (10) quantity covered by this license to duplicate.

Any Bios built from the BiosKit must include the FOSCO copyright information included with the original source code. You may add additional information, but you may not delete any copyright information included in the standard BiosKit.

DISCLAIMER AND LIMITED WARRANTY

SIX

Overview

This chapter is intended to acquaint you with the major functions and parts of the Bios. To do this, we will follow the flow of the Bios from the power-on hardware reset point and summarize what happens as the computer is turned on. Descriptions of detailed operations will follow in later sections and be correlated with the Source Code Listings.

Power On -

The 80286 processors start operation at segment:offset address FFFFF:0000. This means that program execution begins at the last paragraph (16 bytes) at the top of address space. This location normally will have a far jump to a lower address where the Bios will start to initialize and check-out hardware features of the system.

Diagnostics -

This usually starts at a label called 'reset' and includes checking of the CPU chip registers, tests and initialization of the peripheral chips, and ends up by invoking a bootstrap routine.

Bootstrap -

The function of the bootstrap routine is to load DOS from a floppy disk. If you have a hard disk in the system, a failure to boot from a floppy is usually followed by an attempt to boot from the hard disk. If no disk is available, you may wish to default to SysVue or some other Prom resident program you may elect.

Drivers -

The Bios contains the low-level drivers for a variety of devices. DOS normally assumes these drivers are present in the Bios and makes calls to them to perform peripheral access. The drivers correspond to the Bios Interrupts documented in various publications about PC's and DOS. To fully support DOS, these drivers must be included, and their devices initialized by the Bios.

Bios Structure

This description of the Bios structure is intended to clarify some of the characteristics of the Bios.

The 80286 processor starts executing code (from a power-up reset) at the highest paragraph in the memory map. This is at Segment:Offset FFFFF:0000. To properly respond to the hardware reset, valid executable code must be stored at this location in the Bios Prom. To insure that the instruction intended for this location is assembled and linked correctly, the final module of the Bios is manipulated in such a way that the normal linker utilities (such as Microsoft's Linker which links programs in an ascending address sequence), can achieve this result. The top module of the Bios is written in Assembly language, so that programmer control is maintained over the location of certain instructions. Additionally, the top module provides a place to include various table structures and assembly language subroutines, which are C functions.

From the power-on reset, a jump is executed (at FFFFF:0000) to an assembly language initialization code sequence which initializes the DMA controller, the Ram Refresh, and the first 64 Kbytes of System Ram. Once this has been accomplished, the C language POD (Power On Diagnostic) module is executed. This module continues the power-up and diagnostic checks of the system, and also calls setup (initialization) routines in the various device driver modules. When the system has been completely initialized, the bootstrap function is called.

The POD process includes:

loading the interrupt vector locations,
initializing the display adapter,
determining the size of the system Ram,
testing the system Ram,
reading the equipment configuration CMOS Ram,
performing a checksum test on the Bios prom,
identifying and initializing COM and LPT ports,
calling any executable Rom option modules,
initializing the floppy disk drive(s),
and finally calling the bootstrap procedure.

Why Link is Done Twice

During the linking process when building the Bios, a trial link is performed. This trial link determines how far short the Bios falls of filling a complete segment (64 Kbytes). This shortfall in length is calculated, and a filler module is built which will compensate for the shortfall. The Bios is then linked again with the filler module to produce a Bios in which the required locations are fixed at the correct addresses. This file is then converted from an .EXE format file to a .BIN binary image file suitable for transfer to a Prom programmer.

How to develop and test a Bios

The simplest way, of course, is to build a Bios, program a Prom, install the Prom in the system, and see how the system runs. This method is time consuming and debugging can be difficult, particularly if the system doesn't run at all. The preferred method is to install a Static Ram Adapter Card residing above the video ram area, and away from any rom-scan devices such as EGA Video and Hard Disk Adapters that might be installed in your target system. Recommended location is segment D000. The Test Bios is then loaded into the Ram area and a Jump instruction is executed to start executing the Test Bios. The Load-and-Jump can be performed using a Debug Batch file as shown in Section F. To effectively integrate the Bios, you should have (or should add) a pushbutton reset switch to your target system. This allows you to easily reset and restart your system with its prom-based Bios if your Test Bios should lock up or run away due to modifications which you are testing.

An additional aid to testing is to add debug mode display statements to track the activity of the Bios. These may be included in your C modules by using the write_string("text") statement. Hexadecimal byte and word values may be dumped using the lbyte(value) and lword(value) statements. By referring to the BiosVue.c module, you may examine examples of the usage of these statements.

When you have verified the operation of your modified Bios, the debug-mode statements may be removed, and the Bios rebuilt.

Driver Structure

By examining the structure of the Driver modules, you will see that they are essentially self-contained. They may call common assembly language functions in BiosTop.asm and common C functions in BiosMisc.c, but they seldom will call other modules except through the Interrupt structure. This helps to isolate interaction and serves to make testing easier.

Drivers have the general structure of a Setup routine followed by the Driver itself. Some drivers may also have a service routine for its associated hardware interrupt. The Setup routines are called from BiosPod to initialize the drivers during the Power-On-Diagnostics.

Programming Style

As you work with the source code, you will begin to apply your particular style of programming to re-writes, additions, and changes. This is normal and it is a sign of your increasing understanding of the programs. Don't be afraid to analyze the program structure and experiment. There are at least as many ways to code a function as there are programmers! Since this book deals with each topic only once, we had to present each example in a particular manner. You have a different perspective and style, and you will probably see it in a slightly different manner. Remember there are many right ways to code a function. It is sometimes difficult to find one of those right ways; it is usually easier to find one of the wrong ways!

Things I Should Avoid Doing

SECTION B

The C Utility Programs

These programs are used during the building process to add the date-stamp, calculate the checksum, install the CPU type identification, and to fix some locations at absolute addresses.

ONE

The Biossum File

The Biossum.c file is used during the Bios Build process to calculate the additive checksum of the Bios and to insert the checksum value into the last byte of the Bios. It looks for the first non-FF location and treats it as the start of the block. If the Bios is padded with FF's at the front of the file, the program will scan forward every 2K bytes to look for the start. This allows you to reserve space in 2 Kbyte blocks at the front for your own special programs. The Rom-Scan routine will determine where the Bios starts and scan up to that point.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: Bios checksum calculator
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 1-5-89
*
* Filename: biossum.c
*
* Functional Description:
*
*  This program calculates a checksum and inserts it into the last byte location of the Bios.
*
* Arguments:
*  Filename
*
* Return:
*  None
*
* Version History:
*
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include <stdio.h>

/* F U N C T I O N   P R O T O T Y P E S */

/* G L O B A L   V A R I A B L E S */

unsigned numread,i;
char sum,buffer[32768];
FILE *stream;

/* G L O B A L   C O N S T A N T S */

/* L O C A L   D E F I N I T I O N S */

/* L O C A L   C O N S T A N T S */

/* P R O G R A M */

main(argc,argv)
int argc;
char *argv[];
{
 if (argc <2)
 {
  printf("Not enough parameters on command line\n");
  exit(0);
 }
 if ((stream = fopen(argv[1],"r+b")) == NULL) printf("Could not open file for reading\n");
 else
 {
  sum=0;
  numread = fread((char *)buffer,sizeof(char),32768,stream);
  /*
  ** Scan for the first non -ff- cell at 2k boundaries to mark the start of the bios code.
  ** Reserve the prior cells for rom module integration later when customizing the bios prom
  ** with extensions.
  */

  for (i=0;(i < numread) && (buffer[i] == 0xff); i += 2048);

  /* i is advanced to beginning of actual bios code */
  for (;i<numread;i++) sum += buffer[i];


  numread = fread((char *)buffer,sizeof(char),32767,stream);
  for (i=0;i<numread;i++) sum += buffer[i];
  sum = 0 -sum;
  buffer[i] = sum;
  fseek(stream,-1L,SEEK_END);
  fwrite(&sum,1,1,stream);
 }
 fclose(stream);
}
```

TWO

The Biosdate File

The Biosdate.c file is the program which inserts the current date value as the date-stamp in the Bios module. This date is determined from the current date in your development machine. It will obviously affect the overall checksum of the Bios module, if you re-build on different days.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: bios date routine
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 1-05-89
*
* Filename: Biosdate.c
*
* Functional Description:
*
*  Insert date string into bios binary file. This program inserts the current date information
*  into the date field in the Bios at location FFF5
*
*
* Arguments:
*  Filename
*
* Return:
*  Writes date field in bios.bin file
*
* Version History:
* 1-5-89 font revision
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include <stdio.h>

/* F U N C T I O N   P R O T O T Y P E S */

/* G L O B A L   V A R I A B L E S */

FILE *stream;
char date_buffer[9];

/* G L O B A L   C O N S T A N T S */

/* L O C A L   D E F I N I T I O N S */

/* P R O G R A M */

main(argc,argv)
int argc;
char *argv[];
{
 if (argc <2)
 {
  printf("Not enough parameters on command line\n");
  exit(0);
 }
 if ((stream = fopen(argv[1],"r+b")) == NULL) printf("Could not open file for reading\n");
 else
 {
  fseek(stream,-11,SEEK_END);     /* find date field */
  _strdate(date_buffer);
  fwrite((char *)date_buffer,sizeof(char),8,stream);
 }
 fclose(stream);
}
```

Things Someone Else Should Do

THREE

The Biostype File

The Biostype.c file is used to insert a byte identifying the type of system in which the Bios is installed. The type identification may be used by some applications (or the Bios if so desired) to determine operational characteristics of the system hardware and/or software. This byte is located at the next- to-last location of the Bios. In an AT Bios, there is also another identification function; Interrupt 15, Function code C0, which returns system configuration parameters. This parameter block provides additional type information.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: Bios type
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 5-5-88
*
* Filename: biostype.c
*
* Functional Description:
*
*  Insert CPU Type byte into bios binary file. This program inserts a machine identification byte
*  at location FFFE in the Bios. Because of an Assembler quirk, if we try to define this in the
*  Source file, the assembler increments the offset counter and believes that the segment becomes
*  larger than 64K bytes, which it considers an error condition.
*
*
* Arguments:
*  biostype filespec [type]
*   where type is pc,xt,at,jr,xt640
*   if type is omitted, then default is xt
*
* Return:
*  Writes type byte in bios.bin file 'filespec'
*
* Version History:
*
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include <stdio.h>
#include <string.h>

/* F U N C T I O N   P R O T O T Y P E S */

/* G L O B A L   V A R I A B L E S */

FILE *stream;
char type_buffer[2];

/* G L O B A L   C O N S T A N T S */

/* L O C A L   D E F I N I T I O N S */

#define at 0xfc  /* these are the standard defined types */
#define xt 0xfe
#define pc 0xff
#define jr 0xfd
#define xt640 0xfb

/* P R O G R A M */

main(argc,argv) /* bios.bin */
int argc;
char *argv[];
{
 char cpu_type = xt; /* default to XT type byte */
 if (argc <2)
 {
  printf("Not enough parameters on command line\n");
  exit(0);
 }
 if (argc > 2)
 {
  if (strcmpi(argv[2],"at") == 0) cpu_type = at;
  if (strcmpi(argv[2],"xt") == 0) cpu_type = xt;
  if (strcmpi(argv[2],"pc") == 0) cpu_type = pc;
  if (strcmpi(argv[2],"jr") == 0) cpu_type = jr;
  if (strcmpi(argv[2],"xt640") == 0) cpu_type = xt640;
 }
 if ((stream = fopen(argv[1],"r+b")) == NULL) printf("Could not open file for reading\n");
 else
 {
  fseek(stream,-2,SEEK_END);     /* find type field */
  type_buffer[0] = cpu_type;
  fwrite((char *)type_buffer,sizeof(char),1,stream);
 }
 fclose(stream);
}
```

FOUR

The Biosgen File

The Biosgen.c file is used during the build process to sense the size of the Bios code and calculate a value which will be used to upward justify the Biostop module so that it resides in the highest portion of the Bios segment. This is required to insure that when a power-on hardware reset occurs there is an instruction present to be executed.

```
/****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: Bios filler generator
*
* Version: 2.00
*
* Author: FOSCO
*
* Date: 12-01-88
*
* Filename: biosgen.c
*
* Functional Description:
*
*  This program reads the map file (bios.map) and calculates how far short it falls from ending at
*  the top of the segment. Then it builds a biosfill.inc file that pads out enough to complete the
*  segment (64k bytes). The .inc file is used in a second linking of the .obj files. This produces
*  a full 64k byte file with the power-on reset jumps fixed at offset FFF0 in the binary image file.
*
* Arguments:
*  biosgen bios.map biosfill.inc
*
* Return:
*  Creates a biosfill.inc file
*
* Version History: 2.00 9/12/88
*  The method of determining the file length was changed to operate with 5.1 tools. This program
*  now looks at the .MAP and searches for the ORG_E000 label. If it finds the label, it looks at
*  the offset value associated with it, and calculates the fill value from that. If it does not
*  find the label, it prints an Abort Message. If the .map file shows that the Bios is too large,
*  it prints a message indicating that the Bios will not fit into the segment.
*
* Notes:
*  This file normally compiles with two Warning Messages. They may be disabled by changing the
*  message level switch in the make file. The program will execute OK even with the warnings.
*
****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include <stdio.h>
#include <string.h>

/* F U N C T I O N   P R O T O T Y P E S */

unsigned int hval(unsigned char);

/* G L O B A L   V A R I A B L E S */

FILE *stream;
int radix = 10;
char first[20],second[20],third[20];
char *p;

unsigned int i;
unsigned int diff;
unsigned int find; /* find/no find flag */
unsigned int find_string;
unsigned long int numread;
unsigned char line[255], *result;

/* G L O B A L   C O N S T A N T S */

/* L O C A L   D E F I N I T I O N S */

/* P R O G R A M */
```

```
main(argc,argv) /* sourcefile.map,destfile.inc */
int argc;
char *argv[];
{

 /* this is the label we will search for */
 char *estring = "_ORG_E000";

 /* this is where we move the offset value of the label*/

 char *vstring = "0000";

 if (argc <3)
 {
  printf("Not enough parameters on command line\n");
  exit(0);
 }
 if ((stream = fopen(argv[1],"r+b")) == NULL) printf("Could not open file for reading\n");
 else
 {
  find = 0;
  /* check each line of the .map file for the wanted label*/
  do
  {
   if((result = fgets(line,255,stream)) != NULL);
   {
    /* check for the label */
    if (strstr(result,estring) != NULL)
    {
     find_string = 0xe000;
     strncpy(vstring,&line[6],4);
     diff = 0;
     for(i=0;i<4;i++)
     {
      diff = (diff << 4) + hval(vstring[i]);
     }
     find = 1;
    }
   }
  }
  while ((result != NULL) && (find == 0));

  fclose(stream);

  if (find == 0) /* no find */
  {
   printf("Could not find ORG_F000 label in .MAP file\n\r");
   printf("Aborting operation.\n\r");
   printf("  1. Check command line argument specifies .MAP file\n\r");
   printf("  2. Check .MAP file for proper labels.\n\r");
  }
  else
  {
   printf("  File size  = %5u\n\r",diff+(0x10000-find_string));
   if (diff >= find_string)
   {
    printf(".EXE file is to large to process - aborting fill generation\n\r");
   }
   else
   {
    numread = find_string - diff;     /* num to write */

    printf("  Free Space = %5u\n\r",numread);

    /* now build the text line for the biosfill.inc file */
    strcpy(first,"       db      ");
    strcpy(second,ultoa(numread,second,radix));
    strcpy(third,"  dup(0ffh)");
    strcat(first,second);
    strcat(first,third);

    if ((stream = fopen(argv[2],"w")) == NULL) printf("Could not open file for writing\n");
    else
    {
     fwrite((char *)first,sizeof(char),strlen(first),stream);
    }
   }
   fclose(stream);
  }
 }
}

/* convert hex ascii to val */

unsigned hval(unsigned char cval) {
 unsigned val;
 cval -= '0';
 if (cval > 9) cval -= 7;
 val = cval;
 return(val);
}
```

Things I keep Forgetting

SECTION C

The Build Programs

These programs are used to assemble, compile and link the Bios modules. The process is automated to permit a complete build with just a few keystrokes.

ONE

The Make File

The Make File is the top level file used in building a Bios. It includes all the operations needed to compile and assemble the source files, to link them into a code module, and to convert the code module into a binary image file for Prom programming. This file will also build the utility programs needed for the build process. Using the MAKE utility simplifies the operation and allows the building of the Bios with just a few keystrokes.

```
#*****************************************************************************************************
#
# Copyright (c) 1988, 1989 FOSCO - All Rights Reserved
#
# Module Name: atbios.make
#
# Version: 1.00
#
# Author: FOSCO
#
# Date: 1-5-89
#
# Filename: at
#
# Functional Description:
#
#        This is the Make file to build the AT Bios
#        To Build the Bios, type 'make bios' <enter>
#
# Version History:
#
#
#*****************************************************************************************************


#--- inference rules - see description of MAKE utility ---

.asm.obj: atprfx.inc
 masm $*/W1/ZI/z;

.obj.exe:
 link $*;

.c.obj: atkit.h
 cl  /Zp1 /J /W0 /G2 /c /Zl /Zi /AS /Ox $*.c

#------------- list of files to build ------------------

#--- These are utility routines used to build the Bios ----


                    # routine to fill up to top module

biosgen.exe: $*.c
 cl /W0 /Ox /Zi $*.c /link /Co

                    # routine to place date in binary file

biosdate.exe: $*.c
 cl /W0 /Ox /Zi $*.c /link /Co

                    # routine to place type byte in binary file

biostype.exe: $*.c
 cl /W0 /Ox /Zi $*.c /link /Co

                    # routine to checksum binary file

biossum.exe: $*.c
 cl /W0 /Ox /Zi $*.c /link /Co


                    # routine to display parenthesized text

parens.exe: $*.c
```

```
 cl /WO /Ox /Zi $*.c /link /Co

                    # routine to indent according to {}

indent.exe: $*.c
 cl /WO /Ox /Zi $*.c /link /Co

                    # routine to display comments

hilite.exe: $*.c
 cl /WO /Ox /Zi $*.c /link /Co

                    # routine to split out even/odd bytes

splitbin.exe: $*.c
 cl /WO /Ox /Zi $*.c /link /Co

                    # routine to merge in even/odd bytes (for test purposes)

mergebin.exe: $*.c
 cl /WO /Ox /Zi $*.c /link /Co

                    # binary image to Intel hexadecimal filter

bin2hex.exe: $*.asm
 masm $*;
 link $*;

          #--------- These are the Bios Modules --------------

                    # routine to pad the front of the Bios

atpad.asm:

atpad.obj:

                    # the data declaration module for biosvue

atdata.asm:

atdata.obj:


                    # print screen driver

atprsc.obj:

                    # memory size driver

atmem.obj:

                    # misc drivers in 'C'

atmisc.obj:


                    # bootstrap routine

atboot.obj:

                    # sys vue

atvue.obj:

                    # comm driver

atcomm.obj:

                    # keyboard driver

atkb.obj:

                    # floppy disk driver

atdisk.obj:

                    # hard disk driver

athard.obj:

                    # lpt driver

atlpt.obj:

                    # crt driver
```

```
atcrt.obj:
                    # timer driver
attmr.obj:
                    # time of day driver
attod.obj:
                    # equipment driver
atequi.obj:
                    # cassette driver
atcass.obj:
                    # power on diagnostics
atpod.obj:
                    # virtual mode driver
atvirt.asm:
atvirt.obj:
                    # absolutized high location module
attop.asm:
attop.obj:

          #---------- This is the linking process -----------

                    # make binary file of bios

at.bin: \
          atdisk.obj atlink     atkb.obj   atvue.obj  \
          atcomm.obj atlpt.obj  atcrt.obj atcut.inp   \
          atpod.obj  atpad.obj  atdata.obj atvirt.obj \
          atboot.obj atcass.obj atequi.obj attmr.obj  \
          atmisc.obj attop.obj  at         \
          attod.obj  atprsc.obj atmem.obj  athard.obj \
          biostype.exe biosdate.exe biossum.exe biosgen.exe
 copy atblank.inc atfill.inc
 masm atfill;
 link @atlink
 biosgen at.map atfill.inc
 masm atfill;
 link /Co @atlink
 debug <atcut.inp
 biosdate at.bin
 biostype at.bin at
 biossum at.bin
 splitbin at
```

Things I Forgot

TWO

The Link File

The Link File is an indirect file used by the linker to convert the group of bios---.obj files into an executable (.EXE) file module. The EXE module will then be converted to a binary image file for loading a prom programmer.

```
atpad+
atdata+
atpod+
atboot+
atcomm+
atkb+
atdisk+
athard+
atlpt+
atcrt+
atequi+
atmem+
attmr+
attod+
atprsc+
atcass+
atmisc+
atvue+
atvirt+
atfill+
attop,
at,
at/m;
```

Things I Remember

THREE

The Cut.inp File

The Cut.inp is used after the second link to transform the .EXE file, which is output by the linker, to a .BIN (binary image file). This type of operation is usually done by using the DOS Utility "EXE2BIN", but "EXE2BIN" cannot handle a full segment .exe file.

```
*************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: Cut of bios after second linking
*
* Version: 2.00
*
* Author: FOSCO
*
* Date: 1-5-89
*
* Filename: atcut.inp
*
* Functional Description:
*
*         This input file is used to perform an exe -> bin load of the bios.exe file. Because it
*         is a 64K file, the traditional exe2bin program overflows. This file directs that a
*         64 Kbyte output file (bios.bin) be written. This output file will be the binary image
*         used for prom programming.
*
* Arguments:
*
* Return:
*
* Version History:
*
*************************************************************************************************

nat.exe
l
rbx
1
rcx
0
nat.bin
w cs:0
q
```

Things Someone Should Tell Me

SECTION D

The ASM Programs

These programs are used to include assembly language functions, tables, virtual memory support, and initial start-up code for the Bios.

ONE

The Prfx.inc File

The Prfx.inc is the include file used for the assembly (.ASM) modules. It specifies the memory model, macro definitions, segment loading order for the linker, and required absolute offset locations for various data and code.

```
         page     55,132
;*****************************************************************************************************
;
; Copyright (c) FOSCO 1988 - All Rights Reserved
;
;
; Module Name: AT Bios assembly module prefix
;
; Version: 2.01
;
; Author: FOSCO
;
; Date: 01-01-89
;
; Filename: biosprfx.inc
;
; Functional Description:
;
;  This include file is used for the standard definitions for assembly language modules of the bios.
;
;  It is used to define the segment names, the segment loading sequence, assorted macros, and
;  location assignments.
;
; Version History:
; 2.01 - Font revision
;
;*****************************************************************************************************

         .286p
         .seq

_text    segment   word public 'code'
_text    ends

_data    segment word public 'data'
_data    ends

const  segment word public 'const'
const    ends

_bss     segment word public 'bss'
_bss     ends

_fill    segment word public 'fill'
_fill    ends

_stack   segment word stack 'stack'
_stack   ends

_atext segment word public 'acode'
_atext ends

         .model small,C

dgroup group _text,_data,const,_bss,_fill,_atext
         assume   cs:dgroup,ds:dgroup

         .xlist

ok       equu     0
error    equ      -1


;        macros for Bios generation

delay    macro
         jmp      $+2
         jmp      $+2
         endm
```

```
send_eoi  macro
          push      ax
          mov       al,20h
          out       20h,al
          delay
          out       0a0h,al
          pop       ax
          endm

write_cmos          macro address,value
          push      value
          push      address
          call      outcmos
          add       sp,4
          endm

read_cmos macro     address
          push      address
          call      incmos
          add       sp,2
          endm

true      equ       -1
false     equ       0

;---- stack swappers for interrupt routines ------

int_header          macro     dest
          push      offset dest
          jmp       interrupt_shell
          endm

; this 8088 version emulates the push immediate opcode
int_header_88       macro     dest                ; this is the 8088 version
          push      ax                            ; this saves ax on the stack
          push      bp                            ; save bp while we access the stack
          mov       bp,sp                         ; use bp for a pointer
          mov       ax,offset dest                ; ax = dest
          xchg      ax,[bp+2]           ; restore ax and put the dest on the stack
          pop       bx                            ; restore bp, leaving the dest
          jmp       interrupt_shell
          endm

;-------- required by MS C Compiler ---------
; This definition is required by the C Compiler.
; The value was determined by looking the standard C libraries.
_acrtused equ       9876h

          ; Bios Equates

dma                 equ       00                  ; dma channel 0 addr reg port addr

pit_port_0          equ       040h
pit_port_1          equ       041h
pit_port_2          equ       042h
pit_port_cmnd       equ       043h

pio_port_a          equ       060h                ; 8255 port a addr
pio_port_b          equ       061h                ; 8255 port b addr
pio_port_c          equ       062h                ; 8255 port c addr
pio_port_cmnd       equ       063h

segment_00          segment at 0
segment_00          ends

          ;comment /* rom bios data area */

segment_40                    segment at 40h
comm_list dw        4 dup(?)
lpt_list dw         3 dup(?)

          org       67h
voffset             dw        ?
vsegment  dw        ?
vflag               db        ?

          org       078h
lpt_timeout_list db 4 dup(?)
comm_timeout_list db 4 dup(?)
segment_40  ends

;--------- macros used to indicate org conflicts ------

free=0
slop=0
```

```
romorg    macro     arg
          ifdef     temp
           temp=$
           if1
            if      ($ le (arg-0e000h)
             @out   <<-          >>,(arg-0e000h)-$,<<               bytes free>>
             free=free+arg-$
            else
             @out   <<-                    >>,$-(arg-0e000h),<<         bytes TOO LONG !!!!!>>
          %out
             slop=slop+$-(arg-0e000h)
            endif
           endif
          else
           temp=$
          endif
          if1
           @out     <<>>,arg,<<                              ORIGIN OF &arg>>
          endif
          org       arg-0e000h
          endm

@out      macro     arg1,arg2,arg3
          .radix 16
          @@out     arg1,%(arg2),arg3
          .radix    10
          endm

@@out     macro     arg1,arg2,arg3
          %out      arg1 arg2 arg3
          endm

;-------------------------------------------------------

;         Rom entry points
; Unfortunately, these entry points need to be enforced since many users assume these to be fixed.
; Even though it violates the PC rules, we have to live with it.
;
          public    _print_screen,_keyboard_io,_keyboard_isr,_disk_io,_disk_isr
          public    _video_io,_cassette_io,_printer_io,_time_of_day,_mem_size
          public    _equipment,_comm_io,_timer_int

_copyright          equ       0e000h
_reset              equ       0e05bh
_nmi                equ       0e2c3h
_boot               equ       0e6f2h
_comm_table         equ       0e729h
_comm_io  equ       0e739h
_keyboard_io        equ       0e82eh
_keyboard_isr       equ       0e987h
_disk_io  equ       0ec59h
_disk_isr equ       0ef57h
_disk_parms         equ       0efc7h
_printer_io         equ       0efd2h
_video_io equ       0f065h
_video_parms        equ       0f0a4h
_mem_size equ       0f841h
_equipment          equ       0f84dh
_cassette_io        equ       0f859h
_video_font         equ       0fa6eh
_time_of_day        equ       0fe6eh
_timer_int          equ       0fea5h
_vector             equ       0fef3h
_dummy_iret         equ       0ff53h
_print_screen       equ       0ff54h
_hard_reset         equ       0fff0h    ; power on start
_date_stamp         equ       0fff5h    ; date stamp of bios
_hardware_id        equ       0fffeh    ; hardware ID byte

          .list

          .data

          .code
```

Things I Should Tell Someone

TWO

The Top File

The Top.asm File is an assembly language file which is linked so as to reside in the top portion of the Bios segment. It is the module which responds to the power-on hardware reset of the system. This module is where the execution of code starts, and is used to reset critical peripheral functions and start Ram Refresh. A minimum block of Ram is assigned to allow the use of a stack and then control is transferred to the POD (Power On Diagnostics) routine. This module also contains the assembly language support functions used by the Bios. The Bios ID and Copyright ASCII text information, the CPU type byte, the Bios date-stamp, and the Bios checksum byte are also in this module.

```
;*****************************************************************************************************
;
; Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
;
; Module Name: AT Bios top of the segment assembly module
;
; Version: 1.02
;
; Author: FOSCO
;
; Date: 10-18-89
;
; Filename: attop.asm
;
; Language MS MASM 5.1
;
; Functional Description:
;
;       This module serves these purposes;
;       1. It provides the power-on jump at location FFFF0.
;       2. It starts the Ram refresh.
;       3. It allocates a stack.
;       4. It transfers flow to the power-on diagnostic module.
;       5. It holds assembly language functions
;
; This module is linked as the final module in order for   it to respond to the hardware reset
; power-on jump.
;
; Version History:
; 1.01
;  In the sense program for test bios' at D000 or E000, changed the lines (2 places) from:
;       cmp     byte ptr ds:[DI+2],80h
;  to the following:
;       cmp     byte ptr ds:[2],80h
; because the inclusion of the DI index was unwanted. DI may or may not be = 0000
; 1.02
;  Deleted the test for a Bios at Segment E000
;  Added interrupt disable around the insw/outsw for the hard disk
;       rep_in and rep_out functions.
;  Added stack swap option for function call interrupts to minimize use of
;  callers stack space (for DOS 4.xx compatiblity)
;  Updated various modules (see modules for descriptions)
;
;*****************************************************************************************************
include atprfx.inc
        extrn   biospod:near        ; pod module is a destination

        extrn   boot:near
        extrn   comm_io:near
        extrn   keyboard_io:near
        extrn   keyboard_isr:near
        extrn   disk_io:near
        extrn   disk_isr:near
        extrn   printer_io:near
        extrn   video_io:near
        extrn   mem_size:near
        extrn   equipment:near
        extrn   cassette_io:near
        extrn   time_of_day:near
        extrn   timer_int:near
        extrn   print_screen:near
        extrn   watch_flag:word
        extrn   syserror:far

        extrn   restart2:near
        extrn   restart5:near
        extrn   restart9:near
        extrn   restart10:near

;------- public labels -------------

        public  ver_id
        public  bios_id
        public  name_id
        public  owner_id

        public  reset
        public  type_byte
        public  date_stamp

        public  comm_list
        public  lpt_list
        public  comm_timeout_list
        public  lpt_timeout_list
        public  config_table
        public  video_font
        public  column_table
        public  video_parms
        public  mode_table
        public  hdisk_table
        public  _acrtused


        public  org_E000
        ; a correct link may be confirmed by checking this label's location in the file Bios.map

;========== This is Bios Identification ===================

_atext  segment
        assume cs:dgroup

org_E000 label      byte

; There may be some programs sensitive to what bit pattern resides in this area. If so, this text
; area may be encoded as needed.

bios_compat_field label byte
        db      12 dup(0ffh)
        db      8ah
        db      'IBMC'
        db      14 dup(0ffh)

;========= This is part of the power up sequence =========

        romorg  _reset
reset   proc
        jmp reseta

        align 16

ver_id label byte
        db      'Version 1.02 ',0
bios_id label byte
name_id label byte

        db      'Annabooks AT BiosKit ',0
owner_id label byte
        db      'Copyright (c) FOSCO 1988, 1989 - All Rights Reserved ',0

; This ASCII field is intended to show the Copyright and even/odd identification when you are
; using 2 Proms to hold the Bios. If you examine (dump) the individual Proms, you will see a
; clear message, and will be able to discern whether it is the even or odd prom.

        align   16
        db      'CCooppyyrriigghhtt  ((cc))  11998888,,  11998899'
        db      '  FFOOSSCCOO   --   AATT--BBIIOOSSKKIITT  '
        db      '--  AAllll  RRiigghhttss  RReesseerrvveedd  ',00
        align   16
        db      'eovdedn ffiieelldd  ',00
        align   16

;--------------------------------------------------------------------------------------
; Programmers Note:
;
```

```
; We have noticed some reset problems with some CPU boards. If the power-on voltage does not come
; up cleanly and strongly, the 286 may not get a good reset. It seems that this sometimes happens
; on loaded systems with marginal power supplies.
;
; The reset signal to the CPU may not delay long enough before it rises through the threshold
; voltage to allow the CPU to start. It also may pass through the indeterminate area (can't decide
; whether it is a logic "0" or a logic "1") too slowly and break into oscillation.
;
; Another problem area we have also seen is that the crystal or oscillator may not get started
; consistently. If these problems are caused by hardware problems, there is little that the
; software can do except try to execute some delay loops until things are stable.
;
; If you experience problems in this area, a scope is handy for checking the sequencing of
; the "power good" signal from the power supply, the oscillator circuits, and the waveform of
; the reset signal. After that, an emulator may be required to help to pinpoint the problem.
;
; From a software standpoint, inserting delay loops may be about the only thing you can attempt.
;
; If you encounter, and especially if you solve a problem in this area, sharing your information
; would be a generous gesture.
;
;----------------------------------------------------------------------------------------------
;
reseta:
        cli                             ; disable interrupts
        cld
        mov     al,80h                  ; disable nmi's
        out     70h,al

;----------- check for bios' patched in ----------
; This patch checking sequence is a powerful development tool. You may link to a "test" Bios at
segment D000. It allows you to modify and test Bios',
; while still retaining an operating Bios at segment F000. For more information, see the Chapter
; on "The Patch.asm file".
;
; If we find a Bios patch (by checking its signature), then we go to it.
;
        mov     ax,0d000h ; check seg d000 first
        mov     ds,ax
        cmp     word ptr ds:[0],05b1h
        jne     check2                  ; no bios here, continue
        cmp     byte ptr ds:[2],80h
        jne     check2                  ; no bios here, continue
        mov     ax,cs                   ; are we checking ourselves ?
        cmp     ax,0d000h
        je      check2                  ; yes, skip it
        xor     cx,cx                   ; set 64k count
        xor     al,al                   ; clear checksum
        xor     si,si                   ; clear index

check1:                 ; do a checksum on the test bios
        add     al,ds:[si]
        inc     si
        loop    check1
        or      al,al
        jne     check2

        db      0eah     ; jump to bios at d000
        dw      00003h
        dw      0d000h
check2:

; First, we need to check to see if this was a power-on start or a restart from virtual mode.
;
; If it is a power-up start, then we will go through this startup and then go to BiosPod.
;
; If it is a restart we will go to a particular restart routine.

        in      al,64h   ; look at the 8742 status port
        test    al,04h   ; test the power on condition flag
        jnz     @F       ; jump if not a power on start
                         ; put valid restart code in cmos 0fh

        mov     al,8fh   ; prepare to clear cmos cell 0f
        out     70h,al   ; select the address
        delay
        xor     al,al
        out     71h,al   ; mark as valid power-up code

        xor     ax,ax
        mov     ds,ax
        mov     word ptr ds:[472h],0            ; reset warm boot flag

        jmp     power_on_start
@@:                      ; determine type of restart
```

```
        mov     al,8fh    ; prepare to read cmos cell 0f
        out     70h,al    ; select the address
        delay
        in      al,71h    ; get the cmos contents
        or      al,al
        jnz     @F
        jmp     power_on_start

@@:     cmp     al,02
        jne     @F

        mov     al,8fh              ; reset the restart code to "0"
        out     70h,al
        delay
        xor     al,al
        out     71h,al
        jmp     restart2  ; memory test return

@@:     cmp     al,05
        jne     @F

        mov     al,8fh              ; reset the restart code to "0"
        out     70h,al
        delay
        xor     al,al
        out     71h,al
        jmp     restart5  ; jmp dword with interrupt
@@:

@@:     cmp     al,09
        jne     @F

        mov     al,8fh              ; reset the restart code to "0"
        out     70h,al
        delay
        xor     al,al
        out     71h,al
        jmp     restart9  ; block move return
@@:
@@:     cmp     al,10
        jne     @F

        mov     al,8fh              ; reset the restart code to "0"
        out     70h,al
        delay
        xor     al,al
        out     71h,al
        jmp     restart10 ; jmp dword without interrupt
@@:
                                    ; unsupported restart code

;-- save the restart code in case we want to display an invalid code --

power_on_start:

;----------- start the refresh timer ---------------

        mov     al,74h                        ; start the ram refresh timer
        out     pit_port_cmnd,al
        mov     al,18
        out     pit_port_1,al
        mov     al,0
        out     pit_port_1,al

; save the warm boot flag while we clear the lower 64k

        xor     ax,ax
        mov     es,ax
        mov     bp,es:[472h]

; make sure all repeats are in forward direction

        cld

; cycle the ram memory to pre-charge the rams

        xor     ax,ax
        mov     di,0                          ; store zeroes
        mov     cx,32768
        rep     stosw

        mov     ax,-1                         ; store ones
        mov     di,0
        mov     cx,32768
        rep     stosw
```

```
          xor       ax,ax
          mov       di,0                          ; store zeroes
          mov       cx,32768
          rep       stosw

          mov       ax,-1                         ; store ones
          mov       di,0
          mov       cx,32768
          rep       stosw

; do a quick ram test on the lower 64k ----
;         copy the bios to low ram

          mov       si,0000h
          mov       di,0000h
          mov       cx,32768
          rep       movs      word ptr es:[di],word ptr cs:[si]


;         comp bios to low ram

          mov       si,0000h
          mov       di,0000h
          mov       cx,32768
          repz      cmps      word ptr es:[di],word ptr es:[si]
          mov       sp,cx               ; save cx - cx = 0 = ok, else ram compare error

; now do a ram clear on 64k - then proceed

          xor       ax,ax
          mov       cx,32768
          xor       di,di
          rep       stosw

          mov       es:[472h],bp        ; restore the warm boot flag in its place
          mov    es:[415h],sp ; save the low ram test indicator

;-- now we can use 'C' because we can have a stack !!!!

;--- set up the local stack - now we can use real calls ---

          xor       ax,ax
          cli
          mov       ss,ax
          mov       sp,ax                         ; top of the sys_seg

;/*       from      here on we can use 'C' because we have a stack */

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

          push      cs
          pop       ds                            ; 'C' wants ds = cs
          jmp       biospod
reset     endp

;=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

; The NMI routine may be expanded as desired. It resets the parity toggles and returns to
; the interrupted program.

          romorg     nmi
nmi_isr   label     byte
          push      ax
          xor       al,al
          out       0a0h,al             ; disable nmi interrupts
          in        al,61h              ; reset the nmi toggles
          or        al,30h
          out       61h,al
          and       al,0cfh
          out       61h,al
          pop       ax
          iret

; This is where the hard disk table can reside. There is room for 60-70 entries as needed. You
; may modify these entries to support the disk types you desire.


hd_tbl_struc struc
hdrive_cyls dw      ?
hdrive_heads db ?
          dw        0
hdrive_precomp dw ?
          db        0
hdrive_contrl db ?
          db        0
          db        0
          db        0
hdrive_lndng dw ?
hdrive_sectors db ?
          db        0
hd_tbl_struc ends

drive_type macro a1,a2,a3,a4,a5,a6,a7
          hd_tbl_struc <a2,a3,0,a4,0,a5,0,0,0,a6,a7,0>
          endm

          align 16

hdisk_table label byte

          drive_type 1, 306, 4,128,0, 305,17
          drive_type 2, 615, 4,300,0, 615,17
          drive_type 3, 615, 6,300,0, 615,17
          drive_type 4, 940, 8,512,0, 940,17
          drive_type 5, 940, 6,512,0, 940,17
          drive_type 6, 615, 4, -1,0, 615,17
          drive_type 7, 462, 8,256,0, 511,17
          drive_type 8, 733, 5, -1,0, 733,17
          drive_type 9, 900,15, -1,8, 901,17
          drive_type 10,820, 3, -1,0, 820,17
          drive_type 11,855, 5, -1,0, 855,17
          drive_type 12,855, 7, -1,0, 855,17
          drive_type 13,306, 8,128,0, 319,17
          drive_type 14,733, 7, -1,0, 733,17
          drive_type 15,  0, 0,  0,0,   0, 0
          drive_type 16,612, 4,  0,0, 663,17
          drive_type 17,977, 5,300,0, 977,17
          drive_type 18,977, 7, -1,0, 977,17
          drive_type 19,1024,7,512,0,1023,17
          drive_type 20,733, 5,300,0, 732,17
          drive_type 21,733, 7,300,0, 733,17
          drive_type 22,733, 5,300,0, 733,17
          drive_type 23,306, 4,  0,0, 336,17
          drive_type 24,612, 4,305,0, 663,17
          drive_type 25,306, 4, -1,0, 340,17
          drive_type 26,612, 4, -1,0, 670,17
          drive_type 27,698, 7,300,0, 732,17
          drive_type 28,976, 5,488,0, 977,17
          drive_type 29,306, 4,  0,0, 340,17
          drive_type 30,611, 4,306,0, 663,17
          drive_type 31,732, 7,300,0, 732,17
          drive_type 32,1023,5, -1,0,1023,17
          drive_type 33,306, 2, -1,0, 305,17

; The SysVue "TYPES" Command looks for this following
; entry to sense the end of the hard drive parameter table.
          drive_type -1,-1,-1,-1,-1,-1,-1 ; end of table


; this function is used to reset the parity logic

reset_parity_enables proc uses ax
          in        al,61h              ; reset the nmi toggles
          or        al,30h
          out       61h,al
          and       al,0cfh
          out       61h,al
          ret
reset_parity_enables          endp

          romorg    _boot
          jmp       boot

; This is the table referenced by the Int 15, Function C0
; get system configuration parameters call.

config_table label byte
          dw        8          ; length of table in bytes
          db        0fch       ; model byte
          db        01h        ; sub model byte
          db        0          ; bios level
          db        70h        ; DMA 3,cascade 8259, rtc clock
          db        4 dup(0); spares

          romorg    _comm_table
comm_table label byte
          dw        1047,768,384,192,96,48,24,12

          romorg    _comm_io
          jmp       comm_io

          romorg    _keyboard_io
```

```
        jmp     keyboard_io

        romorg  _keyboard_isr
        jmp     keyboard_isr

        romorg  _disk_io
        int_header      disk_io         ; disk uses own stack

        romorg  _disk_isr
        jmp     disk_isr

; This is the default disk parameter table. The Biosdisk.c Module actually uses its own set
; of enhanced parameters to support the additional drive types available.

        romorg  _disk_parms
fdisk_table label byte
        db      0dfh,2,25h,2,0fh,1bh,0ffh,54h,0f6h,0fh,8

        romorg  _printer_io
        jmp     printer_io

        romorg  _video_io       ; video uses own stack
        int_header      video_io

        romorg  _video_parms
video_parms label byte
        db      38h,40,2dh,10,1fh,6,19h,1ch,2,7,6,7,0,0,0,0
        db      71h,80,5ah,10,1fh,6,19h,1ch,2,7,6,7,0,0,0,0
        db      38h,40,2dh,10,7fh,6,64h,70h,2,1,6,7,0,0,0,0
        db      61h,80,52h,15,19h,6,19h,19h,2,13,11,12,0,0,0,0

; The length tabel is not used by bioscrt.c
; we include it in case some user thinks it exists.

length_table label byte
        dw      2048,4096,16384,16384

column_table label byte
        db      40,40,80,80,40,40,80,80
mode_table label byte
        db      2ch,28h,2dh,29h,2ah,2eh,1eh,29h

;==== this large area is used for the asm routines ======

; This is used by the hard disk driver for data-in xfers


rep_in  proc    uses es di cx dx,tseg:word,toff:word,port:word,count:word
        mov     dx,port
        mov     es,tseg
        mov     di,toff
        mov     cx,count
        pushf                   ; save flags while disabling
        cli
        cld
        rep     insw
        popf
        ret
rep_in  endp

; This is used by the hard disk driver for data-out xfers

rep_out proc    uses ds si cx dx,tseg:word,toff:word,port:word,count:word
        mov     dx,port
        mov     ds,tseg
        mov     si,toff
        mov     cx,count
        pushf                   ; save flags while disabling
        cli
        cld
        rep     outsw
        popf
        ret
rep_out endp


; --------- this is the ram test called by 'C' ------
; This test is used in real-mode only.

;unsigned int ram_test(unsigned base,unsigned length)

;       returns ok = 1
;       returns error = 0
```

```
                ; write and read bios pattern into 64k
ram_test proc uses cx ds si es di,test_base:word,test_length:word
        mov     cx,test_length
        mov     es,test_base
        push    cs
        pop     ds
        cld

        mov     si,0000h
        xor     di,di               ; set di = 0
        rep     movsw

        mov     si,0000h
        xor     di,di               ; set di = 0
        mov     cx,test_length
        repe    cmpsw

        jcxz    ram_test10
        mov     si,error
        jmp short ram_test20
ram_test10:
        mov     si,ok
ram_test20:
        xor     di,di               ; set di = 0
        mov     cx,test_length
        xor     ax,ax
        rep     stosw               ; clear the block
        mov     ax,si               ; return the condition in ax
        ret
ram_test endp

; This routine is used to re-locate the Bios-system-segment from the top of the 1st 64k segment
; (at power-on time) to the top of the System Ram at run-time.

move_system_segment proc uses ax cx dx ds si es di,to_seg:word,seg_size:word
        cli                         ; disable interrupts because
                                    ; of moving the stack
        mov     es,to_seg ; destination segment

        mov     cx,seg_size         ; calc the base segment address
        shr     cx,4
        mov     ax,1000h
        sub     ax,cx

        mov     ds,ax
        mov     cx,seg_size         ; move nnnn bytes
        xor     si,si
        xor     di,di               ; set di = 0
        rep     movs byte ptr es:[di],byte ptr ds:[si]

        mov     ax,es               ; reset the stack segment
        mov     dx,ax               ; save dest segment for 40:e
        and     ax,0f000h ; stack is top of 64k segment
        mov     ss,ax

        mov     ax,ds               ; now clear the old segment
        mov     es,ax
        xor     di,di               ; set di = 0
        mov     cx,seg_size
        xor     ax,ax
        rep     stosb               ; clear the old block

        mov     ax,40h
        mov     ds,ax
        mov     ds:[0eh],dx         ; set new sys seg ptr
        sti                                 ; re-enable interrupts
        ret
move_system_segment endp

;------- this is the rom scan checksum called by 'C' -----

; unsigned int checksum(unsigned segment,unsigned length)

checksum proc uses bx cx ds,segptr:word,length_arg:word
        mov     ds,segptr           ; get base address
        mov     cx,length_arg       ; get count
        xor     ax,ax
        xor     bx,bx
@@:     add     al,ds:[bx]
        inc     bx                          ; point to next byte
        loop    @B
        ret
checksum endp
```

```
;-- this is the far call routine called by 'C' rom scan ---

; void far_call(unsigned int segment,unsigned int address)

far_call proc segptr:word,addr:word
LOCAL temp_segment:word,temp_offset:word
          pushf
          pusha
          push      ds
          push      es
          push      segptr                  ; get segment
          pop       temp_segment
          push      addr                    ; get offset
          pop       temp_offset
          call      dword ptr [temp_offset]
          pop       es
          pop       ds
          popa
          popf
          ret
far_call endp

;-- this jump to boot passes the drive # in dl --

; boot_jump(seg,off,drive);

boot_jump proc      segptr:word,addr:word,drive:word
local temp_segment:word,temp_offset:word

          push      segptr        ; get segment
          pop       temp_segment
          push      addr                    ; get offset
          pop       temp_offset
          mov       dx,drive
          jmp       dword ptr [temp_offset]
boot_jump endp

;------------ beep routine ---------------

; void beep(void);

beep      proc uses ax bx cx dx
          mov       bx,128        ; 128 cycles last about 1/12 sec.
          mov       dx,pio_port_b
          in        al,dx         ; save the 8255 port byte
          mov       ah,al
beep10:
          and         al,0fch     ; turn the beeper off
          out         dx,al
          mov         cx,64       ; make the duty cycle 50%
@@:       loop      @B
          or        al,2                    ; turn the beeper on
          out       dx,al
          mov       cx,64         ; leave it on = to the off time
@@:       loop      @B
          dec       bx                      ; count the major cycles down
          jnz       beep10
          mov       al,ah         ; get the original port byte
          out       dx,al         ; restore the 8255 to previous
          ret
beep      endp

;-----------------------------------------------

seg_40_constant dw 40h

; void setds_system_segment(void);

setds_system_segment proc
          mov       ds,cs:seg_40_constant
          mov       ds,ds:[0eh]
          ret
setds_system_segment endp

;-- this is used to set the DS to a specified value --
; void setds(unsigned ptr);

setds proc seg_ptr:word
          push      seg_ptr
          pop       ds
          ret
setds endp

;-- get the current stack segment value --
; unsigned get_ss(void);
```

```
get_ss    proc
          mov       ax,ss
          ret
get_ss    endp

;-- test the specified watch flag bits --
; return true if set, false if not

; unsigned watch(unsigned);
watch     proc      uses ds,bits:word
          call      setds_system_segment
          mov       ax,bits
          test      ds:watch_flag,ax
          mov       ax,false
          jz        @F
          mov       ax,true
@@:       ret
watch     endp

;-- this can be used to force a break while debugging --

trap      proc                ; this can be used for hard breakpointing
          int       18h       ; goes to SysVue
          ret
trap      endp

; check to see if any key present at console device --
; returns true if a key waiting

kbhit     proc                ; returns true if any key hit
          mov       ax,0100h
          int       16h
          mov       ax,false
          jz        @F
          mov       ax,true
@@:
          ret
kbhit     endp


;=========== Console In Routine ===============

;         char ci(void);
ci        proc
          mov       ax,0
          int       16h
          ret
ci        endp

;*========= block checksum routine ==============*/

; unsigned checksum_bios_block(void)
          extrn bios_start:near

checksum_bios_block proc uses bx ds,seg_addr:word,off_addr:word
          push      cs
          pop       ds                  ; set ds = cs to eliminate need for CS: overide
          xor       ax,ax               ; clear accumulator
          mov       bx,offset bios_start
@@:       add       al,[bx]
          inc       bx                  ; exit at end of segment
          jnz       @B
          ret
checksum_bios_block endp


;*=========== Console Out Routine =============*/

; console out routine
; co(char);

co        proc uses ax bx,char:byte
          mov       ah,0eh
          mov       al,char
          mov       bx,0
          int       10h
          ret
co        endp

;--- calc the hdisk size - this uses some long unsigneds --

; unsigned calc_hdisk_size(cyls,heads,sectors);
; returns a value in ax in  megabytes
; By doing this in ASM, the C Lib function is not needed.

calc_hdisk_size     proc uses dx si,cyls:word,heads:word,sectors:word
```

```
        xor     dx,dx
        mov     ax,cyls
        mov     si,heads
        mul     si
        mov     si,sectors
        mul     si
        mov     si,2000
        div     si
        ret
calc_hdisk_size endp

;-----------------------------------------------

; this is used by the timer interrupt to chain to a user

timer_chain     proc uses ds
        push    40h
        pop     ds
        int     1ch
        ret
timer_chain     endp

; this chains the hard -> floppy disk without extra
; stack usage

fdisk_chain     proc
        mov     sp,bp     ; discard the callers return address
        pop     es
        pop     ds
        popa
        int     40h
        retf 2
fdisk_chain     endp

;-----------------------------------------
; returns the high order word of an unsigned long integer as an unsigned in ax
; unsigned hi_regs(unsigned long)
;
hi_regs proc    low_arg:word,high_arg:word
        mov     ax,high_arg
        ret
hi_regs endp

;--------long multiply for setting tod clock ------

; unsigned long lmul(unsigned long,unsigned)

lmul    proc    uses bx,ax_arg:word,dx_arg:word,bx_arg:word
        mov     dx,dx_arg
        mov     ax,ax_arg
        mov     bx,bx_arg
        mul     bx
        ret
lmul    endp

convert_binary  proc    arg:word
        mov     ax,arg
        mov     ah,al
        shr     ah,4
        and     al,0fh
        aad
        xor     ah,ah
        ret
convert_binary  endp

; void bin2dec(value,return_array[5]);
; the output array is in DS:

bin2dec proc uses ax di si dx,value:word,array_ptr:word
        mov     di,array_ptr
        mov     byte ptr ds:[di+0],' '
        mov     byte ptr ds:[di+1],' '
        mov     byte ptr ds:[di+2],' '
        mov     byte ptr ds:[di+3],' '
        mov     byte ptr ds:[di+4],'0'
        mov     byte ptr ds:[di+5],0        ; string terminator char

        add     di,4
        mov     si,10
        mov     ax,value
@@:     mov     dx,0
        div     si
        add     dl,'0'
        mov     ds:[di],dl
        dec     di
        or      ax,ax
        jnz     @B
        ret
bin2dec endp

;-------------------------------------------

; unsigned dec2bin(unsigned decval);
; the arg is max of 9999 decimal - ok for up to 8192k of extended memory
; the binary value is returned in ax

dec2bin proc uses dx di cx,decval:word
        mov     dx,0            ; clear accumulator
        mov     ax,0
        mov     di,10   ; load multiplier
        mov     cx,4            ; load loop count
@@:     mul     di              ; mul by ten
        rol     decval,4  ; get next digit
        mov     bx,decval ; load to bx
        and     bx,000fh  ; save only wanted digit
        add     ax,bx     ; add to accumulated number
        loop    @B              ; do for all four digits
        ret
dec2bin endp

;------ move string routine for alpha mode crt scrolling

; void move_row(unsigned segment,unsigned from_offset,unsigned to_offset, unsigned count)

move_row proc uses cx si di es ds,segptr:word,from:word,to:word,count
        mov     ds,segptr       ; segment
        mov     es,segptr
        mov     si,from         ; from
        mov     di,to           ; to
        mov     cx,count        ; count
        cld
        jcxz    @F
        rep       movsw
@@:     ret
move_row endp

;------ clear row routine for alpha mode crt scrolling

; void clear_row(unsigned segment, unsigned address,unsigned count, unsigned fill)

clear_row proc uses ax cx di es,segptr:word,addr:word,count:word,filler:word
        mov     es,segptr       ; segment
        mov     di,addr         ; address
        mov     cx,count        ; count
        mov     ax,filler       ; fill
        cld
        jcxz    @F
        rep       stosw
@@:     ret
clear_row endp

;-- move string routine for graphics mode crt scrolling

; void move_graphics_row(segment,       from_offset,to_offset, count)

move_graphics_row proc uses cx dx si di es ds,segptr:word,from:word,to:word,count:word
        mov     dx,0                    ; use dx to move four lines each
move_graphics_row5:
        mov     ds,segptr       ; segment
        mov     es,segptr
        mov     si,from         ; from
        mov     di,to           ; to
        mov     cx,count        ; count
        cld
        add     si,dx
        add     di,dx
        jcxz    @F
        rep       movsw
@@:     mov     si,[bp+6]               ; from
        mov     di,[bp+8]               ; to
        add     si,2000h                ; add for odd row
        add     di,2000h                ; add for odd row
        mov     cx,[bp+10]              ; count
        cld
        add     si,dx
        add     di,dx
        jcxz    @F
        rep       movsw
@@:     add     dx,80
        cmp     dx,320
        jb      move_graphics_row5
```

```
          ret
move_graphics_row endp

;------ clear row routine for alpha mode crt scrolling

; void clear_graphics_row(segment, address, count, fill)

clear_graphics_row proc uses ax cx dx di es,segptr:word,addr:word,count:word,filler:word
          mov      dx,0                      ; use dx to clear four lines each
clear_graphics_row5:
          mov      es,segptr       ; segment
          mov      di,addr         ; address
          mov      cx,count        ; count
          mov      ax,filler       ; fill
          cld
          add      di,dx
          jcxz     @F
          rep        stosw
@@:       mov      di,[bp+6]                 ; address
          add      di,2000h                  ; add for odd row
          mov      cx,[bp+8]                 ; count
          mov      ax,[bp+10]                ; fill
          add      di,dx
          cld
          jcxz     @F
          rep        stosw
@@:       add      dx,80
          cmp      dx,320
          jb       clear_graphics_row5
          ret
clear_graphics_row endp


; ---- low level routines for 'C' ---------

;=======================================================
; this version assumes the regs are in the system segment. note that it also requires a
; slightly different regs structure than int86 or int86x because it always passes es and ds.

axoff     equ      4
cxoff     equ      6
dxoff     equ      8
sioff     equ      10
dioff     equ      12
bpoff     equ      14
bxoff     equ      16
dsoff     equ      18
esoff     equ      20
floff     equ      22
jmp_off   equ      24
jmp_seg   equ      26
code_string_location equ 28


; void sys_int(char number,reg_block);

code_string        proc far
          lds      bx,cs:[bx+bxoff]
          int      0
number_offset = $-code_string-1
          ret
code_string_length = $-code_string
code_string        endp


sys_int proc uses si di ax bx cx dx es ds,       number:word,regs_block:word

          mov      bx,regs_block

; move the code string image
                   ; get count of bytes to move
          mov      cx,code_string_length
                   ; point to org of code_string
          mov      si,offset code_string
                   ; build ptr to destination
          mov      di,bx
          add      di,offset code_string_location
                   ; set segment for destination
          push     ds
          pop      es
                   ; now do the copy
          rep      movs      byte ptr es:[di], byte ptr cs:[si]

                   ; load the correct interrupt number
          mov      ax,number
          mov      [bx+code_string_location+number_offset],al
```

```
                ; load the jump vector
        mov     word ptr [bx+jmp_off],code_string_location
        add     [bx+jmp_off],bx
        mov     [bx+jmp_seg],ds

                ; load the registers
        mov     ax,[bx+axoff]
        mov     cx,[bx+cxoff]
        mov     dx,[bx+dxoff]
        mov     si,[bx+sioff]
        mov     di,[bx+dioff]
        mov     es,[bx+esoff]

                ; save the real bp for the implied LEAVE instruction
        push    bp
        mov     bp,[bx+bpoff]
        push    ds
        push    bx
        call    dword ptr ds:[bx+jmp_off]

; now we have to swap out ds:bx with the stack-saved ones

        push    bx                      ; save returned ds:bx
        push    ds
        push    bp                      ; save bp - will be used as ptr
        mov     bp,sp
        lds     bx,[bp+6] ; re-load myblock ds:bx

        pop     ds:[bx+bpoff]           ; restore returned bp
        pop     ds:[bx+dsoff]
        pop     ds:[bx+bxoff]

        pop     bp                      ; discard old ds:bx slots
        pop     bp                      ; without affecting flags

                ; bp is now pre-interrupt value
        pop     bp

                ; store the registers

        mov     [bx+esoff],es
        mov     [bx+axoff],ax
        mov     [bx+cxoff],cx
        mov     [bx+dxoff],dx
        mov     [bx+sioff],si
        mov     [bx+dioff],di
        pushf   ; save flags
        pop     [bx+floff]

        ret
sys_int endp

; SysErr routine puts an error code in ax, then goes to sysvue -- This is an expansion feature
; you may wish to implement.

sys_err proc    uses ax,error_code:word
        mov     ax,error_code
        pushf
        call    syserror
        ret
sys_err endp

;-----------------------------------------------
;
; get vector returns the selected vector in DX:AX.
; This routine is done in assembly language because a 'C'
; routine would need the library long shift function to
; build the seg:off into DX:AX.
;
; unsigned long get_vector(char number);

get_vector proc uses bx es,number:word
        xor     bx,bx
        mov     es,bx
        mov     bx,number               ; int #
        xor     bh,bh
        shl     bx,1                            ; mul by 4
        shl     bx,1
        cli
        mov     dx,es:[bx]+2
        mov     ax,es:[bx]
        sti
        ret
get_vector endp

getds proc
```

```
          mov       ax,ds
          ret
getds     endp

; return the bios code segment value in AX.
; This function is coded this way so the Bios may be operated at segments other than F000.

; unsigned bios_cs(void);


bios_cs proc
          mov       ax,cs
          ret
bios_cs endp

; Disable interrupts

; void disable(void);

disable proc
          cli
          ret
disable endp

; Enable interrupts

; void enable(void);

enable proc
          sti
          ret
enable endp


; void delay(unsigned res,unsigned count);


;         res == 0 - delay in useconds
;         res != 0 - use as milliseconds resolution
;         count = count

delay_call proc uses ax cx dx ds,res:word,count:word
          cmp       res,0
          jnz       milli_delay

micro_delay:
          mov       ax,count  ; micro seconds delay
          xor       dx,dx               ; this has 15 usec resolution
          mov       cx,15               ; each toggle of the refresh bit is 15 usecs
          div       cx                  ; now cx = count/15
          add       ax,2                ; make sure we have at least 15-30 usecs min.
          mov       cx,ax               ; cx will be a loop count

@@:       in        al,61h              ; get port_b refresh bit
          and       al,10h              ; save only the bit we want
          cmp       ah,al               ;
          je        @B
          mov       ah,al               ; now use current state
          loop      @B                  ; wait n times
          ret

milli_delay:                            ; res argument = # of millis resolution
          mov       cx,count  ; this is major loop
milli_major:
          push      cx
          mov       cx,res              ; this is minor loop
@@:       call      delay_a_milli
          loop      @B                  ; wait for resolution counts
          pop       cx
          loop      milli_major         ; repeat major loop for total counts
          ret
delay_call          endp

read_timer_count    proc      ; each count = .8 usecs
          xor       al,al               ; latch timer 0 count into storage register
          out       43h,al
          jmp       @F                  ; delay for 8253 response
@@:       jmp       @F
@@:       jmp       @F
@@:       in        al,40h
          mov       ah,al
          jmp       @F                  ; delay for 8253 response
@@:       jmp       @F
@@:       jmp       @F
@@:       in        al,40h
          xchg      ah,al
```

```
        ret                             ; return count in ax
read_timer_count    endp

; Note: Remember that the counters in the 8253/8254 are down counters,
; so that we have to subtract the desired count from the current count.

delay_a_milli       proc uses cx        ; delay 1 millisecond
        call        read_timer_count    ; do a double read
        call        read_timer_count
        mov         cx,ax               ; since count rate is 1.19 Mhz, about
        sub         cx,1250             ; 1250 timer counts = 1 millisecond
        cmp         cx,ax
        jb          @F                  ; delay count > current count, no rollover
        mov         cx,ax               ; use delay based from zero if rollover
        not         cx
@@:     call        read_timer_count
        cmp         ax,cx               ; delay until current count >= calced count
        jae         @B
        ret
delay_a_milli       endp


get_offset          proc      ; this function is used to retrun the callers offset
        pop         ax        ; it pops the callers address into ax
        push        ax        ; puts it back so it can be used for the ret instruction
        ret
get_offset          endp

;--- These are additional peek and poke functions ---

; unsigned peek40(unsigned offset);

peek40 proc uses bx es,offptr:word      ; peeks in segment 40h
        mov         bx,40h
        mov         es,bx
        mov         bx,offptr
        mov         ax,es:[bx]
        ret
peek40 endp

; char peekb40(unsigned offset);

peekb40 proc uses bx es,offptr:word
        mov         bx,40h
        mov         es,bx
        mov         bx,offptr
        mov         al,es:[bx]
        xor         ah,ah
        ret
peekb40 endp

; char peekbcs(unsigned offset);

peekbcs proc uses bx,offptr:word
        mov         bx,offptr
        mov         al,cs:[bx]
        xor         ah,ah
        ret
peekbcs endp

; unsigned peekcs(unsigned offset);

peekcs proc uses bx,offptr:word
        mov         bx,offptr
        mov         ax,cs:[bx]
        ret
peekcs endp

; void pokeb40(unsigned offset, char value);

pokeb40 proc uses ax bx es,offptr:word,value:word
        mov         bx,40h
        mov         es,bx
        mov         bx,offptr
        mov         ax,value
        mov         es:[bx],al
        ret
pokeb40 endp

; void poke40(unsigned offset, unsigned value);

poke40 proc uses ax bx es,offptr:word,value:word
        mov         bx,40h
        mov         es,bx
        mov         bx,offptr
        mov         ax,value
```

```
          mov       es:[bx],ax
          ret
poke40 endp

; void andb40(unsigned offset, char value);

andb40 proc uses ax bx es,offptr:word,value:word
          mov       bx,40h
          mov       es,bx
          mov       bx,offptr
          mov       ax,value
          and       es:[bx],al
          ret
andb40 endp

; void and40(unsigned offset, unsigned value);

and40     proc uses ax bx es,offptr:word,value:word
          mov       bx,40h
          mov       es,bx
          mov       bx,offptr
          mov       ax,value
          and       es:[bx],ax
          ret
and40     endp

; void orb40(unsigned offset, char value);

orb40     proc uses ax bx es,offptr:word,value:word
          mov       bx,40h
          mov       es,bx
          mov       bx,offptr
          mov       ax,value
          or        es:[bx],al
          ret
orb40     endp

; void xorb40(unsigned offset, char value);

xorb40 proc uses ax bx es,offptr:word,value:word
          mov       bx,40h
          mov       es,bx
          mov       bx,offptr
          mov       ax,value
          xor       es:[bx],al
          ret
xorb40 endp

;--- These are the standard peeks, pokes, ins, and outs ----

; unsigned peek(unsigned segment,unsigned offset)

peek      proc uses bx es,segptr:word,offptr:word
          mov       es,segptr
          mov       bx,offptr
          mov       ax,es:[bx]
          ret
peek      endp

; void poke(segment, offset, value);

poke      proc uses ax bx es,segptr:word,offptr:word,value:word
          mov       es,segptr
          mov       bx,offptr
          mov       ax,value
          mov       es:[bx],ax
          ret
poke      endp

; char peekb(unsigned segment,unsigned offset);

peekb     proc uses bx es,segptr:word,offptr:word
          mov       es,segptr
          mov       bx,offptr
          mov       al,es:[bx]
          xor       ah,ah
          ret
peekb     endp

; void pokeb(unsigned segment, unsigned offset, char value);

pokeb     proc uses ax bx es,segptr:word,offptr:word,value:word
          mov       es,segptr
          mov       bx,offptr
          mov       ax,value
          mov       es:[bx],al
```

```
        ret
pokeb   endp

; unsigned inport(unsigned port);

inport proc uses dx,port:word
        mov     dx,port
        in      ax,dx
        ret
inport endp

; void outport(unsigned port, unsigned value);

outport proc uses dx ax,port:word,value:word
        mov     dx,port
        mov     ax,value
        out     dx,ax
        ret
outport endp

; char inportb(unsigned port);

inportb proc uses dx,port:word
        mov     dx,port
        in      al,dx
        xor     ah,ah
        ret
inportb endp

; void outportb(unsigned port, char value);

outportb proc uses ax dx,port:word,value:word
        mov     dx,port
        mov     ax,value
        out     dx,al
        ret
outportb endp

cstods  proc    ; set ds = to cs
        push    cs
        pop     ds
        ret
cstods  endp

$dummy_isr proc
        send_eoi
        iret
$dummy_isr endp

$iret   proc
        iret
$iret   endp


;--- Stack swapper for interrupt routines - (see int_header macro) ---
;
;--- Typically code written in C may use more stack space then
;--- tweaked assembly language code. Intermediate variables created
;--- by using "for" and "while" loops may be assigned to the stack.
;--- Also the use of the "Interrupt" declaration for Interrupt
;--- service routines causes all the registers to be pushed on the
;--- stack. Normally this does not cause problems, because user or
;--- application declared stacks are usually large enough to be safe.
;--- However, there are some instances where a program ignores the user
;--- declared stack (PC-DOS 3.xx VDISK) or system software (PC/MS 4.xx
;--- IO.SYS) sets a very tight stack, which will run with most (but not all)
;--- BIOS' written in assembly. (The more curious among us may have
;--- already discovered the phrase "may run on some PC and XT machines",
;--- in the DOS 4.xx manuals).
;---
;--- Starting with DOS 3.xx, the "stacks=n,y" command has been included
;--- for CONFIG.SYS use to set up a stack pool for swapping stacks on
;--- the hardware (caused through the 8259 interrupt controller(s)) interrupts.
;--- This alleviated some stack overflow problems encountered when
;--- running true IBM PC-DOS on true IBM hardware. So it wasn't an error
;--- committed either by the 3rd party clone or BIOS creators. It simply
;--- was a result of the programming style used in writing the DOS.
;--- Microsoft is currently the most prominent "bender" of the rules
;--- (previously IBM possessed the leverage) and we (as well as Intel)
;--- are forced to conform. The code below implements a stack swapping
;--- mechanism for use by Function Call Interrupts by creating a stack
;--- in the system segment ram area for the desired function call.
;--- The swap occurs before the Interrupt Service Routine is called,
;--- allowing all the user registers to be pushed onto the new stack.
;--- This limits user stack usage and seems to solve the problems.
```

```
; Example of usage:
;
; A non-swapped interrupt -
; Vector ---------> cdecl far interrupt sample(interrupt_registers)
;                         .
;                         .
;               <-----------.
;
; A swapped interrupt -
; Vector ---------> push dest
;               jmp     interrupt_shell
;                               .
;                               . (swap to system stack)
;                               .
;                               . ----> cdecl far interrupt sample(interrupt_registers)
;                                       .
;                                       .
;                               .<----------.
;                               .
;                               . (restore to callers stack)
;                               .
;               <-----------.
;
;
; The code below is entered with destination address on the stack
;       push    dest
;       jmp     interrupt_shell

        extrn   bx_save:word,ds_save:word,dest_save:word
        extrn   ss_save:word,sp_save:word,length_mark:word
        extrn   current_open:word,end_block:word

size_requested equu 256        ; default standard size

interrupt_shell proc
                        ; entered with interrupts disabled as a
                        ; result of the previous Int xx opcode.
        push    ds
        push    bx
        mov     bx,0
        mov     ds,bx
        mov     ds,ds:[040eh]           ; get system segment
        pop     bx_save                 ; sys:xx
        pop     ds_save                 ; sys:xx+2
        pop     dest_save       ; sys:xx+4
        mov     bx,current_open
        pushf                           ; save callers flag state
        add     bx,size_requested   ; check for space available
        cmp     bx,end_block
        jb      @F                      ; jump if enough space
        popf                            ; use current stack, jump to isr service
        push    dest_save
        lds     bx,dword ptr ds:[bx_save]      ; restore ds:bx to callers values
        ret
@@:
        mov     bx,current_open         ; get base address of block
        add     current_open,size_requested
        popf
                                ; mark the block "in use"
        mov     ds:[bx+length_mark],size_requested
        mov     ds:[bx+ss_save],ss  ; save the caller stack pointers
        mov     ds:[bx+sp_save],sp  ; in the allocated stack block
        mov     sp,ds
        mov     ss,sp
        mov     sp,ds:[current_open]        ; set sp at top of new block

        pushf
        push    cs
        push    offset continue     ; set up phony interrupt call
        push    dest_save           ; set destination address
        lds     bx,dword ptr ds:[bx_save]     ; restore ds:bx to callers values

        ret                         ; go to dest with phnoy call

continue:
        push    bx
        push    ds
        mov     bx,0
        mov     ds,bx
        mov     ds,ds:[40eh]
        pop     ds_save
        pop     bx_save
        pushf
        sub     ds:[current_open],size_requested
        popf
```

```
          mov       bx,current_open
          mov       ss,ds:[bx+ss_save]  ; get the callers stack description
          mov       sp,ds:[bx+sp_save]
          mov       ds:[bx+length_mark],0        ; mark the block as free
          lds       bx,dword ptr ds:[bx_save]

          retf      2                            ; return with cuurent flags
interrupt_shell     endp

;============== end of the asm routines ===================

          romorg    _mem_size
          jmp       mem_size

          romorg    _equipment
          jmp       equipment

          romorg    _cassette_io                 ; cassette uses own stack
          int_header          cassette_io

          romorg    _video_font
video_font label byte
;         These are the bit-mapped characters for the graphics mode. They must be here
;         because some user software uses these for generating characters in graphics mode.

          db        000h,000h,000h,000h,000h,000h,000h,000h ; 00 - nul
          db        07eh,081h,0a5h,081h,0bdh,099h,081h,07eh ; 01 - ^A
          db        07eh,0ffh,0dbh,0ffh,0c3h,0e7h,0ffh,07eh ; 02 - ^B
          db        06ch,0feh,0feh,0feh,07ch,038h,010h,000h ; 03 - ^C
          db        010h,038h,07ch,0feh,07ch,038h,010h,000h ; 04 - ^D
          db        038h,07ch,038h,0feh,0feh,07ch,038h,07ch ; 05 - ^E
          db        010h,010h,038h,07ch,0feh,07ch,038h,07ch ; 06 - ^F
          db        000h,000h,018h,03ch,03ch,018h,000h,000h ; 07 - ^G
          db        0ffh,0ffh,0e7h,0c3h,0c3h,0e7h,0ffh,0ffh ; 08 - ^H
          db        000h,03ch,066h,042h,042h,066h,03ch,000h ; 09 - ^I
          db        0ffh,0c3h,099h,0bdh,0bdh,099h,0c3h,0ffh ; 0A - ^J
          db        00fh,007h,00fh,07dh,0cch,0cch,0cch,078h ; 0B - ^K
          db        03ch,066h,066h,066h,03ch,018h,07eh,018h ; 0C - ^L
          db        03fh,033h,03fh,030h,030h,070h,0f0h,0e0h ; 0D - ^M
          db        07fh,063h,07fh,063h,06eh,067h,0e6h,0c0h ; 0E - ^N
          db        099h,05ah,03ch,0e7h,0e7h,03ch,05ah,099h ; 0F - ^O
          db        080h,0e0h,0f8h,0feh,0f8h,0e0h,080h,000h ; 10 - ^P
          db        002h,00eh,03eh,0feh,03eh,00eh,002h,000h ; 11 - ^Q
          db        018h,03ch,07eh,018h,018h,07eh,03ch,018h ; 12 - ^R
          db        066h,066h,066h,066h,066h,000h,066h,000h ; 13 - ^S
          db        07fh,0dbh,0dbh,07bh,01bh,01bh,01bh,000h ; 14 - ^T
          db        03eh,063h,038h,06ch,06ch,038h,0cch,078h ; 15 - ^U
          db        000h,000h,000h,000h,07eh,07eh,07eh,000h ; 16 - ^V
          db        018h,03ch,07eh,018h,07eh,03ch,018h,0ffh ; 17 - ^W
          db        018h,03ch,07eh,018h,018h,018h,018h,000h ; 18 - ^X
          db        018h,018h,018h,018h,07eh,03ch,018h,000h ; 19 - ^Y
          db        000h,018h,00ch,0feh,00ch,018h,000h,000h ; 1A - ^Z
          db        000h,030h,060h,0feh,060h,030h,000h,000h ; 1B - ^[
          db        000h,000h,0c0h,0c0h,0c0h,0feh,000h,000h ; 1C - ^\
          db        000h,024h,066h,0ffh,066h,024h,000h,000h ; 1D - ^]
          db        000h,018h,03ch,07eh,0ffh,0ffh,000h,000h ; 1E - ^6
          db        000h,0ffh,0ffh,07eh,03ch,018h,000h,000h ; 1F - ^-
          db        000h,000h,000h,000h,000h,000h,000h,000h ; 20 - spc
          db        030h,078h,078h,030h,030h,000h,030h,000h ; 21 - !
          db        06ch,06ch,06ch,000h,000h,000h,000h,000h ; 22 - "
          db        06ch,06ch,0feh,06ch,0feh,06ch,06ch,000h ; 23 - #
          db        030h,07ch,0c0h,078h,00ch,0f8h,030h,000h ; 24 - $
          db        000h,0c6h,0cch,018h,030h,066h,0c6h,000h ; 25 - %
          db        038h,06ch,038h,076h,0dch,0cch,076h,000h ; 26 - &
          db        060h,060h,0c0h,000h,000h,000h,000h,000h ; 27 - '
          db        018h,030h,060h,060h,060h,030h,018h,000h ; 28 - (
          db        060h,030h,018h,018h,018h,030h,060h,000h ; 29 - )
          db        000h,066h,03ch,0ffh,03ch,066h,000h,000h ; 2A - *
          db        000h,030h,030h,0fch,030h,030h,000h,000h ; 2B - +
          db        000h,000h,000h,000h,000h,030h,030h,060h ; 2C - ,
          db        000h,000h,000h,0fch,000h,000h,000h,000h ; 2D - -
          db        000h,000h,000h,000h,000h,030h,030h,000h ; 2E - .
          db        006h,00ch,018h,030h,060h,0c0h,080h,000h ; 2F - /
          db        07ch,0c6h,0ceh,0deh,0f6h,0e6h,07ch,000h ; 30 - 0
          db        030h,070h,030h,030h,030h,030h,0fch,000h ; 31 - 1
          db        078h,0cch,00ch,038h,060h,0cch,0fch,000h ; 32 - 2
          db        078h,0cch,00ch,038h,00ch,0cch,078h,000h ; 33 - 3
          db        01ch,03ch,06ch,0cch,0feh,00ch,01eh,000h ; 34 - 4
          db        0fch,0c0h,0f8h,00ch,00ch,0cch,078h,000h ; 35 - 5
          db        038h,060h,0c0h,0f8h,0cch,0cch,078h,000h ; 36 - 6
          db        0fch,0cch,00ch,018h,030h,030h,030h,000h ; 37 - 7
          db        078h,0cch,0cch,078h,0cch,0cch,078h,000h ; 38 - 8
          db        078h,0cch,0cch,07ch,00ch,018h,070h,000h ; 39 - 9
          db        000h,030h,030h,000h,000h,030h,030h,000h ; 3A - :
          db        000h,030h,030h,000h,000h,030h,030h,060h ; 3B - ;
          db        018h,030h,060h,0c0h,060h,030h,018h,000h ; 3C - <
```

```
        db      000h,000h,0fch,000h,000h,0fch,000h,000h ; 3D - =
        db      060h,030h,018h,00ch,018h,030h,060h,000h ; 3E - >
        db      078h,0cch,00ch,018h,030h,000h,030h,000h ; 3F - ?
        db      07ch,0c6h,0deh,0deh,0deh,0c0h,078h,000h ; 40 - @
        db      030h,078h,0cch,0cch,0fch,0cch,0cch,000h ; 41 - A
        db      0fch,066h,066h,07ch,066h,066h,0fch,000h ; 42 - B
        db      03ch,066h,0c0h,0c0h,0c0h,066h,03ch,000h ; 43 - C
        db      0f8h,06ch,066h,066h,066h,06ch,0f8h,000h ; 44 - D
        db      0feh,062h,068h,078h,068h,062h,0feh,000h ; 45 - E
        db      0feh,062h,068h,078h,068h,060h,0f0h,000h ; 46 - F
        db      03ch,066h,0c0h,0c0h,0ceh,066h,03eh,000h ; 47 - G
        db      0cch,0cch,0cch,0fch,0cch,0cch,0cch,000h ; 48 - H
        db      078h,030h,030h,030h,030h,030h,078h,000h ; 49 - I
        db      01eh,00ch,00ch,00ch,0cch,0cch,078h,000h ; 4A - J
        db      0e6h,066h,06ch,078h,06ch,066h,0e6h,000h ; 4B - K
        db      0f0h,060h,060h,060h,062h,066h,0feh,000h ; 4C - L
        db      0c6h,0eeh,0feh,0feh,0d6h,0c6h,0c6h,000h ; 4D - M
        db      0c6h,0e6h,0f6h,0deh,0ceh,0c6h,0c6h,000h ; 4E - N
        db      038h,06ch,0c6h,0c6h,0c6h,06ch,038h,000h ; 4F - O
        db      0fch,066h,066h,07ch,060h,060h,0f0h,000h ; 50 - P
        db      078h,0cch,0cch,0cch,0dch,078h,01ch,000h ; 51 - Q
        db      0fch,066h,066h,07ch,06ch,066h,0e6h,000h ; 52 - R
        db      078h,0cch,0e0h,070h,01ch,0cch,078h,000h ; 53 - S
        db      0fch,0b4h,030h,030h,030h,030h,078h,000h ; 54 - T
        db      0cch,0cch,0cch,0cch,0cch,0cch,0fch,000h ; 55 - U
        db      0cch,0cch,0cch,0cch,0cch,078h,030h,000h ; 56 - V
        db      0c6h,0c6h,0c6h,0d6h,0feh,0eeh,0c6h,000h ; 57 - W
        db      0c6h,0c6h,06ch,038h,038h,06ch,0c6h,000h ; 58 - X
        db      0cch,0cch,0cch,078h,030h,030h,078h,000h ; 59 - Y
        db      0feh,0c6h,08ch,018h,032h,066h,0feh,000h ; 5A - Z
        db      078h,060h,060h,060h,060h,060h,078h,000h ; 5B - [
        db      0c0h,060h,030h,018h,00ch,006h,002h,000h ; 5C - \
        db      078h,018h,018h,018h,018h,018h,078h,000h ; 5D - ]
        db      010h,038h,06ch,0c6h,000h,000h,000h,000h ; 5E - ^
        db      000h,000h,000h,000h,000h,000h,000h,0ffh ; 5F -
        db      030h,030h,018h,000h,000h,000h,000h,000h ; 60 - `
        db      000h,000h,078h,00ch,07ch,0cch,076h,000h ; 61 - a
        db      0e0h,060h,060h,07ch,066h,066h,0dch,000h ; 62 - b
        db      000h,000h,078h,0cch,0c0h,0cch,078h,000h ; 63 - c
        db      01ch,00ch,00ch,07ch,0cch,0cch,076h,000h ; 64 - d
        db      000h,000h,078h,0cch,0fch,0c0h,078h,000h ; 65 - e
        db      038h,06ch,060h,0f0h,060h,060h,0f0h,000h ; 66 - f
        db      000h,000h,076h,0cch,0cch,07ch,00ch,0f8h ; 67 - g
        db      0e0h,060h,06ch,076h,066h,066h,0e6h,000h ; 68 - h
        db      030h,000h,070h,030h,030h,030h,078h,000h ; 69 - i
        db      00ch,000h,00ch,00ch,00ch,0cch,0cch,078h ; 6A - j
        db      0e0h,060h,066h,06ch,078h,06ch,0e6h,000h ; 6B - k
        db      070h,030h,030h,030h,030h,030h,078h,000h ; 6C - l
        db      000h,000h,0cch,0feh,0feh,0d6h,0c6h,000h ; 6D - m
        db      000h,000h,0f8h,0cch,0cch,0cch,0cch,000h ; 6E - n
        db      000h,000h,078h,0cch,0cch,0cch,078h,000h ; 6F - o
        db      000h,000h,0dch,066h,066h,07ch,060h,0f0h ; 70 - p
        db      000h,000h,076h,0cch,0cch,07ch,00ch,01eh ; 71 - q
        db      000h,000h,0dch,076h,066h,060h,0f0h,000h ; 72 - r
        db      000h,000h,07ch,0c0h,078h,00ch,0f8h,000h ; 73 - s
        db      010h,030h,07ch,030h,030h,034h,018h,000h ; 74 - t
        db      000h,000h,0cch,0cch,0cch,0cch,076h,000h ; 75 - u
        db      000h,000h,0cch,0cch,0cch,078h,030h,000h ; 76 - v
        db      000h,000h,0c6h,0d6h,0feh,0feh,06ch,000h ; 77 - w
        db      000h,000h,0c6h,06ch,038h,06ch,0c6h,000h ; 78 - x
        db      000h,000h,0cch,0cch,0cch,07ch,00ch,0f8h ; 79 - y
        db      000h,000h,0fch,098h,030h,064h,0fch,000h ; 7A - z
        db      01ch,030h,030h,0e0h,030h,030h,01ch,000h ; 7B - {
        db      018h,018h,018h,000h,018h,018h,018h,000h ; 7C - |
        db      0e0h,030h,030h,01ch,030h,030h,0e0h,000h ; 7D - }
        db      076h,0dch,000h,000h,000h,000h,000h,000h ; 7E - ~
        db      000h,010h,038h,06ch,0c6h,0c6h,0feh,000h ; 7F -

        romorg  _time_of_day
        jmp     time_of_day

        romorg  _timer_int
        jmp     timer_int

        romorg  _dummy_iret
        iret

        romorg  _print_screen                   ; print screen uses own stack
        int_header print_screen

abort_string label byte
        db      13,10,'WARNING - This program cannot be executed under DOS.',13,10,'$'

abort:          ; if this program is started by dos, we display a message and abort
        push    cs
        pop     ds
        mov     dx,offset abort_string
        mov     ax,0900h
        int     21h                 ; print string
        mov     ax,4c00h
        int     21h                 ; exit to DOS

;=== This is the hardware entry point after a reset ===

        romorg  _hard_reset

starting_point:
        db      0eah
        dw      0e05bh
        dw      0f000h

;=== This is the standard location for the date-stamp ===

        romorg  _date_stamp
date_stamp label byte
        db      '--/--/--',0

; These locations are loaded with the current date by biosdate during the build process.

;======== This is the CPU type byte ==========

type_byte label byte
        db      0

; This location is loaded with the type byte value by biostype during the build process.

;=== This is where the Bios checksum value is stored =====

        db      0

; This location is loaded with the calculated checksum by biossum during the build process.

_atext  ends

        end     abort
```

THREE

The Pad File

The Pad.asm File is an assembly language file used to assign the starting offset of the Bios to a location greater than 0000 in the segment. If it is desired to reserve space at the beginning of the Bios segment for optional rom-scan modules, or to reduce the size of the Bios to less than 64 Kbytes in order to use a smaller-size prom, the biospad module is used to declare the Bios starting offset. A signature is placed at the beginning of this module to facilitate the detection of secondary and test Bios'. The signature is a representation of the word "bios" expressed by the hexadecimal characters "b105".

```
;*****************************************************************************************************
;
; Copyright (c) FOSCO 1988 - All Rights Reserved
;
; Module Name: AT Bios pad out the front of the bios
;
; Version: 1.00
;
; Author: FOSCO
;
; Date: 12-01-88
;
; Filename: atpad.asm
;
; Functional Description:
;
;  This module is used to fill out the front of the bios prom when a less than 64k bios is desired.
;  The linker always generates a 64k binary image module. The unused locations are preset to all
;  ones (FF) so that prom programmers may skip over these locations. This leaves them unused
;  and available for merging in optional rom-scan modules for an extended custom Bios.
;
; Version History:
;
;*****************************************************************************************************

          include atprfx.inc
          .code
          extrn     reset:near

bios_start proc
          dw        05b1h               ; this is a Bios signature
          db        80h
          jmp       reset
bios_start endp

          end
```

Things I Should Forget

FOUR

The Fill File

The Fill.asm File is an assembly language file used during the linking process to upwardly justify the biostop module so that it resides at the end of the Bios segment.

```
;**************************************************************************************************************
;
; Copyright (c) FOSCO 1988 - All Rights Reserved
;
; Module Name: Bios fill at link time
;
; Version: 1.00
;
; Author: FOSCO
;
; Date: 5-12-88
;
; Filename: atfill.asm
;
; Functional Description:
;
;        Uses biosfill.inc (generated by BIOSGEN) to fill out area so the biostop module will
;        be positioned correctly      at the top of the bios.
;
; Version History:
;
;*************************************************************************************************************/

include  atprfx.inc

_fill    segment

         include   atfill.inc

_fill    ends

         end
```

Meetings I Should Go To

FIVE

The Data File

The Data.asm File is an assembly language file used to declare the variables assigned to the system scratch Ram segment and those used by the SysVue program. These variables will be assigned to the top 2 Kbytes of space in the system Ram Memory map by the power-up (POD) routine.

The memory block pool definitions also reside in this file. The acquire_block and release_block operations (in Biosmisc.c) are used to assign memory for use as variables to limit use of the stack by Bios function call routines.

```
;*****************************************************************************************************
;
; Copyright (c) FOSCO 1988 - All Rights Reserved
;
; Module Name: AT Bios system data definition
;
; Version: 1.00
;
; Author: FOSCO
;
; Date: 12-01-88
;
; Filename: atdata.asm
;
; Language: MS MASM 5.1
;
; Functional Description:
;
;  This module is used to declare the variables in the system ram segment. Most of these variables
;  are used by biosvue.
;  A stack is assigned at the top of the module for use by the POD routines.
;
; Version History:
;
;*****************************************************************************************************/

        include atprfx.inc

; the data section to declare the system ram for biosvue

        public  mode_size
        public  mode_type
        public  sysvue_busy
        public  inchar
        public  buffer
        public  exit_flag
        public  buffer_index
        public  reg_saves
        public  inreg_saves
        public  outreg_saves
        public  token_index
        public  token_value
        public  cc
        public  token_buffer
        public  command_index
        public  flag_index
        public  token_number
        public  list_index
        public  last_delim
        public  break_flag
        public  last_dump_start_seg
        public  last_dump_start_off
        public  int_seg
        public  int_off
        public  reg_index
        public  break_chain
        public  trace_chain
        public  trap_chain
        public  present
        public  sum
        public  enter_seg,enter_off
        public  char_count
        public  record_type
        public    trace_count
        public    watch_flag
        public    watch_selection
        public    u_found
        public    jj
        public    redirect_flag
        public    dec_array

;=== definition of system scratch object ===

scratch segment at 0                ; Object Structure Definitions

@id     dw        ?                 ; optional id field = 5aa5 to be valid
                                    ; load this field if you need to locate
                                    ; this object in an expanded system.
; sysvue variables


sysvue_busy dw ?
mode_size dw      ?
mode_type dw      ?
inchar dw ?
buffer dw 80 dup(?)         ; line input buffer
exit_flag dw      ?                           ; -1 = exit from sysvue
buffer_index dw ?                   ; ptr for buffer
        align 16
reg_saves label word                          ; these are normal saves
        dw        16 dup (?)
inreg_saves label word                        ; these are used for INT command
        dw        16 dup(?)
outreg_saves label word                       ; these are used for INT command
        dw        16 dup(?)
token_index dw ?                    ; ptr for token string
token_value dw 8 dup(?)
cc      dw        ?
token_buffer dw 16 dup(?)           ; token string
command_index dw ?                  ; gotten from _token_index
flag_index        dw        ?
token_number dw ?                   ; nr. of matching item
list_index dw     ?
last_delim dw     ?                           ; last delimiter found
break_flag dw     ?
last_dump_start_seg dw ?
last_dump_start_off dw ?
int_seg dw        ?
int_off dw        ?
reg_index dw      ?
break_chain dd ?
trace_chain dd ?
trap_chain dd ?
present dw        ?
trace_count       dw        ?
sum     dw        ?
enter_seg dw      ?
enter_off dw      ?
char_count dw     ?
record_type dw ?
watch_flag        dw        ?
watch_selection   dw        ?
u_found   dw      ?
jj      dw        ?
redirect_flag     dw        ?
dec_array db      6 dup(?)

;============== variables for functions ============

;------ reserved space for protected mode tables -------

;----- pointer to parent bios -------

;       org       48h
        public    parent
parent  dw        ?

;------ reserved space for protected mode tables -------

        align     16
        public    GDT_block
GDT_block label byte
        dq        7 dup(?)

        public    IDT_block
IDT_block dq      0

        align     16
        public    psuedo_LIDT
```

```
psuedo_LIDT         df         0

          align     8
          public    psuedo_LGDT
psuedo_LGDT         df         0


          align     16
          public    virtual_block
virtual_block       label byte
          db        256 dup(?)


; block space for allocating
          public    start_block
          public    current_open
          public    end_block
          public    block_beg
          public    block_end

; The following area is a block-pool, from which blocks can be allocated by the functions:
;                   acquire_block
;                   release_block
; These allow dynamic allocation for recursive functions while minimizing stack space used.
          public    ds_save,bx_save,dest_save,length_mark
          public    sp_save,ss_save


          align     16

start_block         dw         ?
current_open        dw         ?
end_block dw        ?

bx_save             dw         ?
ds_save             dw         ?
dest_save dw        ?

          align     16

block_beg label     word
          dw        1024 dup(?)
block_end label     word

;==================================================

scratch ends

; This structure is used to save the old stack pointer in the new
; stack block, It is used to switch back to the callers stack
; after a swapping interrupt.

swap_seg  segment at 0
length_mark         dw         ?
                    dw         ?         ; also user ID field
sp_save             dw         ?         ; save of callers ss:sp
ss_save             dw         ?
swap_seg  ends

          end
```

Meetings I Should Avoid

SIX

The Virt File

The Virt.asm File contains the virtual mode routines.

Three major items are handled by the virtual routines:
1. Sizing and testing any extended ram
2. Block move routine which supports Virtual Ram Disks.
3. Switching machine operation to virtual mode.

Default descriptor tables are built for ram test and for the block move. The user needs to build the tables when switching to virtual mode. The test and the move routines have their own restart sequences, so that the CPU may be restarted back in real mode, and continue operation.

A restart consists of saving a restart code in the CMOS RAM at location 0F. An error code may be returned in DMA page register port 80. The values are used to determine what action to take. An optional argument may be saved in port 88. A signal is then sent to the 8042 keyboard controller which pulses the reset line to the CPU. The CPU then restarts in the default real-mode and the restart-code value is checked to see if it is other than a power-up or a normal reset switch sequence. If a valid restart code value is found in port 80, the Biostop.asm module code diverts to the specific restart sequence. This restart sequence then restores the machine state (registers) and returns control to the program which caused the CPU to go to virtual mode.

The 286 CPU normally starts operation in the real-mode upon power-up. It may be switched to virtual mode by setting the protection enable (PE) bit in the machine status word (MSW). Because of a quirk in the original design of the 286 CPU, the only way to switch back to the real-mode, (which is the normal mode for DOS), is by actually causing a hardware reset of the CPU chip. To permit this, the 8042 output port has an output line which connects to the 286's reset input line. A command may be sent to the 8042 to pulse this reset line.

The 386 has provision for switching to and from the virtual mode internally by modifying the machine status register (MSW) by a 386-specific instruction (MOV to Control Register). This enhancement of the 386 negates the need for the complicated restart handling of the 286.

One item about the 286 which is of interest is that there is a Interrupt Descriptor Table Register (IDTR) which provides the base address of the interrupt vectors. Upon a hardware reset, this register is set to a value of 0000, so that the base address of the interrupt vector segment is at zero. When switching to virtual mode, this register is changed to point to an Interrupt Descriptor Table (IDT). This permits the interrupt vector segment to be re-located without changing the contents of the vector locations. This basing register is also active in the real-mode (since it defaults to 0000, we are not aware of its existence in real-mode) and can be used to relocate or substitute multiple sets of interrupt vectors. In standard DOS practice, however, this option is not exercised.

Why the DMA page register is used to save parameters - During the restart sequence, there are usually some values that need to be passed from the virtual-mode program back to the real-mode program. These could be saved in the CMOS RAM, or somewhere in system memory. A very convenient spot that was originally used on the AT type machines, is the DMA page register chip. In the earlier PC and XT type of machines, the register was a write only device, which handled the four standard DMA channels. This chip held the high four bits of the memory address during a DMA transfer, as the 8237 DMA chip only has 16-bit internal registers. The AT type of machines have an additional 8237 DMA controller for 16-bit wide data transfers. Now there was a need for 8 page registers (one for each of the 8 DMA channels). Instead of using two of the XT type chips (74670), a newer chip was selected. This is the 74612 type of chip. This contains 16 8-bit registers, 8 of which are used for the DMA channels.
These registers are both read/write, so two of these are used for passing parameters by the AT BiosKit. The basic restart-type code is passed in location 0F of the CMOS-RAM, but an error-descriptive value is passed in the page register port at 80, and a second argument may be passed in register port 88.

Notes for potential 386/376 users - The 386, as mentioned previously, has an enhanced mechanism for switching between real and virtual modes, while the 376 assumes the virtual mode of operation from reset, since it does not have the real-mode capability.

```
;*************************************************************************************************
;
; Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
;
; Module Name: AT BiosKit Virtual memory support
;
; Version: 1.02
;
; Author: FOSCO
;
; Date: 10-20-89
;
; Filename: atvirt.asm
;
; Language MS MASM 5.1
;
; Functional Description:
;
; Version History:
; 1.01
;  Deleted some extraneous code (GDT structures)
; 1.02
;  Corrected block move errors
;  Changed restart from 8042 sequence to double fault shutdown sequence
;  Added support for restart 5 and 10
;  Changed "jmp myself" loop after shutdown to a HLT to keep busses quiet
;
;*************************************************************************************************
;
          include atprfx.inc

          extrn   outcmos:near
          extrn   GDT_block:near
          extrn   IDT_block:near
          extrn   psuedo_LGDT:near
          extrn   psuedo_LIDT:near

          extrn   co:near
          extrn   lword:near
          public  restart5
          public  restart9
          public  restart10

err_port  equ     080h      ; save stuff in page register chip
arg_port  equ     088h      ; byte for saving aux. data on restarts
port_a            equ       060h
port_b            equ       061h
inta00            equ       020h
inta01            equ       021h
```

```
intb00              equ      0a0h
intb01              equ      0a1h
status_port         equu     064h
input_buffer_full equ       02h
restart_cell         equ     08fh
restart_command equ 0feh
cmos_port equ       070h
disable_parity      equ      00ch
enable_parity       equ      0f3h
parity_error        equ      0c0h

build_24bit_address macro arg1,arg2
          push      arg2
          push      arg1
          call      build_24
          add       sp,4
          endm

build_descriptor macro       arg1,arg2
          push      arg2
          push      arg1
          call      build_descriptor_routine
          add       sp,4
          endm

; These are macros to enable or disable A20, added 5-12-89 <BG>


active              equ      0dfh
inactive  equ       0ddh

change_number = 0 ; Number of change_a20 call

empty_8042          macro
          local     empty_loop
          empty_number = empty_number + 1 ; Inc number of empty call
          xor       cx,cx
empty_loop:
          in        al,64h
          and       al,02h
          loopnz    empty_loop
          mov       al,empty_number
          endm

change_a20          macro    arg       ;; 0dfh = enable, 0ddh = disable
          local error_check,exit_change
empty_number =      0 ; Start off with empty number = 0
change_number = change_number + 10h
          empty_8042
          jnz       error_check
          mov       al,0d1h
          out       64h,al
          empty_8042
          jnz       error_check
          mov       al,arg
          out       60h,al
          empty_8042
error_check:
          clc
          jz        exit_change
          stc
          add       al,change_number
exit_change:
          endm

; End of new macros <BG>

;==== definitions for virtual modes =====
;==== (test_ram and move_block) ====

;------ Code/Data Descriptor definitions ---------------

desc      struc
lim_15_0  dw        0                  ; limit (15-0)
bas_15_0  dw        0                  ; base (15-0)
bas_23_16 db        0                  ; base (23-16)
access              db       0                   ; access rights
irsv                dw       0                   ; intel 386 reserved next 2 bytes
desc      ends

;         trap/interrupt gate definition

tigdesc struc
code_offset         dw       ?                   ; offset to routine
code_select         dw       ?                   ; selector to routine
                    db       0                   ; always zero
```

```
hb_arb              db        ?                   ; high base and access rights
                    dw        0                   ; reserved for 386
tigdesc ends

len_desc equ        8

;========== Global Descriptor Table structures ==========

;------- the GDT for the block move organization ------
; This structure order is determined by the function call format.
; Do not alter it !
; We also use this structure for the extended memory testing
move struc
move_dmy  dq 0               ; 00
move_gdt  dq 0               ; 08
move_ds             dq 0               ; 10
move_es             dq 0               ; 18
move_cs             dq 0               ; 20
move_ss             dq 0               ; 28
move_idt  dq 0               ; 30 - this exists for both move and mem-test
move_ends

idt       struc
idt_                dq 0               ; 00
idt_      ends

;--------- the GDT for virtual mode function call --------
; This structure order is determined by the function call format.
; Do not alter it !
mode struc
mode_dmy dq         0                             ; 00 - unusable
mode_gdt dq         0                             ; 08 - GDT
mode_idt dq         0                             ; 10 - IDT
mode_ds dq          0                             ; 18 - data
mode_es dq          0                             ; 20 - extra
mode_ss dq          0                             ; 28 - stack
mode_cs dq          0                             ; 30 - code
mode_bios dq        0                             ; 38 - bios
mode_ends

;---------------------------------------------------------

;---------- the gdt segment for the POD and the move ----

gdt_segment segment at 0
gdt_start label byte
          desc      <>                            ; table 0 not usable
          desc      <>                            ; 08 - GDT
          desc      <>                            ; 10 - temp for ds
          desc      <>                            ; 18 - temp for es
          desc      <>                            ; 20 - temp for ss
          desc      <>                            ; 28 - temp for cs
          desc      <>                            ; 30 - IDT - created for move
gdt_end label       byte
gdt_len = gdt_end-gdt_start
gdt_segment ends

idt_segment segment at 0
idt_start label byte
          desc      <>                            ; rom idt descriptor
idt_segment         ends

virtual_enable equ 1

null_idt: desc<>

;========== end of definitions ============

; this module includes all the code for accesssing the virtual memory and running in protected mode

;----------------------------------------------------------
          .code

;-------- move virtual block - function code 15h ----------

virtual_move proc uses ds,word_count:word,global_segment:word,global_offset:word
          cli                                     ; disable interrupts/execeptions
          cld
          xor       al,al                         ; clear the error bucket
          out       err_port,al
          change_a20 active             ; enable full range addressing <BG>
          jnc       @F                            ; no carry says A20 ok
          change_a20 inactive           ; disable it again <BG>
          mov       ax,0300h            ; mark A20 error, return to caller
          ret                                     ; quick return when A20 error
```

```
@@:
          pusha
          push      ds
          push      es
          push      40h
          pop       ds                        ; save return pointers in low ram
          assume    ds:segment_40

          mov       ax,ss                     ; save stack description in low ram
          mov       vsegment,ax
          mov       ax,sp
          mov       voffset,ax

          write_cmos restart_cell,9     ; set restart type 9

          build_descriptor es,si

          mov       cx,word_count             ; get and save the move count

          mov       ax,40h
          mov       es,ax
          mov       es, word ptr es:[0eh]     ; get system seg

          lgdt      fword ptr es:psuedo_LGDT; point the gdt-reg to the block

          lidt      fword ptr es:psuedo_LIDT     ; IDT's are built

          mov       ax,virtual_enable
          lmsw      ax                  ; go into protected mode
          delay                         ; this jump to clear queue
          db        0eah                ; jmp far ptr into cs: selector to
                                        ; set access rights
          dw        continue  ; continue executing 2 lines below
          dw        move_cs
continue:
          nop
          mov       ax,move_ss          ; set up selectors
          mov       ss,ax
          mov       ax,move_ds
          mov       ds,ax
          mov       ax,move_es
          mov       es,ax
          sub       di,di
          sub       si,si                     ; word count is already in CX
          rep       movsw                     ; this does the actual move
          in        al,port_b           ; see if a parity error on move
          test      al,parity_error
          jz        @F                        ; jump if no error
          xchg      ds:[di],ax                ; re-write the bad word
          xchg      ds:[di],ax
          mov       al,1                      ; set parity error code in bucket
          out       err_port,al
          in        al,port_b
          delay
          or        al,disable_parity   ; reset parity
          out       port_b,al
          delay
          and       al,enable_parity
          out       port_b,al
@@:
          change_a20 inactive
          jnc       @F
          in        al,err_port         ; were there any previous errors ?
          or        al,al               ; test for non-zero
          jnz       @F
          mov       al,3                ; no - set A20 error code
          out       err_port,al
@@:
ifndef    SLOW_RESET                    ; Shutdown may not work on some chipsets <BG>
; try causing a shutdown
          lidt      fword ptr cs:null_idt
          int       3
else                                    ; Use keyboard controller to reset if shutdown won't work
<BG>
          mov       al,restart_command  ; this causes restart
          out       status_port,al
endif                                   ; End of alternate reset code <BG>
          hlt                           ; stop cpu to quiet the busses

;- This is where the restart from the block moves comes in -

restart9: ; proc
          mov       ax,40h
          mov       ds,ax
          assume    ds:segment_40
```

```
        cli
        mov     ax,vsegment         ; restore the original real stack
        mov     ss,ax
        mov     ax,voffset
        mov     sp,ax
        pop     es
        pop     ds
        popa
        in      al,err_port         ; get returning error code
        mov     ah,al
        xor     al,al
        ret             ; return with error code in AH
virtual_move endp


;--------- switch to virtual mode ----------------
; Int 15, function 89
; void virtual_mode(bhbl,seg,off)
; called with bh = 8259 #1 index
; called with bl = 8259 #2 index
; called with es = segment of pointer to global desc. table
; called with si = offset of pointer to global desc. table
; returns with ax = 0 = ok, else not zero = error
; all other registers destroyed !!!!

virtual_mode proc   bhbl:word,desc_seg:word,desc_off:word
        cli
        change_a20 active
        jnc     @F
        mov     ax,-1    ; set error, quick return to caller
        ret
@@:
        mov     es,desc_seg
        mov     si,desc_off
        lgdt    fword ptr es:[si].mode_gdt

        lidt    fword ptr es:[si].mode_idt

        ; reload the 8259 interrupt controllers because we have changed the base address of the
IDT
        ; ( the interrupt vectors).

        mov     al,11h
        out     inta00,al
        delay
        mov     ax,bhbl             ; this is parm from caller
        xchg    ah,al
        out     inta01,al
        delay
        mov     al,04h
        out     inta01,al
        delay
        mov     al,01h
        out     inta01,al
        delay
        mov     al,0ffh
        out     inta01,al

        mov     al,11h
        out     intb00,al
        delay
        mov     ax,bhbl             ; this is parm from caller
        out     intb01,al
        mov     al,2
        delay
        out     intb01,al
        delay
        mov     al,1
        out     intb01,al
        delay
        mov     al,0ffh
        out     intb01,al

; build our own CS: as the bas_23_16 value for the descriptor table, This way we can execute from
; a secondary or a test bios, at other than segment F000.

        mov     ax,cs
        shr     ax,12               ; now looks as 32-16 value

        mov     es:[si].mode_cs.lim_15_0,-1
        mov     es:[si].mode_cs.bas_23_16,al
        mov     es:[si].mode_cs.bas_15_0,0
        mov     es:[si].mode_cs.access,93h
        mov     es:[si].mode_cs.irsv,0
        mov     ax,virtual_enable
        lmsw    ax
```

```
         delay                         ; this jump to clear queue
         mov      ax,mode_ds
         mov      ds,ax
         mov      ax,mode_es
         mov      es,ax
         mov      ax,mode_ss
         mov      ss,ax
         pop      bx                   ; get the return address off the stack
         add      sp,4                 ; need to adjust this !!!!
                                       ; we may have to back up through the call
                                       ; into the Int 15 function to return
                                       ; to the real caller. ????
         push     mode_cs              ; push cs selector on stack
         push     bx                   ; push return on stack
         retf                          ; return to user
virtual_mode endp

; These are the "continue" points for returning from virtual mode to
; real mode, when using the switch to virtual mode function above
; The User MUST disable both interrupts and NMI, and load V_OFFSET and
; V_SEGMENT with a valid continuation address. Then the user must
; load the restart code (5d or 10d (05h or 0ah)) in CMOS location
; xx, then cause a double fault exception to force a CPU reset. Not many
; users get involved in this.
; Programmers Opinion: If you really want to operate in virtual mode,
; get a 386 machine.

restart5:
         in       al,60h               ; purge the keyboard buffer
         mov      al,20h               ; send an EOI to the interrupt controller
         out      20h,al               ; to purge pending timer interrupts
restart10:
         mov      ax,40h               ; voffset is in segment 40:
         mov      ds,ax
         jmp      dword ptr voffset

;============================

idt_descriptor label byte
         dw       rom_idt_len
         dw       offset rom_idt
         db       0fh,0                ; resident idt table is in bios_cs
idt_descriptor_length = $-idt_descriptor
rom_idt:
         tigdesc <exc_00,move_cs,,87h>
         tigdesc <exc_01,move_cs,,87h>
         tigdesc <exc_02,move_cs,,87h>
         tigdesc <exc_03,move_cs,,87h>
         tigdesc <exc_04,move_cs,,87h>
         tigdesc <exc_05,move_cs,,87h>
         tigdesc <exc_06,move_cs,,87h>
         tigdesc <exc_07,move_cs,,87h>
         tigdesc <exc_08,move_cs,,87h>
         tigdesc <exc_09,move_cs,,87h>
         tigdesc <exc_10,move_cs,,87h>
         tigdesc <exc_11,move_cs,,87h>
         tigdesc <exc_12,move_cs,,87h>
         tigdesc <exc_13,move_cs,,87h>
         tigdesc <exc_14,move_cs,,87h>
         tigdesc <exc_15,move_cs,,87h>
         tigdesc <exc_16,move_cs,,87h>
         tigdesc <exc_17,move_cs,,87h>
         tigdesc <exc_18,move_cs,,87h>
         tigdesc <exc_19,move_cs,,87h>
         tigdesc <exc_20,move_cs,,87h>
         tigdesc <exc_21,move_cs,,87h>
         tigdesc <exc_22,move_cs,,87h>
         tigdesc <exc_23,move_cs,,87h>
         tigdesc <exc_24,move_cs,,87h>
         tigdesc <exc_25,move_cs,,87h>
         tigdesc <exc_26,move_cs,,87h>
         tigdesc <exc_27,move_cs,,87h>
         tigdesc <exc_28,move_cs,,87h>
         tigdesc <exc_29,move_cs,,87h>
         tigdesc <exc_30,move_cs,,87h>
         tigdesc <exc_31,move_cs,,87h>
rom_idt_len equ $-rom_idt

;--------------------------------

ex_int:                                ; exception interrupts for move
         mov      al,2                 ; error code = 2 = excep. int. error
         out      err_port,al
         mov      al,restart_command
         out      status_port,al
@@:      hlt
```

```
        jmp     @8


restore_idt label fword
        dw      3ffh                            ; lim_15_0
        dw      0                               ; base
        dw      0                               ;

iret_addr label word
        iret

; this sets up an all inclusive exception table. if we get an exception while we are in
; virtual mode, we want to process and report it.

sys_idt_offsets label word
        dw      offset exc_00                   ; 00 = divide error
        dw      offset exc_01                   ; 01 = single step
        dw      offset exc_02                   ; 02 = nmi
        dw      offset exc_03                   ; 03 = trap
        dw      offset exc_04                   ; 04 = into detect
        dw      offset exc_05                   ; 05 = bounds check
        dw      offset exc_06                   ; 06 = invalid opcode
        dw      offset exc_07                   ; 07 = NPX not available
        dw      offset exc_08                   ; 08 = double fault
        dw      offset exc_09                   ; 09 = NPX segment error
        dw      offset exc_10                   ; 10 = invalid tss
        dw      offset exc_11                   ; 11 = cs, ds, es not present
        dw      offset exc_12                   ; 12 = SS: not present
        dw      offset exc_13                   ; 13 = general prot error
        dw      offset exc_14                   ; 14 =
        dw      offset exc_15                   ; 15 =
        dw      offset exc_16                   ; 16 = NPX error
        dw      offset exc_17                   ; 17 =
        dw      offset exc_18                   ; 18 =
        dw      offset exc_19                   ; 19 =
        dw      offset exc_20                   ; 20 =
        dw      offset exc_21                   ; 21 =
        dw      offset exc_22                   ; 22 =
        dw      offset exc_23                   ; 23 =
        dw      offset exc_24                   ; 24 =
        dw      offset exc_25                   ; 25 =
        dw      offset exc_26                   ; 26 =
        dw      offset exc_27                   ; 27 =
        dw      offset exc_28                   ; 28 =
        dw      offset exc_29                   ; 29 =
        dw      offset exc_30                   ; 30 =
        dw      offset exc_31                   ; 31 =

;----- this is the canned gdt for the pod routines ---------

gdt_data_start label word
        desc    <0,0,0,0>             ; 00
        desc    <gdt_len,offset GDT_block,0,93h>        ; 08 - the GDT
        desc    <-1,0000h,0fh,93h>    ; 10 - temp for ds
        desc    <-1,0000h,10h,93h>    ; 18 - temp for es
        desc    <-1,0000h,00h,93h>    ; 20 - temp for cs
        desc    <-1,0000h,00h,93h>    ; 28 - temp for ss
        desc    <rom_idt_len,rom_idt,0,93h>    ; 30 - the IDT
gdt_data_end label word

;-----------------------------------------------
exc_00: mov     al,0                    ; load al with exception number
        jmp     test_exc
exc_01: mov     al,1
        jmp     test_exc
exc_02: mov     al,2
        jmp     test_exc
exc_03: mov     al,3
        jmp     test_exc
exc_04: mov     al,4
        jmp     test_exc

exc_05: push    es                      ; this is a bounds check
        mov     ax,move_es
        mov     es,ax
        sub     di,di
        mov     word ptr es:[di],0
        mov     word ptr es:[di+2],07fffh
        pop     es
        mov     al,5
        jmp     test_exc

exc_06: mov     al,6
        jmp     test_exc
exc_07: mov     al,7
```

```
          jmp      test_exc
exc_08: mov        al,8
          jmp      test_exc
exc_09: mov        al,9
          jmp      test_exc
exc_10: mov        al,10
          jmp      test_exc
exc_11: mov        al,11
          jmp      test_exc
exc_12: mov        al,12
          jmp      test_exc
exc_13: mov        al,13
          jmp      test_exc
exc_14: mov        al,14
          jmp      test_exc
exc_15: mov        al,15
          jmp      test_exc
exc_16: mov        al,16
          jmp      test_exc
exc_17: mov        al,17
          jmp      test_exc
exc_18: mov        al,18
          jmp      test_exc
exc_19: mov        al,19
          jmp      test_exc
exc_20: mov        al,20
          jmp      test_exc
exc_21: mov        al,21
          jmp      test_exc
exc_22: mov        al,22
          jmp      test_exc
exc_23: mov        al,23
          jmp      test_exc
exc_24: mov        al,24
          jmp      test_exc
exc_25: mov        al,25
          jmp      test_exc
exc_26: mov        al,26
          jmp      test_exc
exc_27: mov        al,27
          jmp      test_exc
exc_28: mov        al,28
          jmp      test_exc
exc_29: mov        al,29
          jmp      test_exc
exc_30: mov        al,30
          jmp      test_exc
exc_31: mov        al,31
          jmp      test_exc

; if we got an exception, process it so we can return an
; exception code.

test_exc:
          add      al,80h             ; add 80h to these exceptions
          out      err_port,al
          iret

;------- build a descriptor table in the system ram segment ---
;
; we do this for both the block move and the mem-test
; t_seg:t_off points to the template to build
; we will fill in some dynamic values
; it will be built in system:segment:GDT_block

;------- build the global descriptor tables --------------

; These are built with current CS: in case we are in test mode

build_descriptor_routine proc t_seg:word,t_off:word

          ;-- ds:si = seg:off of the template

          mov      ds,t_seg
          mov      si,t_off

          ;-- es:di = location in system segment to build the descriptor

          mov      di,offset GDT_block
          mov      ax,40h
          mov      es,ax
          mov      es, word ptr es:[0eh]

          ;-- cx = length of the template

          mov      cx,24                        ; template is 48 bytes long
```

```
        push    di                              ; save base address of GDT table
        rep     movsw
        pop     di

        ;-- build a dynamic GDT descriptor

        build_24bit_address es,<offset GDT_block>
        mov     es:[di].move_gdt.lim_15_0,38h
        mov     es:[di].move_gdt.bas_15_0,ax
        mov     es:[di].move_gdt.bas_23_16,dl
        mov     es:[di].move_gdt.access,93h
        mov     es:[di].move_gdt.irsv,0

        ;-- build a dynamic CS: descriptor

        build_24bit_address cs,0
        mov     es:[di].move_cs.lim_15_0,-1
        mov     es:[di].move_cs.bas_15_0,ax
        mov     es:[di].move_cs.bas_23_16,dl
        mov     es:[di].move_cs.access,9bh
        mov     es:[di].move_cs.irsv,0

        ;-- build a dynamic SS: descriptor

        build_24bit_address ss,0
        mov     es:[di].move_ss.lim_15_0,-1
        mov     es:[di].move_ss.bas_15_0,ax
        mov     es:[di].move_ss.bas_23_16,dl
        mov     es:[di].move_ss.access,93h
        mov     es:[di].move_ss.irsv,0

        ;-- build a dynamic IDT descriptor

        build_24bit_address es,<offset IDT_block>
        mov     es:[di].move_idt.lim_15_0,6
        mov     es:[di].move_idt.bas_15_0,ax
        mov     es:[di].move_idt.bas_23_16,dl
        mov     es:[di].move_idt.access,93h
        mov     es:[di].move_idt.irsv,0

        ;-- now build the IDT descriptor

        build_24bit_address cs,<offset rom_idt>
        mov     di,offset IDT_block
        mov     es:[di].idt_.lim_15_0,6
        mov     es:[di].idt_.bas_15_0,ax
        mov     es:[di].idt_.bas_23_16,dl
        mov     es:[di].idt_.access,93h
        mov     es:[di].idt_.irsv,0

        ;-- now build the psuedo GDT descriptor

        build_24bit_address es,<offset GDT_block>
        mov     di,offset psuedo_LGDT
        mov     word ptr es:[di+00],56
        mov     es:[di+02],ax
        mov     es:[di+04],dx

        ;-- now build the psuedo IDT descriptor

        build_24bit_address es,<offset IDT_block>
        mov     di,offset psuedo_LIDT
        mov     word ptr es:[di+00],256
        mov     es:[di+02],ax
        mov     es:[di+04],dx
        ret
build_descriptor_routine      endp

;-----------------------------------------------------

; returns a 24 bit address in DX:AX

build_24 proc   xseg:word, xoff:word
        xor     dx,dx
        mov     ax,xseg                 ; build 24-bit address from seg:off
        mov     dl,ah
        shr     dl,4
        shl     ax,4
        add     ax,xoff
        adc     dl,0
        ret
build_24 endp

;-----------------------------------------------------
;
;       Test Ram - size, test, and clear
```

```
;
;-----------------------------------------------------
; returns ok/error code in AX - size in cmos(30 and 31)
; note: this are all near procs, since they can be
; called from the Bios CS: only. After we do a restart,
; we are also in the Bios CS:, so we automatically connect
; with the right return address.
; The ram test is a combination of a block-move (Bios ->
; test area) followed by a block compare.
;

qmsg:     db        13,10,"A20 error on entry",0

test_ext_mem proc
          cli                           ; disable interrupts/execeptions
          cld
          mov       al,0ffh
          out       21h,al              ; mask the 8259
          pusha                         ; save all the reggies
          push      ds
          push      es
          write_cmos 30h,0    ; clear the reporting cells
          write_cmos 31h,0    ; for ext mem size
          xor       al,al
          out       err_port,al         ; clear the error reporting cell

; Note: Interesting point:
; It takes the 8042 a discrete amount of time to actually enable the A20 address line after
; this command is given. It is given in some publications as > 20 useconds. If the extended
; mem tests cuase a system crash, it may be because A20 has not switched yet, and < 1Mb ram
; is getting trashed. See the delay routines in the wrap enable/disable functions.

          change_a20 active   ; enable full range addressing <BG>
                                        ; Error numbers 51,52,53 <BG>
          jnc       @F                  ; no carry says A20 ok
          mov       ah,al               ; save number of "empty" that failed <BG>
          change_a20 inactive ; <BG>
          mov       al,ah               ; retrieve number of "empty" that failed <BG>
          out       err_port,al
          pop       es
          pop       ds
          popa
          mov       ax,0300h  ; mark A20 error, return to caller
          sti                           ; re-enable interrupts
          ret                           ; quick return when A20 error

@@:
          push      40h
          pop       ds                  ; save return pointers in low ram
          assume    ds:segment_40

          mov       ax,ss               ; save stack description in low ram
          mov       vsegment,ax
          mov       ax,sp
          mov       voffset,ax

          write_cmos restart_cell,2     ; set restart type 2

;------ build a complete GDT table set ------
; Once we do this, we can't do any interrupts, because the LIDT changes the interrput vector table
; base. Since we set it up for exceptions, we won't be able to do video interrupts.

          build_descriptor cs,<offset GDT_data_start>

          mov       ax,40h
          mov       es,ax
          mov       es, word ptr es:[0eh]         ; get system seg

          lgdt      fword ptr es:psuedo_LGDT; point the gdt-reg to the block

          lidt      fword ptr es:psuedo_LIDT      ; IDT's are built

          mov       ax,virtual_enable
          lmsw      ax                  ; goto virtual mode

          delay                         ; this jump to clear queue

          db        0eah                ; jmp far ptr into cs: selector
                                        ; to set access rights
          dw        next
          dw        move_cs

next:     delay

          mov       ax,offset move_ss
          mov       ss,ax
          mov       ax,offset move_gdt
          mov       ds,ax
          assume    ds:gdt_segment
start_loc equ       00fh
end_loc             equ       0feh
          mov       dl,start_loc         ; start at 1 Mb - 64K
          mov       ds:move_es.bas_15_0,0 ; set es: segment = xx0000
mem_check:
          inc       dl
          cmp       dl,end_loc           ; > 15 Mb boundary ?
          jae       mem_exit
          mov       ds:move_es.bas_23_16,dl ; set es: segment

          cld                            ; make sure direct is forward
          mov       ax,move_es
          mov       es,ax
          sub       di,di
          sub       si,si
          mov       cx,8000h  ; move 64k
                                         ; this preloads the block
          rep       movs word ptr es:[di],word ptr cs:[si]
          sub       di,di
          sub       si,si
          mov       cx,8000h  ; compare 64k
                                         ; this does the actual test
          repe      cmps word ptr cs:[si],word ptr es:[di]
          jcxz      mem_ok    ; it's OK
          jmp       mem_exit  ; exit
mem_ok:
          sub       di,di                ; now clear the block
          mov       cx,8000h  ; clear 64k
          xor       ax,ax
          rep       stosw
          jmp       mem_check ; check the next block
mem_exit:
          xor       dh,dh
          dec       dl                   ; bump dl down to last good segment
          sub       dx,start_loc         ; rejustify to "0000"
          shl       dx,6                 ; example 256K = 100h
          ; just return tested OK size - pod can display findings
          write_cmos 30h,dx   ; write the cmos
          xchg      dh,dl
          write_cmos 31h,dx

          change_a20 inactive ; empty numbers 71,72,73 <BG>
          jnc       @F
          mov       ah,al                ; save number of "empty" that failed <BG>
          in        al,err_port          ; were there any previous errors ?
          or        al,al                ; test for non-zero
          jnz       @F
          mov       al,ah                ; no - set A20 exit error code to
                                         ; number of failed "empty" <BG>
          out       err_port,al
@@:
ifndef    NO_SHUTDOWN                    ; Shutdown may not work on some chipsets <BG>
; try causing a shutdown
          lidt      fword ptr cs:null_idt
          int       3
else                                     ; Use keyboard controller to reset if shutdown won't work
<BG>
          mov       al,restart_command
          out       status_port,al
endif                                    ; End of alternate reset code <BG>
          hlt                            ; stop the CPU to quiet the busses
test_ext_mem endp

; This is the real return from the ext mem test

restart2  proc
          mov       ax,40h
          mov       ds,ax
          assume    ds:segment_40
          mov       ax,vsegment          ; restore the original real stack
          mov       ss,ax
          mov       ax,voffset
          mov       sp,ax
          pop       es                   ; now pop the registers
          pop       ds
          popa
          in        al,err_port          ; get returning error code
          or        al,al                ; if non-zero, return "error"
          jnz       @F
          mov       ax,ok
          sti
          ret                            ; return(ok);
@@:
```

```
          mov       ax,error
          sti
          ret                           ; return(error);
restart2  endp

;-----------------------------------------------------------
;
          end
```

Meetings I Should Call

SECTION E

The C Programs

These are the modules of the Bios which are in C. They include the Function Calls and the Interrupt Service Routines.

ONE

The Kit.h File

The Kit.h file is the 'C' header file used for all the Bios modules. The 'C' language utilities used in the build process use normal .h files since they run from normal Ram. Because the Bios is a prom-resident program, it has its own special requirements in the use of the CPU's segment registers. Declarations to provide interrupt routine support are also in this file. A special simplified interrupt sequence is provided in order to save and restore registers when either calling interrupt routines, or in defining a function to act as an interrupt routine. Because of these features included in the bioskit.h file, interrupt service routines are written as regular functions, with the machine register state being declared as unsigned integer parameters on the stack.

```
/*************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios h file of standard definitons
*
* Version: 1.02
*
* Author: FOSCO
*
* Date: 11-01-89
*
* Filename: atkit.h
*
* Language MSC 5.1
*
* Functional Description:
*
*  This file is used for the standard header for bios 'C' programs. It is the only header file
*  that should be #defined in the Bios modules. Other headers may contain conflicting definitions.
*
*        The Bios does not use any standard library for two main reasons:
*                 1. The linker attempts to place library routines at the top of the segment,
*                    which must contain some fixed code.
*
*                 2. The standard libraries assume that the program is running under DOS (which is
*                    not the case for Bios), and may make DOS references.
*
*        The file Biostop.asm contains the assembly-language functions required for the Bios.
*
* Version History:
* 1.01
*  Font revision - content unchanged
*
* 1.02
*  Defined casts for low ram variables
*
*************************************************************************************************/

unsigned peek(unsigned,unsigned);
unsigned peek40(unsigned);
unsigned peekcs(unsigned);
void poke(unsigned,unsigned,unsigned);
void poke40(unsigned,unsigned);

unsigned char peekb(unsigned,unsigned);
unsigned char peekb40(unsigned);
unsigned char peekbcs(unsigned);
unsigned peekcs(unsigned);

void pokeb(unsigned,unsigned,unsigned char);
void pokeb40(unsigned,unsigned char);
void beep(void);

void and40(unsigned,unsigned);
void andb40(unsigned,unsigned char);
void or40(unsigned,unsigned);
void orb40(unsigned,unsigned char);
void xor40(unsigned,unsigned);
void xorb40(unsigned,unsigned char);

unsigned inport(unsigned);
unsigned char inportb(unsigned);
void outport(unsigned,unsigned);
void outportb(unsigned,unsigned char);

unsigned bios_cs(void);
void set_vector(unsigned,unsigned,unsigned *);
void lbyte(unsigned char);
void lword(unsigned);
void write_string(unsigned *);
void bios_unsigned(unsigned,unsigned *,unsigned *);

void outcmos(unsigned,unsigned char);
unsigned char incmos(unsigned);
unsigned acquire_scratch_block(unsigned,unsigned,unsigned,struct callers far *);
void release_block(unsigned);
unsigned long set_timeout_count(unsigned);
void snapshot_in(unsigned, struct callers far *);
void snapshot_out(unsigned, struct callers far *);

#define true 1
#define false 0

#define yes true
#define no false

#define ok 0
#define error -1    /* error codes may be 0001-ffff */

#define at 0xfc              // type byte definition
#define xt 0xfe              // type byte definition

#define cr 13
#define lf 10
#define bspace 8

#define sys_seg_size 4096    // size of the system segment

#define BIT15  0x8000
#define BIT14  0x4000
#define BIT13  0x2000
#define BIT12  0x1000
#define BIT11  0x0800
#define BIT10  0x0400
#define BIT9   0x0200
#define BIT8   0x0100
#define BIT7   0x0080
#define BIT6   0x0040
#define BIT5   0x0020
#define BIT4   0x0010
#define BIT3   0x0008
#define BIT2   0x0004
#define BIT1   0x0002
#define BIT0   0x0001

// Define the watch bits
// These bits are also used by the block acquire/release
// function as the owners ID codes. This is for future enhancements.

#define VIDEO BIT0
#define VUE BIT1
#define CASSETTE BIT2
#define EQUIPMENT BIT3
#define MEMORY BIT4
#define LPT BIT5
#define COM BIT6
#define FLOPPY BIT7
#define HARD BIT8
#define TOD BIT9
#define BOOT BIT10
#define TIMER BIT11
#define KEYBOARD BIT12
#define PRSC BIT13
#define POD BIT14

// define the devices subject to watching - exclude some !
#define WATCH_MASK CASSETTE|EQUIPMENT|MEMORY|COM|FLOPPY|HARD|TOD|BOOT
#define carry_bit 0x0001
#define zero_bit 0x0040

#define interrupt_registers \
          unsigned es,unsigned ds, unsigned di, unsigned si,unsigned bp, unsigned sp, \
          unsigned bx,unsigned dx, unsigned cx, unsigned ax,unsigned ip, unsigned cs, unsigned flags
```

```
// This definition is the same as above except for the semi-colons in place of the commas.
// It is used for snapshots.

#define snap_registers \
          unsigned es;unsigned ds; unsigned di; unsigned si;unsigned bp; unsigned sp; \
          unsigned bx;unsigned dx; unsigned cx; unsigned ax;unsigned ip; unsigned cs; unsigned
flags;


// This is the structure of the callers save registers on the stack
// after an int. function entry

struct callers
 {
unsigned es;
unsigned ds;
unsigned di;
unsigned si;
unsigned bp;
unsigned sp;
unsigned bx;
unsigned dx;
unsigned cx;
unsigned ax;
unsigned ip;
unsigned cs;
unsigned flags;
} ;

// This definition provides a simplified means to set the interrupt vector to the service routine.

#define link_interrupt(level,name) set_vector(level,bios_cs(),name)

//---------------------------------------------------------------------------------------------

#define variables typedef struct {\
unsigned length_tag; \
unsigned user_id;\
unsigned ax;\
unsigned cx;\
unsigned dx;\
unsigned si;\
unsigned di;\
unsigned bp;\
unsigned bx;                        /* this is so we can do LDS BX */ \
unsigned ds;\
unsigned es;\
unsigned flags;\
unsigned jmp_off;\
unsigned jmp_seg;\
unsigned code_string[18];    /* reserved for code string */  \

#define end_variables unsigned char size; }

// This definition sizes the block acquire to include the variables specified
// in the variable declaration

#define acquire_block(id) acquire_scratch_block(id, \
(&myblock->size - &myblock->length_tag) + \
(16-((&myblock->size - &myblock->length_tag) % 16)))

//----------------------------------------------------
// definitions of variables in data segment segment 40h


#define VECTOR(nr)     (*((unsigned long far *)    4 * nr))
#define VECTOR_00      (*((unsigned long far *)    0x0000))
#define VECTOR_00_OFFSET  (*((unsigned far *)                0x0000))
#define VECTOR_00_SEGMENT (*((unsigned far *)                0x0002))

#define VECTOR_00      (*((unsigned long far *)    0x0000))
#define VECTOR_00_OFFSET  (*((unsigned far *)                0x0000))
#define VECTOR_00_SEGMENT (*((unsigned far *)                0x0002))

#define VECTOR_00      (*((unsigned long far *)    0x0000))
#define VECTOR_00_OFFSET  (*((unsigned far *)                0x0000))
#define VECTOR_00_SEGMENT (*((unsigned far *)                0x0002))

#define VECTOR_13      (*((unsigned long far *)    0x004c))
#define VECTOR_13_OFFSET  (*((unsigned far *)                0x004c))
#define VECTOR_13_SEGMENT (*((unsigned far *)                0x004e))

#define VECTOR_40      (*((unsigned long far *)    0x0100))
#define VECTOR_40_OFFSET  (*((unsigned far *)                0x0100))
#define VECTOR_40_SEGMENT (*((unsigned far *)                0x0102))
```

```
#define SYSTEM_SEGMENT_PTR (*((unsigned far *) 0x40e))
#define SWITCH_BYTE (*((unsigned char far *) 0x410))
#define EQUIP_FLAG (*((unsigned far *) 0x410))
#define MEMORY_SIZE (*((unsigned far *) 0x413))
#define KB_FLAG (*((unsigned char far *) 0x417))
#define KB_FLAG_1 (*((unsigned char far *) 0x418))
#define ALT_INPUT (*((unsigned char far *) 0x419))
#define BUFFER_HEAD (*((unsigned far *) 0x41a))
#define BUFFER_TAIL (*((unsigned far *) 0x41c))
#define KB_BUFFER (*((unsigned far *) 0x41e))
#define SEEK_STATUS (*((unsigned char far *) 0x43e))
#define MOTOR_STATUS (*((unsigned char far *) 0x43f))
#define MOTOR_COUNT (*((unsigned char far *) 0x440))
#define DISK_STATUS (*((unsigned char far *) 0x441))
#define CRT_MODE (*((unsigned char far *) 0x449))
#define CRT_COLS (*((unsigned far *) 0x44a))
#define CRT_LENGTH (*((unsigned far *) 0x44c))
#define CRT_START (*((unsigned far *) 0x44e))
#define CURSOR_POSN (*((unsigned far *) 0x450))
#define CURSOR_MODE (*((unsigned far *) 0x460))
#define ACTIVE_PAGE (*((unsigned char far *) 0x462))
#define ADDR_6845 (*((unsigned far *) 0x463))
#define CRT_MODE_SET (*((unsigned char far *) 0x465))
#define CRT_PALLETTE (*((unsigned char far *) 0x466))
#define V_OFFSET (*((unsigned far *) 0x467))
#define V_SEGMENT (*((unsigned far *) 0x469))
#define V_FLAG (*((unsigned char far *) 0x46b))
#define TIMER_LOW (*((unsigned far *) 0x46c))
#define TIMER_HIGH (*((unsigned far *) 0x46e))
#define TIMER_LONG (*((unsigned long far *) 0x46c))
#define TIMER_LONG_MAX 0x1800b0 /* 24 hour count */
#define TIMER_OFL (*((unsigned char far *) 0x470))
#define BIOS_BREAK (*((unsigned char far *) 0x471))
#define RESET_FLAG (*((unsigned far *) 0x472))
#define HD_STATUS1 (*((unsigned char far *) 0x474))
#define HD_NUM (*((unsigned char far *) 0x475))
#define HD_CONTROL (*((unsigned char far *) 0x476))
#define LPT_TIMEOUT_LIST (*((unsigned char far *) 0x478))
#define COMM_TIMEOUT_LIST (*((unsigned char far *) 0x47c))
#define BUFFER_START (*((unsigned far *) 0x480))
#define BUFFER_END  (*((unsigned far *) 0x482))
#define ROWS (*((unsigned char far *) 0x484))
#define POINTS (*((unsigned far *) 0x485))
#define INFO (*((unsigned char far *) 0x487))
#define INFO_3 (*((unsigned char far *) 0x488))
#define HD_STATUS (*((unsigned char far *) 0x48c))
#define HD_ERROR (*((unsigned char far *) 0x48d))
#define HD_INT_FLAG (*((unsigned char far *) 0x48e))
#define HF_CNTRL (*((unsigned char far *) 0x48f))
/* 490-493 are floppy disk drive and media type bytes */
/* 494-497 are floppy disk drive current track positions */
/* kb flags 2 and 3 must be re-located if using more than 2 floppy drives */
#define KB_FLAG_3 (*((unsigned char far *) 0x496))
#define KB_FLAG_2 (*((unsigned char far *) 0x497))
#define USER_FLAG (*((unsigned far *) 0x498))
#define USER_FLAG_SEG (*((unsigned far *) 0x49a))
#define RTC_LOW (*((unsigned far *) 0x49c))
#define RTC_HIGH (*((unsigned far *) 0x49e))
#define RTC_WAIT_FLAG (*((unsigned char far *) 0x4A0))
#define PRSC_BUSY (*((unsigned char far *) 0x500))
#define BOOT_AREA (*((unsigned char far *) 0x7c00))

/*======================= end of bioskit.h =================================*/
```

TWO

The Misc File

The Misc.c file contains functions which are used by various modules. Many of the functions used by the Setup and Watch commands in Sysvue reside in this file. The register snapshot routines are in here, as well as most of the support functions for the 8042 keyboard controller chip.

```
/****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios misc. functions
*
* Version: 1.02
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: atmisc.c
*
* Language: MS C 5.1
*
* Functional Description:
*  This module includes miscellaneous functions used by various modules.
*
* Arguments:
*
* Return:
*
* Version History:
* 1.01
*  Changed the "current video" peekb40 from 0x13 (mem size) to 0x10 (switch byte)
* 1.02
*  Added 8253 timer-based timeout support
*
****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

void eoi_sequence();

extern unsigned char type_byte;
extern unsigned char dec_array[6];
extern const unsigned char hdisk_table;
void pause(void);
unsigned get_video(void);
unsigned send_8042(unsigned);
unsigned send_8042_data(unsigned);
unsigned wait_to_rx_8042(void);
unsigned wait_to_send_8042(void);

/* L O C A L   C O N S T A N T S */

// these are the text words for the setup program

const display_list[] =
{
 "EGA",
 "CGA 40 x 25",
 "CGA 80 x 25",
 "Monochrome",
};

/* P R O G R A M S */

/*=========== Send EOI to Interrupt Controller ==========*/

void eoi_sequence() { outportb(0x20,0x20); }

/*======== Dump hex word on Console ==============*/
```

```
void lword(w) {lbyte(w >> 8); lbyte(w);}

/*======== Dump hex byte on Console ==============*/

void lbyte(outchar)
{
 unsigned char c;
 c = (outchar >> 4) & 0x0f;
 if (c > 9) c+= 7;
 co(c + '0');
 c = outchar & 0x0f;
 if (c > 9) c+= 7;
 co(c + '0');
}

/*======== Check for 0-9, a-f, A-F character =====*/

unsigned ishex(c) /* returns true or false */
{
 if ((c >= '0') && (c <= '9')) return (true);
 if ((c >= 'a') && (c <= 'f')) return (true);
 if ((c >= 'A') && (c <= 'F')) return (true);
 return(false);
}
/*======== Check for a-z character ==============*/

unsigned islower(c) /* returns true or false */
{
 if ((c >= 'a') && (c <= 'z')) return (true);
 return(false);
}
/*======== Check for a-z, A-Z character =======*/

unsigned isalpha(c) /* returns true or false */
{
 if ((c >= 'a') && (c <= 'z')) return (true);
 if ((c >= 'A') && (c <= 'Z')) return (true);
 return(false);
}
/*======== Write String on Console ==============*/

void write_string(loc)
{
 while (peekb(bios_cs(),loc) != 0) { co(peekb(bios_cs(),loc++)); }
}

/*======== Set Interrupt Vector =============*/

void set_vector(int_number,seg,off)
{
 disable();
 poke(0x00,int_number * 4,off);
 poke(0x00,(int_number * 4) + 2,seg);
 enable();
}

/*======== Read CMOS Ram ============*/

unsigned char incmos(unsigned address)
{
 outportb(0x70,address | 0x80); return(inportb(0x71));
}

/*======== Write CMOS Ram ============*/

void outcmos(unsigned address,unsigned char value)
{
 outportb(0x70,address | 0x80); outportb(0x71,value);
}


/*======= return true if (type == arg) =======*/
// If you wish to dynamically determine the kind of machine, then these functions will check
// the type byte.
// To determine CPU type, expand this function with CPU testing.

// check for motherboard type

unsigned type(unsigned int arg)
{
 if((arg == at) || (arg == xt))
 {
  if ((arg == xt) && (peekbcs(&type_byte) == xt))\
  return(true);
  if ((arg == at) && (peekbcs(&type_byte) == at))\
  return(true);
 }
 if((arg == 86) || (arg == 186) ||
 (arg == 286) || (arg == 386))// check for cpu type
 {
  return(false);
 }
 return(false);
}

//========================================================

#define swapped 1   // in myblock->status, this tells if stack is swapped

extern unsigned start_block;
extern unsigned current_open;
extern unsigned end_block;
extern unsigned watch_flag;

variables
end_variables poolregs;


// block size is passed as number of bytes needed.

unsigned acquire_scratch_block(unsigned id,unsigned block_size)
{
 poolregs *block_ptr;
 setds_system_segment();
 disable();
 if ((current_open + block_size) < end_block)
 {
  // this open space is allocatable
  block_ptr = current_open;
  current_open += block_size;
  block_ptr -> length_tag = block_size;
  block_ptr -> user_id = id;

  enable();
  if((id & (watch_flag & (WATCH_MASK))) != 0)
  {
   write_string("\n\rAcquire Block "); lword(block_ptr);
   if(id == FLOPPY) write_string(" Floppy");
   if(id == HARD) write_string(" Disk");
   if(id == BOOT) write_string(" Boot");
   if(id == CASSETTE) write_string(" Cass");
   if(id == EQUIPMENT) write_string(" Equi");
   if(id == MEMORY) write_string(" Mem");
   if(id == LPT) write_string(" Lpt");
   if(id == COM) write_string(" Com");
   if(id == TOD) write_string(" Tod");
   if(id == KEYBOARD) write_string(" Kb");
  }
  return(block_ptr);
 }
 // acquire the start block so we can flag system error
 current_open = start_block;
 sys_err(0x1111); // this is an arbitrary code
 enable();
 return(current_open); // this is an error condition
}


void release_block(poolregs *block_ptr)
{
 if((block_ptr ->user_id & ((WATCH_MASK) & watch_flag)) != 0)
 {
  write_string("\n\rRelease Block "); lword(block_ptr);
  if(block_ptr -> user_id == FLOPPY)
  write_string(" Floppy");
  if(block_ptr -> user_id == HARD) write_string(" Disk");
  if(block_ptr -> user_id == BOOT) write_string(" Boot");
  if(block_ptr -> user_id == CASSETTE)
  write_string(" Cass");
  if(block_ptr -> user_id == EQUIPMENT)
  write_string(" Equi");
  if(block_ptr -> user_id == MEMORY) write_string(" Mem");
  if(block_ptr -> user_id == LPT) write_string(" Lpt");
  if(block_ptr -> user_id == COM) write_string(" Com");
  if(block_ptr -> user_id == TOD) write_string(" Tod");
  if(block_ptr -> user_id == KEYBOARD) write_string(" Kb");

 }

 disable();
 // If you wish to zero out the released block, then a clear block function could be created here.
 // clear_block(block_ptr,block_ptr -> length_tag);

 block_ptr -> length_tag = 0;
 current_open = block_ptr;
 enable();
}

//--------- register snapshots ---------------

// this will execute if the "watch" bit for the specified device is set

void snapshot_in(unsigned device, struct callers far * frame)
{
 if(watch(device))
 {
  write_string("\n\rax="); lword(frame->ax);
  write_string(" bx="); lword(frame->bx);
  write_string(" cx="); lword(frame->cx);
  write_string(" dx="); lword(frame->dx);
  write_string(" sp="); lword(frame->sp);
  write_string(" bp="); lword(frame->bp);
  write_string(" si="); lword(frame->si);
  write_string(" di="); lword(frame->di);

  write_string("\n\rds="); lword(frame->ds);
  write_string(" es="); lword(frame->es);
  write_string(" ss="); lword(&frame->es + 1);    // this = old ss
  write_string(" cs="); lword(frame->cs);
  write_string(" ip="); lword(frame->ip);
  write_string("    ");
  if(device == FLOPPY) write_string(" FLOPPY ");
  if(device == HARD) write_string(" DISK ");
  if(device == BOOT) write_string(" BOOT ");
  if(device == CASSETTE) write_string(" CASS ");
  if(device == EQUIPMENT) write_string(" EQUI ");
  if(device == MEMORY) write_string(" MEM ");
  if(device == LPT) write_string(" LPT ");
  if(device == COM) write_string(" COM ");
  if(device == TOD) write_string(" TOD ");
  if(device == KEYBOARD) write_string(" KB ");
 }
}

void snapshot_out(unsigned device,struct callers far *frame)
{
 if(watch(device))
 {
  write_string("\n\rax="); lword(frame->ax);
  write_string(" bx="); lword(frame->bx);
  write_string(" cx="); lword(frame->cx);
  write_string(" dx="); lword(frame->dx);
  write_string(" sp="); lword(frame->sp);
  write_string(" bp="); lword(frame->bp);
  write_string(" si="); lword(frame->si);
  write_string(" di="); lword(frame->di);

  write_string("\n\rds="); lword(frame->ds);
  write_string(" es="); lword(frame->es);
  write_string(" ss="); lword(&frame->es + 1);
  write_string(" cs="); lword(frame->cs);
  write_string(" ip="); lword(frame->ip);
  write_string(" Carry="); lword(frame->flags & 0x01);
 }
}

void watch_string(unsigned device,unsigned msg)
{
 if(watch(device)) write_string(msg);
}

void watch_word(unsigned device,unsigned msg)
{
 if(watch(device)) lword(msg);
}

void watch_byte(unsigned device,unsigned msg)
{
 if(watch(device)) lbyte(msg);
}

void watch_char(unsigned device,unsigned msg)
{
 if(watch(device)) co(msg);
}

//================================================
// these will execute only in the cold-boot condition
// they will not execute for warm-boot

void cold_string(unsigned msg)
{
 if(cold()) write_string(msg);
}

void cold_word(unsigned msg)
{
 if(cold()) lword(msg);
}

void cold_byte(unsigned msg)
{
 if(cold()) lbyte(msg);
}

void cold_char(unsigned msg)
{
 if(cold()) co(msg);
}

//============ set the timer counter on boot up ============

#define seconds 0
#define minutes 2
#define hours 4

#define counts_sec 18
#define counts_min 1092
#define counts_hr 60 * counts_min

unsigned long lmul(unsigned long,unsigned);
unsigned convert_binary(unsigned char);

unsigned set_count(void)
{
 unsigned char temp;
 unsigned long count,long_temp;

 if ((temp = incmos(hours)) > 0x23) return(error);
 long_temp = convert_binary(temp);
 count = lmul(long_temp,counts_hr);

 if ((temp = incmos(minutes)) > 0x59) return(error);
 long_temp = convert_binary(temp);
 count += lmul(long_temp,counts_min);

 if ((temp = incmos(seconds)) > 0x59) return(error);
 long_temp = convert_binary(temp);
 count +=lmul(long_temp,counts_sec);

 TIMER_LONG = count;
 return(ok);
}

/*==================================================

Return the drive type for a specified drive.
drive 00-03 = floppy
drive 80-81 = hard

--------------------------------------------------*/

unsigned char cmos_drive_type(unsigned char drive_nr)
{
 switch(drive_nr)
 {
  case 0: return(incmos(0x10) >> 4);
  case 1: return(incmos(0x10) & 0x0f);
  case 2: return(incmos(0x11) >> 4);
  case 3: return(incmos(0x11) & 0x0f);
  case 0x80:
  if((incmos(0x12) >> 4) == 15) return(incmos(0x19));
  return(incmos(0x12) >> 4);

  case 0x81:
  if((incmos(0x12) & 15) == 15) return(incmos(0x1a));
  return(incmos(0x12) & 0x0f);
 }
 return(0);
}

//========== return the cmos system memory size =====
unsigned cmos_mem()
{
 unsigned temp;
 temp = incmos(0x16);
 temp = temp << 8;
 temp |= incmos(0x15);
 return(temp);
}

//========== return the cmos extended memory size =====

unsigned cmos_ext()
{
 unsigned temp;
 temp = incmos(0x18);
 temp = temp << 8;
 temp |= incmos(0x17);
 return(temp);
}

//==== return the cmos extended found memory size =====

unsigned cmos_ext_found()
{
 unsigned temp;
 temp = incmos(0x31);
 temp = temp << 8;
 temp |= incmos(0x30);
 return(temp);
}

// ---- display the time -----------

variables
end_variables timevars;

void display_time(void)
{
 timevars *myblock;
 myblock = acquire_block(VUE);
 myblock ->ax = 0x0200;
 sys_int(0x1a,myblock);
 co(((myblock->cx >> 12) & 0x0f) | '0');
 co(((myblock->cx >> 8) & 0x0f) | '0');
 co(':');
 co(((myblock->cx >> 4) & 0x0f) | '0');
 co(((myblock->cx      ) & 0x0f) | '0');
 co(':');
 co(((myblock->dx >> 12) & 0x0f) | '0');
 co(((myblock->dx >> 8) & 0x0f) | '0');
}

// ---- display the date -----------

variables
end_variables datevars;

void display_date(void)
{
 datevars *myblock;
 myblock = acquire_block(VUE);
 myblock ->ax = 0x0400;
 sys_int(0x1a,myblock);

 co(((myblock->dx >> 12) & 0x0f) | '0');
 co(((myblock->dx >> 8) & 0x0f) | '0');
 co('/');
 co(((myblock->dx >> 4) & 0x0f) | '0');
 co(((myblock->dx      ) & 0x0f) | '0');
 co('/');
 co(((myblock->cx >> 12) & 0x0f) | '0');
 co(((myblock->cx >> 8) & 0x0f) | '0');
 co(((myblock->cx >> 4) & 0x0f) | '0');
 co(((myblock->cx      ) & 0x0f) | '0');
}

// this is the text list for the floppy drive types
const drive_list[] =
{
 "No Drive",
 "360k",
 "1.2M",
 "720k",
 "1.44M",
 "Unknown", /* "2.88M could go here if implemented */
 "Unknown",
 "Unknown",
};

//------- display the disk drive configuration ------

void display_drives(void)
{
 unsigned next_hard = 'C';
 // the letter code of the next hard drive

 // find the number of floppy drives
 if((incmos(0x14) & 0x01) != 0)
 {
  if (cmos_drive_type(0) != 0)
  {
   write_string("\n\r       Drive A: ");
   write_string(peek(bios_cs(),&drive_list[cmos_drive_type(0)]));
  }

  if(((incmos(0x14) >> 6) & 0x03) > 0)
  {
   if (cmos_drive_type(1) != 0)
   {
    write_string("\n\r       Drive B: ");
    write_string(peek(bios_cs(),&drive_list[cmos_drive_type(1)]));
   }

   if(((incmos(0x14) >> 6) & 0x03) > 1)
   {
    if (cmos_drive_type(2) != 0)
    {
     write_string("\n\r       Drive C: ");
     write_string(peek(bios_cs(),&drive_list[cmos_drive_type(2)]));
    }

    if(((incmos(0x14) >> 6) & 0x03) > 2)
    {
     if (cmos_drive_type(3) != 0)
     {
      write_string("\n\r       Drive D: ");
      write_string(peek(bios_cs(),&drive_list[cmos_drive_type(3)]));
     }
    }
   }
  }
 }

 // The letters used for the disk drives can change depending on the number of floppy drives in the
 // system. Find the number of floppy drives

 if((incmos(0x14) & 0x01) != 0)
 {
  if(((incmos(0x14) >> 6) & 0x03) > 1) next_hard++;
  if(((incmos(0x14) >> 6) & 0x03) > 2) next_hard++;
 }

 if (cmos_drive_type(0x80) != 0)
 {
  write_string("\n\r       Drive "); co(next_hard);
  write_string(": Type ");

  // this should be printed in decimal !!
  lbyte(cmos_drive_type(0x80));
 }

 if (cmos_drive_type(0x81) != 0)
 {
  write_string("\n\r       Drive "); co(next_hard+1);
  write_string(": Type ");

  // this should be printed in decimal !!
  lbyte(cmos_drive_type(0x81));
 }
}

/*--------- display video type ------------*/

void display_video(void)
{
 write_string("\n\r Default Video: ");
 write_string(peek(bios_cs(),&display_list[get_video()]));

 write_string("\n\rExpected Video: ");
 write_string(peek(bios_cs(),&display_list[(incmos(0x14) >> 4) & 0x03]));

 write_string("\n\r Current Video: ");
 write_string(peek(bios_cs(),&display_list[(SWITCH_BYTE >> 4) & 0x03]));
}

/*--------- calculate cmos checksum -------------*/

unsigned calc_cmos(void)
{
 unsigned j,x,sum = 0;
 x = (incmos(0x2e) << 8) | incmos(0x2f);
 for(j=0x10;j<=0x2d;j++) sum += incmos(j);
 outcmos(0x2e,sum >> 8);
 outcmos(0x2f,sum);
 return(x);
}

/*----------- display sys mem size -----------*/

display_sys_mem()
{
 bin2dec(cmos_mem(),&dec_array);
 co(dec_array[0]);
 co(dec_array[1]);
 co(dec_array[2]);
 co(dec_array[3]);
 co(dec_array[4]);
}

/*----------- display ext mem size -----------*/

display_ext_mem()
{
 bin2dec(cmos_ext(),&dec_array);
 co(dec_array[0]);
 co(dec_array[1]);
 co(dec_array[2]);
 co(dec_array[3]);
 co(dec_array[4]);
}

/*----------- set sys mem size -----------*/

set_sys_mem(unsigned sys_size)
{
 outcmos(0x15,sys_size);
 outcmos(0x16,sys_size >> 8);
 calc_cmos();
}

/*----------- set ext mem size -----------*/

set_ext_mem(unsigned ext_size)
{
 outcmos(0x17,ext_size);
 outcmos(0x18,ext_size >> 8);
 calc_cmos();
}

/*== return the number of floppy drives in the system ==*/

unsigned number_of_floppies(void)
{
 if ((incmos(0x14) & 0x01) == 0) return(0);
 return((incmos(0x14) >> 6) + 1);
}

/*==== display hard disk table parameters ====*/

void display_hdisk_types(void)
{
 unsigned qq,jk,cyls,heads,sectors,approx_size;
 // list the table entries, calc the approximate disk size
 // use the sectors/track to accomodate the RLL drives
 // format is:
 // type,  cyls,  heads,  sectors,  size
 write_string("Type   Cylinders   Heads   Sectors  Size\n\r");

 jk = 0; qq=1;
 while(peek(bios_cs(),jk+&hdisk_table) != -1)
 {
  // display drive type
  bin2dec(qq,&dec_array);
  co(dec_array[2]);
  co(dec_array[3]);
  co(dec_array[4]);
  write_string("    ");
  // display cylinders
  cyls = peek(bios_cs(),jk+&hdisk_table);
  bin2dec(cyls,&dec_array);
  co(dec_array[0]);
  co(dec_array[1]);
  co(dec_array[2]);
  co(dec_array[3]);
  co(dec_array[4]);
  write_string("    ");

  // display heads
  heads = peekb(bios_cs(),jk+2+&hdisk_table);
  bin2dec(heads,&dec_array);
  co(dec_array[0]);
  co(dec_array[1]);
  co(dec_array[2]);
  co(dec_array[3]);
  co(dec_array[4]);

  write_string("    ");

  // display sectors
  sectors = peekb(bios_cs(),jk+14+&hdisk_table);
  bin2dec(sectors,&dec_array);
  co(dec_array[0]);
  co(dec_array[1]);
  co(dec_array[2]);
  co(dec_array[3]);
  co(dec_array[4]);

  write_string("    ");

  // display size (cyls*heads*sectors) in Mbytes

  approx_size = calc_hdisk_size(cyls,heads,sectors);

  bin2dec(approx_size,&dec_array);
  co(dec_array[0]);
  co(dec_array[1]);
  co(dec_array[2]);
  co(dec_array[3]);
  co(dec_array[4]);

  write_string("\n\r");

  qq++;
  jk = jk + 16; // look to next table entry
  if((qq & 0x000f) == 0)
  {
   pause();
   write_string("Type   Cylinders   Heads   Sectors  Size\n\r");
  }
 }
}

/*---------- pause and wait for key ---------*/

void pause(void)
{
 while(kbhit()){ci();}; // get any pending keys
 write_string("    ---- hit any key for more ----\n\r");
 while(!kbhit()){};
}

/*--------- display NPX status -----------*/

void display_npx(void)
{
 write_string("Expected: ");
 if((incmos(0x14) & 0x02) == 0)
 {
  write_string("No");
 }
 else
 {
  write_string("Yes");
 }
 write_string("    Found: ");
 if((SWITCH_BYTE & 0x02) == 0)
 {
  write_string("No");
 }
 else
 {
  write_string("Yes");
 }
```

```
}

//-------- return a long timeout count -----------

// - returns a long TIMER_LONG + count, corrects for 24 hour rollover
// - if rollover near, then delay until rollover, then return count

unsigned long get_current_timer(void)
{
return(TIMER_LONG);
}

unsigned long set_timeout_count(unsigned count)
{
 while((get_current_timer() + count) > TIMER_LONG_MAX);
 return(get_current_timer() + count);
}

//---------- keyboard controller support -------

/*===============================================/

The 8042 keyboard controller uses ports 0x60 and 0x64.
0x60 = data in and out
0x64 = status in.
The status bits are:
Bit Description
7 Parity error 0 = odd (no error), 1 = even
6 rx timeout 1 = error
5 tx timeout 1 = error
4 inhibit  0/1 = inhibit/not inhibited
3 data/command port 60 = data/command
2 system flag 0= powerup, 1= reset
1 inp bfr full 0/1 = empty/full
0 out bfr full 0/1 = empty/full

8042 commands (to port 64):

0x60 write next byte (0x64) to controller: (use 0x45)
Bit Description
7 always 0
6 1 = PC compatibility mode
5 1 = PC mode
4 1 = disable keyboard
3 1 = inhibit override
2 0 = reset system flag
1 always 0
0 1 = enable out bfr full int. (IRQ1)

0xaa self test 8042
returns 0x55 in out bfr (0x60) if OK

x0c0 read 8042 input port into out bfr (0x60)
bit Description of input port
7 0/1 = keylock switch locked/not locked
6 0/1 = CGA/MDA vidoe jumper
5 0/1 Mfg Jumper present/absent
4 0/1 system ram = 512/256 K


0xd1 write next byte (0x64) to controller
bit Description
7 kb data out line
6 kb clock out line
1 Gate A20
0 1 = reset system

/===============================================*/

//------- reset the 8042 -------

unsigned reset_8042()
{
 unsigned stat = ok;
 // mask out interrupts in case they are still enabled
 outportb(0x21,inportb(0x21) | 0x02);
 // purge the 8042's output buffer
 inportb(0x60);
 // reset the 8042
 if(wait_to_send_8042() == error) stat = error;
 if(send_8042(0xaa) == error) stat = error;
 if(wait_to_rx_8042() == error) stat = error;
 if(inportb(0x60) != 0x55) stat = error;
 // configure 8042
 if(wait_to_send_8042() == error) stat = error;
 if(send_8042(0x60) == error) stat = error;
 outportb(0x60,0x45);
 outportb(0x21,inportb(0x21) & ~0x02);
 return(stat);
}

//----- this gets the video jumper byte from the 8042 ----

// returns 2 = CGA, 3 = MDA

unsigned get_video()
{
 unsigned temp = 0;
 outportb(0x21,inportb(0x21) | 0x02);
 wait_to_send_8042();
 send_8042(0xc0);  // read the jumper command
 wait_to_rx_8042();
 temp = inportb(0x60);
 outportb(0x21,inportb(0x21) & ~0x02);
 if ((temp & 0x40) == 0) return(2);
 // return CGA index value, else
 return(3);    // return MDA index value
}

//---- wait until 8042 ready to accept another byte ----

unsigned wait_to_send_8042()
{
 unsigned long jj = set_timeout_count(20);        // 1 second delay

 do { if((inportb(0x64) & 0x02) == 0) return(ok); } while (TIMER_LONG < jj);
 return(error);
}

//--- wait for 8042 output buffer to be loaded ----

unsigned wait_to_rx_8042()
{
 unsigned long jj = set_timeout_count(20);        // 1 second delay

 do { if((inportb(0x64) & 0x01) != 0) return(ok); } while (TIMER_LONG < jj);
 return(error);
}

//--- send a command and wait until accepted ---

unsigned send_8042(value)
{
 outportb(0x64,value); return(wait_to_send_8042());
}

//--- check buffer ready and send data  ---

unsigned send_8042_data(value)
{
 unsigned trys = 10;
 while (trys-- > 0)
 {
  if(wait_to_send_8042() == ok) { outportb(0x60,value); return(ok); }
 }
 return(error);
}
```

Things Worth Making A Note Of

THREE

The Pod File

The Pod.c (Power On Diagnostics)file contains power-up diagnostics and initializing routines. This file inits some of the peripherals and also calls the setups for most of the other function calls. A general convention to develop (it has been implemented for some cases) is to examine the status returned from a function call setup routine. to determine the degree of success in initializing a device (such as video, disk, keyboard, etc.) This returned status can then be used to display an enhanced message during the POD.

```
/****************************************************************************************************
*
* Copyright (c) FOSCO 1988 - All Rights Reserved
*
* Module Name: AT bios power on diagnostics
*
* Version: 1.01
*
* Author: FOSCO
*
* Date: 12-31-88
*
* Filename: biospod.c
*
* Language: MS C 5.1
*
* Functional Description:
*
* This module does the power on diagnostics and initialization. When it finishes, it does
* a bootstrap to load the system.
*
* Arguments:
*
* Return:
*
* Version History:
*  Added NMI disable bit to the read_swtiches function.
*
****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

void biospod(void);
unsigned comm_setup(void);
unsigned lpt_setup(void);
void biospod(void);

unsigned xchg(unsigned,unsigned,unsigned);
unsigned ram_test(unsigned segment,unsigned length);
void move_system_segment(unsigned,unsigned);
unsigned char read_switches(void);
unsigned rom_check(unsigned);
void cout(unsigned char);
unsigned checksum(unsigned,unsigned,unsigned);
unsigned cold(void);

/* G L O B A L   V A R I A B L E S */

extern unsigned start_block;
extern unsigned current_open;
extern unsigned end_block;
extern unsigned block_beg;
extern unsigned block_end;

variables
end_variables podregs;

/* G L O B A L   C O N S T A N T S */

extern $iret;
extern $dummy_isr;
extern divide;
extern bios_id;
extern date_stamp;
extern bios_start;

/* L O C A L   D E F I N I T I O N S */

#define pio_port_b    0x61
#define default_equip 0x006d
#define boot_retrys 6
#define dma 0x00
#define inta01 0x21
#define timer_control_port 0x43
#define timer_counter_0_port 0x40
#define timer_counter_2_port 0x42
#define port_b 0x61
#define cr      13
#define escape  27
#define carry_bit 0x0001

/* P R O G R A M */

// Biostop module already has found 64k of ram to operate with.

void biospod(void)
{
 podregs *myblock;
 unsigned ext_mem_size = 0;
 unsigned dummy,
 rom_scan_end, /* end segment value for rom scan */
 i,j,
 cs,  /* save for code segment value */
 error_code; /* temp save for returned error codes */
 unsigned char checksum;
 unsigned video_status;  // returned from video setup
 unsigned equipment_status; // returned from equip setup
 cs = bios_cs();

 /* use top of 1st 64k for system segment */
 SYSTEM_SEGMENT_PTR = 0x1000-(sys_seg_size >> 4);

 /* set pool delimiters */
 setds_system_segment();
 start_block = current_open = &block_beg;
 end_block = &block_end;

 /* set the equipment switch flag */
 SWITCH_BYTE = read_switches();

 /* load default interrupt vectors */

 for(i=0,j=0;i<31;i++,j+=4)
 {
  poke(0x0000,j,&$iret);      /* load software defaults */
  /* override hardware defaults */
  if ((i >= 8) && (i <= 16)) poke(0x0000,j,&$dummy_isr);
  poke(0x0000,j+2,cs);
 }

 /* load default interrupt vectors for 8259 #2 */

 for(i=0,j=0x70*4;i<8;i++,j+=4)
 {
  poke(0x0000,j,&$dummy_isr);
  poke(0x0000,j+2,cs);
 }

 /* div by zero interrupt */
 poke(0x0000,0x0000,&divide);

 /*------- setup the 8255 if one exists ------------*/

 outportb(0x63,0x99); /* set the control register */
 outportb(port_b,0x70); /* disable the parity checkers */

 /*---------- setup the dma controllers ---------*/

 #define DMA1 0x00
 #define DMA2 0xc0

 outportb(DMA1+8,0x04);  // disable dma #1
 outportb(DMA2+16,0x04);  // disable dma #2

 outportb(DMA1+13,0x00);  // master clear
 outportb(DMA2+26,0x00);
outportb(DMA1+8,0x00);  // now do dma setup
outportb(DMA2+16,0x00);

outportb(DMA1+11,0x40);  // set the channel defaults
outportb(DMA2+22,0xc0);
outportb(DMA1+11,0x41);
outportb(DMA2+22,0x41);
outportb(DMA1+11,0x42);
outportb(DMA2+22,0x42);
outportb(DMA1+11,0x43);
outportb(DMA2+22,0x43);

outportb(DMA2+20,0x00);  // enable cascade of ch# 0

/*------- setup the 8259 interrupt controller -------*/

outportb(0x20,0x11); // icw1
outportb(0x21,0x08); // icw2
outportb(0x21,0x04); // icw3
outportb(0x21,0x01); // icw4
outportb(0x21,0xff); // disable all interrupts

outportb(0xa0,0x11); // icw1
outportb(0xa1,0x70); // icw2
outportb(0xa1,0x02); // icw3
outportb(0xa1,0x01); // icw4
outportb(0xa1,0xff); // disable all interrupts

/* allow any stray interrupts to be reset */
enable();

/*------- setup adapters and peripherals ------------*/


equipment_status=equipment_setup();
video_status=video_setup();

write_string("\n\r");
write_string(&bios_id);
write_string(&date_stamp);

// If a restart sequence is generated which we do not recognize, we abort to a regular re-boot,
// and display the unknown restart code.
if(incmos(0x0f) != 0) // invalid restart code
{
 write_string("\n\rInvalid Restart Code -");lbyte(incmos(0x0f));
}

cold_string("\n\rBios at............"); cold_word(bios_cs());
cold_string("\n\rLow 64K Ram Test......");
if(peek(0x00,0x412) == 0)    // check low-ram error flag
{
 cold_string("OK");
}
else
{
 cold_string("Error");
}
cold_string("\n\rVideo Function........");
cold_string("OK");
cold_string("\n\rEquipment Function....");
cold_string("OK");
cold_string("\n\rKeyboard Function.....");
if(keyboard_setup() == ok)
{
 cold_string("OK");
}
else
{
 cold_string("Error");
}
enable();
cold_string("\n\rSysVue................"); sysvue_setup();
cold_string("OK");
cold_string("\n\rMemory Size Function.."); mem_size_setup();
cold_string("OK");
cold_string("\n\rTimer Function........"); timer_setup();
cold_string("OK");
cold_string("\n\rTime of Day Function.."); time_of_day_setup();
if(set_count() == ok)  // set the timer counter
{
 cold_string("OK");
}
else
{
 cold_string("Error");
}
cold_string("\n\rPrint Screen.........."); print_screen_setup();
cold_string("OK");
cold_string("\n\rCassette Function....."); cassette_setup();
cold_string("OK");
cold_string("\n\rBootstrap Function...."); boot_setup();
cold_string("OK");

/* enable speaker */
outportb(pio_port_b,inportb(pio_port_b) & ~0x03);

/*---------- setup timer 0 for the rtc  ----------*/

outportb(timer_control_port,0x36);
outportb(timer_counter_0_port,0xff);
outportb(timer_counter_0_port,0xff);
outportb(0x21,inportb(0x21) & ~0x01); /* enable the rtc */

/*---------- setup timer 2 for the beeper  ----------*/

outportb(timer_control_port,0xb6);
outportb(timer_counter_2_port,0x33);
outportb(timer_counter_2_port,0x05);

/*-------------------------------------------------

Size and Test Ram - from 64K up to 640K

-----------------------------------------------------*/

cold_string("\n");

for (i = 64; (i < 640) && (ram_test(i << 6,32768) == ok); MEMORY_SIZE = i += 64);
{
 cold_string("\rRam Test at........ ");cold_word(i << 6);
 cold_string(" - OK   ");
}

// Now reset the sys_segment to the top of ram. The function will copy the 0f80h block to
// the system segment then reset the stack segment register to it, then do a return.

MEMORY_SIZE = (i = MEMORY_SIZE - sys_seg_size/1024);
move_system_segment(i << 6,sys_seg_size);
cold_string("\n\rSystem Segment at...");cold_word(i<<6);

/*-------------- now test ram above 1Meg --------------*/

// extended mem test returns ok/error
// if ok - size is in cmos(30 and 31)

write_string("\n");
if (test_ext_mem() == ok)
{
 ext_mem_size = incmos(0x31) << 8 | incmos(0x30);
 if(ext_mem_size == 0)
 {
  cold_string("\rExtended Ram Not Found");
 }
 else
 {
  for (i = 0;i <= ext_mem_size;i +=64)
  {
   cold_string("\rExt Ram at........ ");cold_word((i >> 4) + 0x1000);
   cold_string("0");
   cold_string(" - OK   ");
  }
 }
}
else
{
 if(inportb(0x80) < 0x80)
 {
  cold_string("\rError on A20 Control...");
 }
 else
 {
  cold_string("\rException Error .....");
  cold_byte(inportb(0x80) & ~0x80);
 }
}

/*-------- do a checksum on the rom bios prom ----------*/

cold_string("\n\rRom checksum..........");
checksum = checksum_bios_block();
```

```
if (checksum == 0) cold_string("OK");
else
{
 write_string("Error ");
 lbyte(checksum);
}

/* enable timer and kb interrupts */

outportb(0x21,inportb(0x21) & 0xfc);

/*--------- setup the comm and lpt ports --------------*/

cold_string("\n\rCOM Function.........."); comm_setup();
cold_string("OK");

cold_string("\n\rLPT Function.........."); lpt_setup();
cold_string("OK");

/*-------- check for game port ------------*/

if ((inportb(0x0201) & 0x0f) == 0 ) EQUIP_FLAG |= 0x1000;

/*------------------------------------------------

setup the floppy disk controller

-------------------------------------------------*/

cold_string("\n\rFloppy Disk...........");
if (fdisk_setup() == ok)
{
 cold_string("OK");
}
else
{
 cold_string("OK");
}

/*-----------------------------------------------------

setup the hard disk controller

;-----------------------------------------------------*/

cold_string("\n\rHard Disk Check.....");
if((error_code = hdisk_setup()) == ok)
{
 cold_string("..OK");
}
else
{
 cold_word(error_code);
}

/*-------------------------------------------------

Check for optional rom modules from c8000 -> beginning of bios in 2k increments.
A valid module has'55aa' in the first 2 locations, length indicator (length/512) in the
3rd location, and test/init. code starting in the 4th location. These modules may adjust
the mem size downward to reserve scratch ram memory.

-----------------------------------------------------*/

rom_scan_end = &bios_start;
rom_scan_end = (rom_scan_end >> 4) | 0xf000;

cold_string("\n\rScan Rom from ");
cold_word(0xc800); cold_string(" to ");
cold_word(rom_scan_end); cold_string("\n\r");

// for debugging !!
//rom_scan_end = 0xd000; // keeps from re-loading ourself
// rom-scan logic has problem with >= 64k blocks !!

for (i=0xc800;i<rom_scan_end ;i = rom_check(i));

cold_string("\n\rRom scan complete");

/*-----------------------------------------------------*/

/* re-enable the rtc in case scan module disabled it */
outportb(0x21,inportb(0x21) & ~0x01);

/*=-=-=-=-=-= now do the boot =-=-=-=-=-=-=-=-=*/
```

```
 cold_string("\n\r>-------- Booting System -----------<\n\r\n");

 myblock = acquire_block(POD);

 while (0==0) // stay in this loop forever
 {
  /* make sure kb interrupt is enabled */
  outportb(0x21,inportb(0x21) & ~0x02);
  enable();
  beep();
  write_string("\n\r");
  sys_int(0x19,myblock); // try boot from boot module
  sys_int(0x18,myblock); // try booting to SysVue
 }
 release_block(myblock);
}

/*====================================================*/

unsigned xchg(unsigned segment,
unsigned address,unsigned value)
{
 register i,j;
 i = peek(segment,address);
 poke(segment,address,value);
 j = peek(segment,address);
 poke(segment,address,i);
 return (j);
}

/*----- read the cmos dipswitch ------------*/

char read_switches()
{
 outportb(0x70,0x94);
 return(inportb(0x71));
}
/*----------------------------------------------------*/

unsigned rom_check( address)
{
 unsigned length,next_address;
 /* if not a module return next address */

 if(peek(address,0) != 0xaa55) return(address + 0x80);
 /* get length to checksum */
 length = peekb(address,2)*512;  // blocks are 512 bytes
 next_address = address + (length >> 4);

 if (checksum(address,length) == 0)
 {
  if(address != bios_cs()) // don't call ourselves !!!
  {
   cold_string("\n\rRom Signature found at : "); cold_word(address);
   far_call(address,3);
  }
 }
 return(next_address);
}

unsigned warm(void)
{
 if (RESET_FLAG == 0x1234) return(true);
 return(false);
}
unsigned cold(void)
{
 if (RESET_FLAG != 0x1234) return(true);
 return(false);
}
```

FOUR

The Equipment Driver

The Equipment Configuration Driver is an Interrupt Function Call (0x11) that is used by DOS and other programs to determine the options which are present in a system. It is called with no input parameters and returns a value in the AX register describing the system options. The POD reads the configuration dipswitch and stores this value along with an adapter configuration byte into the word at 0040:0010.

```
/**********************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios equipment function
*
* Version: 1.01
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: biosequi.c
*
* Language MS C 5.1
*
* Functional Description:
*
* Interrupt 0x11 - get equipment configuration
*
* This function call returns an equipment configuration word.
*
* The equip_flag variable is set during the power on.
*
* Returns the Equipment Word in AX:
*
* Bit Description
* 15,14 number of printers attached
* 13 = 0
* 12 = 1 = game port attached
* 11,10,9 number of rs232 cards attached
* 8 = 0
* 7,6 number of diskette drives
*  00=1, 01=2, 10=3, 11=4 only if bit 0 = 1
* 5,4 initial video mode
*  00 - no video adapter
*  01 - 40x25 bw using color card
*  10 - 80x25 bw using color card
*  11 - 80x25 bw using bw card
* 3,2 not used
* 1 0/1 = no 8087/8087
* 0 0/1 = boot from int 18/floppy disk
*
* Arguments:
* None
*
* Return:
* Equipment word in AX
*
* Version History:
* 1.01
*  Replaced peeks and pokes with casts
*
**********************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

unsigned equipment_setup(void);
void interrupt cdecl far equipment(interrupt_registers);

/* G L O B A L S */

extern _equipment;
```

```
/* L O C A L   D E F I N I T I O N S */

/* P R O G R A M */

unsigned equipment_setup()
{
 link_interrupt(0x11,&_equipment);
 return(ok);
}


void interrupt cdecl far equipment(interrupt_registers)
{
 ax = EQUIP_FLAG;
}
```

FIVE

The Memory Size Driver

The Memory Size Driver is an Interrupt (0x12) Function Call that is used by various programs (including DOS) to determine how much RAM is in the system.

```
/************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios memory size function
*
* Version: 1.01
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: atmem.c
*
* Language: MS C 5.1
*
* Functional Description:
*
* Interrupt 12h - get memory size
*
* This function call returns the amount of System RAM in AX.
* The value indicates the number of 1024 byte blocks of RAM.
*
* Arguments:
* None
*
* Return:
* Mem size in AX
*
* Version History:
* 1.01
*  Replaced peeks and pokes with casts
*
************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

unsigned mem_size_setup(void); // called by pod to set the interrupt vector to the service routine
void interrupt cdecl far mem_size(interrupt_registers); // returns the mem size in 1kbyte blocks

/* G L O B A L S */

extern _mem_size;

/* D E F I N I T I O N S */

/* P R O G R A M */

unsigned mem_size_setup()
{
 link_interrupt(0x12,&_mem_size);
 return(ok);
}

void interrupt cdecl far mem_size(interrupt_registers)
{
 ax = MEMORY_SIZE;
}
```

Things Not Worth Writing Down

SIX

The Keyboard Driver

The Keyboard Driver is used to obtain keyboard input from the keyboard.

If a non-standard keyboard or alternate input device is being used, it is suggested that the keyboard function call structure be retained. The application will then be independent of the hardware characteristics of the actual input device, and may be exercised in a development environment with the standard keyboard input device.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios keyboard driver
*
* Version: 1.06
*
* Author: FOSCO
*
* Date: 11-01-89
*
* Filename: atkb.c
*
* Language: MS C 5.1
*
* Functional Description:
*
* Arguments:
*
* Return:
*
* Version History:
* 1.01
*  Added an nmi disable (out 70 with 8x) for re-boots (ctrl-alt_del).
*
* 1.02
*  Modified buffer store to discard newest key on overrun.
*
* 1.03
*  Improved numeric keypad handling.
*
* 1.04
*   On "check for key in buffer", zeroed out only the zero bit rather than all the flag bits.
*  Disabled interrupts during the "check for" sequence to insure correct key code was returned.*
*
* 1.05
*  Added F11 and F12 support
*
* 1.06
*  Replaced peeks and pokes with casts
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

unsigned keyboard_setup(void);
void interrupt cdecl far keyboard_io(interrupt_registers);
void interrupt cdecl far keyboard_isr();
unsigned translate_from_column(unsigned char);
void reset(void);
void int1b(void);
void update_leds(void);
void disable_keyboard(void);
void enable_keyboard(void);
unsigned send_8042_data(unsigned);

/* G L O B A L   V A R I A B L E S */

/* G L O B A L   C O N S T A N T S */
```

```
extern _keyboard_io;
extern _keyboard_isr;

/* L O C A L   D E F I N I T I O N S */

#define pio_port_a 0x60
#define pio_port_b 0x61
#define status_port 0x64
#define in_bfr_full 0x02
//#define buffer_head 0x1a
//#define buffer_tail 0x1c
#define buffer_begin 0x1e
//#define buffer_start 0x80
//#define buffer_end  0x82
#define wait_for_key 0x00
#define check_for_key_in_buffer 0x01
#define get_flags 0x02

#define break_code 0x80 /* add to key to get break code */
#define tab_key 15
#define ctl_key  29
#define left_key  42
#define right_key 54
#define print_screen 55
#define alt_key  56
#define caps_key  58
#define f1 59
#define f2 60
#define f3 61
#define f4 62
#define f5 63
#define f6 64
#define f7 65
#define f8 66
#define f9 67
#define f10 68
#define f11 87
#define f12 88
#define num_key  69
#define scroll_key 70
#define num_0 82
#define num_1 79
#define num_2 80
#define num_3 81
#define num_4 75
#define num_5 76
#define num_6 77
#define num_7 71
#define num_8 72
#define num_9 73

#define minus_key 74
#define plus_key 78
#define ins_key  82
#define del_key  83
#define sys_key 84

#define ack_code  0xfa
#define resend_code 0xfe

#define ins_state 0x80
#define caps_state 0x40
#define num_state 0x20
#define scroll_state 0x10
#define alt_shift 0x08
#define ctl_shift 0x04
#define left_shift 0x02
#define right_shift 0x01

#define ins_shift 0x80
#define caps_shift 0x40
#define num_shift 0x20
#define scroll_shift 0x10
#define hold_state 0x08

#define alt_column 3
#define ctl_column 2
#define shift_column 1
#define base_column 0

/* L O C A L   C O N S T A N T S */

/*======== This is the keyboard table ===================*/


const unsigned char kb_table[89][4] ={
```

```
/* This table covers all keys.
** Some keys are pre-checked for special handling.
** Some special keys are padded out in this table.
** The format is: base,upper,ctrl,alt cases.
** The alt cases are handled as extended codes.
*/
{0  ,0  ,0  ,0 }, /* base 0 padding for indexing*/
{27 ,27 ,27 ,-1 }, /* key  1 - Escape key*/
{'1','!',-1 ,120 }, /* key  2 - '1'*/
{'2','@',0  ,121 }, /* key  3 - '2' - special handling */
{'3','#',-1 ,122 }, /* key  4 - '3'*/
{'4','$',-1 ,123 }, /* key  5 - '4'*/
{'5','%',-1 ,124 }, /* key  6 - '5'*/
{'6','^',30 ,125 }, /* key  7 - '6'*/
{'7','&',-1 ,126 }, /* key  8 - '7'*/
{'8','*',-1 ,127 }, /* key  9 - '8'*/
{'9','(',-1 ,128 }, /* key 10 - '9'*/
{'0',')',-1 ,129 }, /* key 11 - '0'*/
{'-','_',31 ,130 }, /* key 12 - '-'*/
{'=','+',-1 ,131 }, /* key 13 - '='*/
{8  ,8  ,127, -1 }, /* key 14 - backspace*/
{9  ,-1 ,-1 , -1 }, /* key 15 - tab*/
{'q','Q',17 , 16 }, /* key 16 - 'Q'*/
{'w','W',23 , 17 }, /* key 17 - 'W'*/
{'e','E',5  , 18 }, /* key 18 - 'E'*/
{'r','R',18 , 19 }, /* key 19 - 'R'*/
{'t','T',20 , 20 }, /* key 20 - 'T'*/
{'y','Y',25 , 21 }, /* key 21 - 'Y'*/
{'u','U',21 , 22 }, /* key 22 - 'U'*/
{'i','I',9  , 23 }, /* key 23 - 'I'*/
{'o','O',15 , 24 }, /* key 24 - 'O'*/
{'p','P',16 , 25 }, /* key 25 - 'P'*/
{'[','{',27 , -1 }, /* key 26 - '['*/
 {']','}',29 , -1 }, /* key 27 - ']'*/
{13 ,13 ,10 , -1 }, /* key 28 - CR*/
{-1 ,-1 ,-1 , -1 }, /* key 29 - control shift */
{'a','A',1  , 30 }, /* key 30 - 'A'*/
{'s','S',19 , 31 }, /* key 31 - 'S'*/
{'d','D',4  , 32 }, /* key 32 - 'D'*/
{'f','F',6  , 33 }, /* key 33 - 'F'*/
{'g','G',7  , 34 }, /* key 34 - 'G'*/
{'h','H',8  , 35 }, /* key 35 - 'H'*/
{'j','J',10 , 36 }, /* key 36 - 'J'*/
{'k','K',11 , 37 }, /* key 37 - 'K'*/
{'l','L',12 , 38 }, /* key 38 - 'L'*/
{';',':',-1 , -1 }, /* key 39 - ';'*/
{39,'"' ,-1 , -1 }, /* key 40 - '''*/
{'`','~',-1 , -1 }, /* key 41 - '`'*/
{-1 ,-1 ,-1 , -1 }, /* key 42 - left shift */
{92,'|',28 ,  -1 }, /* key 43 - '\'*/
{'z','Z',26 , 44 }, /* key 44 - 'Z'*/
{'x','X',24 , 45 }, /* key 45 - 'X'*/
{'c','C',3  , 46 }, /* key 46 - 'C'*/
{'v','V',22 , 47 }, /* key 47 - 'V'*/
{'b','B',2  , 48 }, /* key 48 - 'B'*/
{'n','N',14 , 49 }, /* key 49 - 'N'*/
{'m','M',13 , 50 }, /* key 50 - 'M'*/
{',','<',-1 , -1 }, /* key 51 - ','*/
{'.','>',-1 , -1 }, /* key 52 - '.'*/
{'/','?',-1 , -1 }, /* key 53 - '/'*/
{-1 ,-1 ,-1 , -1 }, /* key 54 - right shift - */
{'*',-1,114,  -1 }, /* key 55 - prt-scr - */
{-1 ,-1 ,-1,  -1 }, /* key 56 - Alt - */
{32 ,32 ,32 , 32 }, /* key 57 - space bar*/
{-1 ,-1 ,-1,  -1 }, /* key 58 - caps-lock -  */
/* function key cases*/
{59,84,94,104 }, /* key 59 - F1*/
{60,85,95,105 }, /* key 60 - F2*/
{61,86,96,106 }, /* key 61 - F3*/
{62,87,97,107 }, /* key 62 - F4*/
{63,88,98,108 }, /* key 63 - F5*/
{64,89,99,109 }, /* key 64 - F6*/
{65,90,100,110 }, /* key 65 - F7*/
{66,91,101,111 }, /* key 66 - F8*/
{67,92,102,112 }, /* key 67 - F9*/
{68,93,103,113 }, /* key 68 - F10*/
{-1,-1,-1,-1 }, /* key 69 - num-lock - */
{-1,-1,-1,-1 }, /* key 70 - scroll-lock -  */
/* num key pad - cases are base,upper,ctrl,alt*/
{71,'7',119,-1 }, /* key 71 - home*/
{72,'8',-1,-1 }, /* key 72 - cursor up*/
{73,'9',132,-1 }, /* key 73 - page up*/
{'-','-',-1,-1 }, /* key 74 - minus sign*/
{75,'4',115,-1 }, /* key 75 - cursor left*/
{-1,'5',-1,-1 }, /* key 76 - center key*/
{77,'6',116,-1 }, /* key 77 - cursor right*/
```

```
 {'+','+',-1,-1 }, /* key 78 - plus sign*/
 {79,'1',117,-1 }, /* key 79 - end*/
 {80,'2',-1,-1 }, /* key 80 - cursor down*/
 {81,'3',118,-1 }, /* key 81 - page down*/
 {82,'0',-1,-1 }, /* key 82 - insert */
 {83,'.',-1,-1 }, /* key 83 - delete */
 {-1, -1,-1,-1 }, /* key 84 - sys key */
 {-1, -1,-1,-1 }, /* key 85 */
 {-1, -1,-1,-1 }, /* key 86 */
 {133,135,137,139 }, /* key 87 - F11 */
 {134,136,138,140 }, /* key 88 - F12 */
};

/* P R O G R A M */

/*========================================*/
/*========================================*/
/*  Interrupt 16h - Keyboard function call  */
/*========================================*/
/*========================================*/

/*========================================*/

unsigned keyboard_setup(void)
{
 unsigned status = ok;
 link_interrupt(0x16,&_keyboard_io);
 link_interrupt(0x09,&_keyboard_isr);
 if(reset_8042() == error) status = error;
 BUFFER_HEAD = buffer_begin;
 BUFFER_TAIL = buffer_begin;
 BUFFER_START = buffer_begin;
 BUFFER_END = buffer_begin + 32;
 outportb(0x21,inportb(0x21) & ~0x02);
 enable_keyboard();
 //outportb(pio_port_b,inportb(pio_port_b) | 0xc0);
 /* reset the port */
 // outportb(pio_port_b,inportb(pio_port_b) & ~0x80);
 KB_FLAG = 0; // reset the flags
 KB_FLAG_1 = 0; // reset the flags
 KB_FLAG_2 = 0; // reset the flags
 KB_FLAG_3 = 0; // reset the flags
 return(status);
}

/*========================================*/

void interrupt cdecl far keyboard_io(interrupt_registers)
{
 enable ();
 switch(ax >> 8)
 {
  case wait_for_key:
  while (BUFFER_HEAD == BUFFER_TAIL)
  {
   enable();
   disable();
  };
  ax = peek40(BUFFER_HEAD); BUFFER_HEAD += 2;
  if (BUFFER_HEAD >= BUFFER_END) BUFFER_HEAD = BUFFER_START;
  enable();
  break;

  case check_for_key_in_buffer:
  disable();
  flags &= ~zero_bit;              // clear only the zero bit in the flags
  ax = peek40(BUFFER_HEAD);
  if (BUFFER_HEAD == BUFFER_TAIL) flags |= zero_bit;
  enable();
  break;

  case get_flags:
  ax = (ax & 0xff00) | KB_FLAG;
  break;
 }
}

/*====================================================*/

/* ==== keyboard hardware interrupt service routine ==== */

variables
unsigned char scan_code,scan_byte,col;
unsigned scan_word,temp;
end_variables kb_regs;
```

```
void interrupt cdecl far keyboard_isr(interrupt_registers)
{
 kb_regs *myblock;
 myblock = acquire_block(KEYBOARD);
 enable();

 disable_keyboard();
 /* get scan code from input port */
 myblock->scan_code = inportb(pio_port_a);
 myblock->scan_word = -1;  /* preset scan word to default no-load */

 switch (myblock->scan_code)
 {
  /* first do all the special handling keys */

  case ack_code:
  KB_FLAG_2 |= 0x10;
  break;

  case resend_code:
  KB_FLAG_2 |= 0x20;
  break;

  case sys_key:
  break;

  case tab_key:
  if((KB_FLAG & (left_shift | right_shift)) != 0)
  myblock->scan_word = 15*256;
  else
  myblock->scan_word = 15*256+9;
  break;

  case ctl_key:
  KB_FLAG |= ctl_shift;
  break;

  case break_code+ctl_key:
  KB_FLAG &= ~ctl_shift;
  break;

  case left_key:
  KB_FLAG |= left_shift;
  break;

  case break_code+left_key:
  KB_FLAG &= ~left_shift;
  break;

  case right_key:
  KB_FLAG |= right_shift;
  break;

  case break_code+right_key:
  KB_FLAG &= ~right_shift;
  break;

  case print_screen:
  if((KB_FLAG & ctl_shift) != 0)
  myblock->scan_word = 114*256;
  else
  {
   if((KB_FLAG & (left_shift | right_shift)) != 0)
   sys_int(0x05,myblock);
   else
   myblock->scan_word = 55*256+'*';
  }
  break;

  case break_code+print_screen: /* print screen break */
  break;

  case alt_key:
  KB_FLAG |= alt_shift;
  /* if((KB_FLAG & alt_shift) == 0) */
  ALT_INPUT = 0;
  break;

  case break_code+alt_key:
  KB_FLAG &= ~alt_shift;
  if(ALT_INPUT != 0) myblock->scan_word = ALT_INPUT;
  break;

  case caps_key:
  if ((KB_FLAG_1 & caps_shift) == 0)
  /* if key not depressed */
```

```
{
 KB_FLAG_1 |= caps_shift;
 KB_FLAG ^= caps_state;
}
break;

case break_code+caps_key:
KB_FLAG_1 &= ~caps_shift;
break;

case num_key:
if ((KB_FLAG_1 & num_shift) == 0)
/* if key not depressed */
{
 KB_FLAG_1 |= num_shift;
 KB_FLAG ^= num_state;
}
if((KB_FLAG & ctl_shift) != 0)
/* do pause mode */
{
 KB_FLAG_1 |= hold_state;
 if(CRT_MODE != 7)
 {
  outportb(0x3d8,CRT_MODE_SET);
 }
 eoi_sequence();
 enable_keyboard();
 while ((KB_FLAG_1 & hold_state) !=0) {}
}
break;

case break_code+num_key:
KB_FLAG_1 &= ~num_shift;
break;

case scroll_key:
if ((KB_FLAG_1 & scroll_shift) == 0)
/* if key not depressed */
{
 KB_FLAG_1 |= scroll_shift;
 if((KB_FLAG_1 & (ctl_shift | alt_shift)) == 0)
 {
  // don't change state if ctrl or alt keys down
  KB_FLAG ^= scroll_state;
 }
}

if((KB_FLAG & ctl_shift) != 0)
{
 eoi_sequence();
 enable_keyboard();
 if((KB_FLAG & alt_shift) != 0)
 {
  sys_int(0x18,myblock);
 }
 else /* break condition */
 {
  BIOS_BREAK = 0x80;
  sys_int(0x1b,myblock);
 }
}
break;

case break_code+scroll_key:
KB_FLAG_1 &= ~scroll_shift;
break;

case ins_key:
if ((KB_FLAG & alt_shift) != 0)
{
 assemble_alt_numeric(myblock->scan_code);
}
else
{
 if ((KB_FLAG & ins_shift) == 0)
 {
  KB_FLAG ^= ins_state; /* toggle insert state */
  KB_FLAG_1 |= ins_shift;
 }
 // If in either num_state or shift_state, but not bithe, code = '0'
 // else code = 0
 if((KB_FLAG & num_state) == 0)
 {
  if ((KB_FLAG & ( left_shift | right_shift)) == 0)
   myblock->scan_word = ins_key*256;
  else
```

```
   myblock->scan_word = ins_key*256+'0';
  }
  else
  {
   if ((KB_FLAG & (left_shift | right_shift)) != 0)
    myblock->scan_word = ins_key*256;
   else
    myblock->scan_word = ins_key*256+'0';
  }
 }
break;

case break_code+ins_key:
KB_FLAG_1 &= ~ins_shift;
break;

case minus_key:
myblock->scan_word = 74*256+'-';
break;

case plus_key:
myblock->scan_word = 78*256+'+';
break;

case del_key:
if((KB_FLAG & (ctl_shift | alt_shift)) == (ctl_shift | alt_shift))
{
 if ((KB_FLAG & (left_shift | right_shift)) == 0)
 {
  RESET_FLAG = 0x1234; /* warm boot */
 }
 else
 {
  RESET_FLAG = 0; /* cold boot */
 }
 disable();
 incmos(0x00);    // set NMI mask
 far_call(bios_cs(),&reset);
}
else
{
 if((KB_FLAG & num_state) != 0)
 {
  if((KB_FLAG & (left_shift | right_shift)) == 0)
  myblock->scan_word = 83*256+'.';
  else
  myblock->scan_word = 83*256;
 }
 else
 {
  if((KB_FLAG &
  (left_shift | right_shift)) != 0)
  myblock->scan_word = 83*256+'.';
  else
  myblock->scan_word = 83*256;
 }
}
break;

case break_code+del_key:
break;

default: /* not a special handling key */

if ((myblock->scan_code & 0x80) == 0)
/* do only the makes for these */
{
 myblock->scan_code &= 0x7f; /* clear the break bit */

 /* release the hold state if it is active */
 if((KB_FLAG_1 & hold_state) != 0) KB_FLAG_1 &= ~hold_state;
 else
 {
  switch (myblock->scan_code)
  {
   case num_0:
   case num_1:
   case num_2:
   case num_3:
   case num_4:
   case num_5:
   case num_6:
   case num_7:
   case num_8:
   case num_9:
   if ((KB_FLAG & alt_shift) != 0)
```

```
    assemble_alt_numeric(myblock->scan_code);
    else
    {
     myblock->col = 0; /* BASE = 0 (all extended codes) */
     if ((KB_FLAG & alt_shift) != 0 )
                                 myblock->col = alt_column; /* ALT = 3 (all suppressed) */
     else if ((KB_FLAG & ctl_shift) != 0)
                                 myblock->col = ctl_column; /* CTRL = 2 (all extended codes */

     if ((KB_FLAG & (num_state)) != 0)
      myblock->scan_word = peekbcs(&kb_table[myblock->scan_code][1]);
     else
      myblock->scan_word = peekbcs(&kb_table[myblock->scan_code][myblock->col])<<8;

    }
    break;

    case f1:
    case f2:
    case f3:
    case f4:
    case f5:
    case f6:
    case f7:
    case f8:
    case f9:
    case f10:
    case f11:
    case f12:
    myblock->scan_word = translate_from_column(myblock->scan_code);
    if ((myblock->scan_word & 0xff00) == 0)
    myblock->scan_word = myblock->scan_word << 8;
    break;

    default:
    myblock->scan_word = translate_from_column(myblock->scan_code);
    /* if not an extended code return */
    if((myblock->scan_word & 0xff00) == 0)
    {
     myblock->scan_byte = myblock->scan_word;
     /* caps lock correction */

     if((KB_FLAG & caps_state) != 0)
     {
      if (((myblock->scan_byte >= 'A') &&
      (myblock->scan_byte <= 'Z')) ||
      ((myblock->scan_byte >= 'a') &&
      (myblock->scan_byte <= 'z')))
      myblock->scan_byte ^= 0x20;
     }
     myblock->scan_word = (myblock->scan_code << 8) | myblock->scan_byte;
    }
    break;
   }
  }
  break;
 }
}
if(myblock->scan_word != -1) /* load scan word into buffer */
{
 //-----------------------------------------------
 // increment the temporary tail pointer
 myblock->temp = BUFFER_TAIL + 2;
 // check for wrap-around and reset to start if needed
 if (myblock->temp == BUFFER_END) myblock->temp = BUFFER_START;
 // if overwrite will not happen, then store the scan word
 if (myblock->temp != BUFFER_HEAD)
 {
  poke40(BUFFER_TAIL,myblock->scan_word);
  // now update the tail
  BUFFER_TAIL = myblock->temp;
 }
 else
 {
  beep();
 }
 //-----------------------------------------------
}
/* read the in service register to see if eoi is needed */
outportb(0x20,0x0b);
if((inportb(0x20) & 0x02) != 0)
{
 outportb(0x20,0x20);
 enable_keyboard();
}
update_leds();
```

```
 release_block(myblock);
}

/*========== end of keyboard isr routine ===========*/

unsigned translate_from_column(unsigned char scan_code)
{
 unsigned char col = 0;
 if ((KB_FLAG & alt_shift) != 0 )
 return(peekbcs(&kb_table[scan_code][alt_column]) << 8);
 else
 {
  if ((KB_FLAG & ctl_shift) != 0) col = ctl_column; /* 2 */
  else
  {
   if ((KB_FLAG & (left_shift | right_shift)) != 0)
   col = shift_column; /* 1 */
  }
 }
 return (peekbcs(&kb_table[scan_code][col]));
}

/*-------------------------------------------*/

assemble_alt_numeric(unsigned char scan_code)
{
 unsigned char code;
 code = peekbcs(&kb_table[scan_code][1]);
 if ((code >= '0') && (code <= '9'))
 (ALT_INPUT *= 10) + (code & 0x0f);
}

/*-- update the keyboard leds if state has changed --*/

void update_leds(void)
{
 if(((KB_FLAG & 0x70) >> 4) !=        (KB_FLAG_2 & 0x07))
 {
  // check update busy bit
  if ((KB_FLAG_2 & 0x40) == 0)
  {
   eoi_sequence();
   enable_keyboard();
   KB_FLAG_2 |= 0x40; // set update busy
   if(send_8042_data(0xed) == ok) // send the led command
   {
    delay_call(0,10000);      // wait about 10 millisecs
    KB_FLAG_2 &= 0xf8;  // build new led bits
    KB_FLAG_2 |= (KB_FLAG & 0x70) >> 4;
    eoi_sequence();
    enable_keyboard();
    // send the led code
    if(send_8042_data(KB_FLAG_2 & 0x07) != ok)
    {
     // send enable if we had a tx error
     send_8042_data(0xf4);
    }
   }
   KB_FLAG_2 &= ~0x40; // clear update busy
  }
  enable();
 }

}

void disable_keyboard()
{
 send_8042(0xad);
}

void enable_keyboard()
{
 send_8042(0xae);
}

/*====================================================*/
```

SEVEN

The Video Driver

The Video Driver is used to operate the CRT display.

If a video adapter is not installed in a system, it is wise to retain the video function, and internally modify it to direct output to an alternate device. This permits character oriented video output (such as the "write_tty" command) to retain PC compatibility during the development phases. As an example, the video driver might relay characters to a serial channel output routine, but the application could be exercised on a system with a video device installed.

Note: Screen flickering - some early generation video adapters do not have clean memory-write/refresh-read timing and flickering may be observed when updating the screen. Most of the currently available adapters, in our experience, seem to minimize this flickering effect. If you observe flickering, the general method to pursue to minimize this is to sample the horizontal re-trace status, (usually bit 0 of the status port, 3BA or 3DA), and write into display memory only when the adapter is in retrace (moving the blanked beam from right to left to start a new line). The vertical sync time can also be used for writing.

Note: 24 line scrolling - The standard Write-TTY function scrolls all 25 lines. A 24 line scroll may be a desirable option in order to maintain a "status" line at the bottom of the screen. This may be accomplished by modifying the Write-TTY function to either always scroll 24 (or 25) or to sense a flag (or variable) to determine how many lines should be scrolled.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios crt driver
*
* Version: 1.03
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: atcrt.c
*
* Language: MSC 5.1
*
* Functional Description:
*
* This driver manages the video display.
*
* Arguments:
*
* Return:
*
* Version History:
* 1.01
*  Corrected some get cursor position statements; added active page index to the
*  peekb40(cursor_posn+(peekb40(active_page)<<1)) uses active page # as word index
*  into cursor co-ordinates table in segment 40:50-5f.
* 1.02
*  Added some graphics improvements
* 1.03
*  Used typecast segment 40: variables
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */
```

```
#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

void $iret(void);

unsigned video_setup(void);
void interrupt cdecl far video_io(interrupt_registers);

extern void move_row (unsigned,unsigned,unsigned,unsigned);
extern void clear_row (unsigned,unsigned,unsigned,unsigned);
extern void  move_graphics_row (
unsigned,unsigned,unsigned,unsigned);
extern void clear_graphics_row (
unsigned,unsigned,unsigned,unsigned);
unsigned check_mode(void);
void load_6845_byte(unsigned,unsigned char);
void load_6845_word(unsigned,unsigned);
unsigned expand_byte(unsigned char,unsigned char);
void set_the_cursor_to_loc(unsigned);
void store_cursor_position(unsigned,unsigned);
unsigned find_position(void);
void scroll_end(void);
unsigned swap(unsigned);
unsigned position(unsigned char,unsigned char);
unsigned rom_check(unsigned);
unsigned find_page_position(unsigned char);

/* G L O B A L   V A R I A B L E S */

extern unsigned redirect_flag;

/* G L O B A L   C O N S T A N T S */

extern _video_io;

/* L O C A L   D E F I N I T I O N S */

#define cursor_posn 0x50

#define up              0
#define down            1
#define is_graphics     0x01
#define is_alpha        0x02

#define bell            0x07
#define backspace       0x08
#define carriage_return 0x0d
#define line_feed       0x0a

#define VIDEO_IO 0x10

#define color_address   0x03d4
#define mono_address    0x03b4
#define ega_address 0x03c0

/*=== command equates ===*/

#define set_mode  0
#define set_cursor_shape        1
#define set_cursor_position 2
#define read_cursor_position 3
#define read_light_pen  4
#define set_active_page         5
#define scroll_up  6
#define scroll_down  7
#define read_ac_current  8
#define write_ac_current 9
#define write_c_current  10 /* 0A */
#define set_color  11 /* 0B */
#define write_pixel             12 /* 0C */
#define read_pixel              13 /* 0D */
#define write_tty  14 /* 0E */
#define return_video_state      15 /* 0F */
#define write_string  19 /* 13 */

/* L O C A L   C O N S T A N T S */

/*------ constants ----------*/

extern const unsigned char video_font[128][8];
extern const unsigned char mode_table[8];
extern const unsigned char column_table[8];
extern const unsigned char video_parms[4][16];
```

```
// We do not use the length table in biostop.asm because it only has four entries.
const unsigned length_table[8] =
{
 2048,2048,4096,4096,16384,16384,16384,4096
};

/* P R O G R A M */

/*------------ video setup --------------------*/
variables
end_variables video_setup_regs;

unsigned video_setup(void)
{
 video_setup_regs *myblock;
 myblock = acquire_block(VIDEO);

 /* reset any video cards */
 outportb(color_address + 4, 0);
 outportb(mono_address + 4, 1);
 inportb(mono_address + 6);
 inportb(color_address + 6);
 outportb(ega_address, 0);

 /* load vector for this routine */
 link_interrupt(0x10,&_video_io);
 set_vector(0x1d,bios_cs(),&video_parms[0]);
 // set default port address to color board
 // this is done in case we have an EGA
 ADDR_6845 = color_address;

 /* determine type of video */

 switch (EQUIP_FLAG & 0x0030)
 {
  case 0x30: /* monochrome */
  ADDR_6845 = mono_address;
  myblock -> ax = 0x0002;
  sys_int(0x10,myblock);

  break;

  case 0x00: /* none - assume color */
  case 0x20: /* cga 80 */
  myblock -> ax = 0x0003;
  sys_int(0x10,myblock);
  break;

  case 0x10: /* cga 40 */
  myblock -> ax = 0x0001;
  sys_int(0x10,myblock);
  break;
 }

 rom_check(0xc000);

 release_block(myblock);

 return(ok);
}

/*===                                          ===*/
/*===             MAIN VIDEO ROUTINE           ===*/
/*===                                          ===*/

variables
unsigned i,j,reg_index,video_segment;
unsigned parm_segment,parm_offset,fill_word;
unsigned row,col,line_count;
unsigned char outchar;
unsigned temp,temp_cursor,fill_length;
end_variables video_regs;

void interrupt cdecl far video_io(interrupt_registers)
{
 video_regs *myblock;
 enable();

 myblock = acquire_block(VIDEO);

 if (ADDR_6845 != 0)
 {
  if (ADDR_6845 == mono_address)
  {
   myblock->video_segment = 0xb000; /* mono card */
  }
```

```
else
{
 myblock->video_segment = 0xb800;       /* color card */
}

switch (ax >> 8)
{

 /*===========================================*/
 /*===                                     ===*/
 /*===  AH = 0 = set mode                  ===*/
 /*===  AL = mode to be selected (0-9)     ===*/
 /*===                                     ===*/
 /*===========================================*/

 case set_mode:

 if ((SWITCH_BYTE & 0x30) == 0x30)
 {
  CRT_MODE = 7;   /* if mono, force mode 7 */
  ADDR_6845 = mono_address;
  myblock->video_segment = 0xb000;
 }
 else
 {
  ADDR_6845 = color_address;
  myblock->video_segment = 0xb800;

  if ((ax & 0xff) != 7)
  CRT_MODE = ax; /* if color, use selected mode */
  else
  {
   if ((SWITCH_BYTE & 0x30) == 0x20) CRT_MODE = 0; else CRT_MODE = 2;
  }
 }

 CRT_MODE_SET = peekbcs(&mode_table[CRT_MODE]);
 outportb((ADDR_6845) + 4, CRT_MODE_SET);

 myblock->parm_offset  = peek(0x00,0x74);
 myblock->parm_segment = peek(0x00,0x76);

 if (CRT_MODE >=2) myblock->parm_offset += 16; /* 16 */
 if (CRT_MODE >=4) myblock->parm_offset += 16; /* 16+16 */
 if (CRT_MODE ==7) myblock->parm_offset += 16; /* 16+16+16 */

 CURSOR_MODE = swap(peek(myblock->parm_segment,(myblock->parm_offset+10)));

 for (myblock->reg_index=0;myblock->reg_index < 16;myblock->reg_index++)
 {
  load_6845_byte(myblock->reg_index,peekb(myblock->parm_segment,myblock->parm_offset+myblock-
>reg_index));
 }
 CRT_START = 0;
 ACTIVE_PAGE = 0;  /* set active page to zero */

 /* enable video and correct port setting */

 outportb((ADDR_6845) + 4,peekbcs(&mode_table[CRT_MODE]));

 CRT_MODE_SET = peekbcs(&mode_table[CRT_MODE]);

 CRT_COLS = peekbcs(&column_table[CRT_MODE]);

 CRT_LENGTH = peekcs(&length_table[CRT_MODE]);

 for (myblock->i = 0; myblock->i < 8; myblock->i++)
 poke40(cursor_posn + myblock->i, 0); /* clear all cursor positions */

 if (CRT_MODE == 6) /* set pallette register */
 {
  outportb((ADDR_6845) + 5,0x3f);
  CRT_PALLETTE = 0x3f;
 }
 else
 {
  outportb((ADDR_6845) + 5,0x30);
  CRT_PALLETTE = 0x30;
 }

 if (check_mode() == is_graphics)
 myblock->fill_word = 0;
 else
 myblock->fill_word = 0x0720;

 myblock->fill_length =  2 * CRT_LENGTH;
CURSOR_MODE = 0x0607;

for (myblock->i = 0; myblock->i < myblock->fill_length; myblock->i += 2)   /* clear the screen */
{
 poke(myblock->video_segment, myblock->i, myblock->fill_word);
}

break;

/*============================================*/
/*===                                      ===*/
/*=== AH = 1 = set the cursor shape        ===*/
/*===                                      ===*/
/*===   CH = start line                    ===*/
/*===   CL = stop line                     ===*/
/*===                                      ===*/
/*============================================*/

case set_cursor_shape:

CURSOR_MODE = cx; load_6845_word(10,cx);

break;

/*======================================*/
/*===                                ===*/
/*=== AH = 2 = set cursor position   ===*/
/*===                                ===*/
/*===   DH = row            |        ===*/
/*===   DL = column         |        ===*/
/*===   BH = page           |        ===*/
/*===                       |        ===*/
/*======================================*/

case set_cursor_position:

poke40(cursor_posn+ ((bx >> 8) << 1), dx);

if (ACTIVE_PAGE == (bx >> 8))

load_6845_word(14,((CRT_START) +
       (((dx >> 8) * CRT_COLS) + (dx & 0xff) ) << 1) >>1 ));

break;

/*============================================*/
/*===                                      ===*/
/*=== AH = 3 = read cursor position        ===*/
/*===                                      ===*/
/*===   BH = page  | DH = row              ===*/
/*===          |  DL = column              ===*/
/*===          |  CX = cursor mode         ===*/
/*===          |                           ===*/
/*============================================*/

case    read_cursor_position:

cx = CURSOR_MODE; dx = CURSOR_POSN + ((bx >> 8) << 1);

break;

/*============================================*/
/*===                                      ===*/
/*=== AH = 4 = read light pen position     ===*/
/*===                                      ===*/
/*===                | AH = 0 = no info    ===*/
/*===                | AH = 1 = info       ===*/
/*===                |   DH = row          ===*/
/*===                |   DL = column       ===*/
/*===                |   CH = raster line  ===*/
/*===                |   BX = horz. position ===*/
/*===                |                     ===*/
/*============================================*/

case read_light_pen:

ax = ax & 0x00ff;   /* set no light pen return code */

break;

/*==========================================*/
/*===                                    ===*/
/*=== AH = 5 = set active page           ===*/
/*===                                    ===*/
/*===   AL = active page #      |        ===*/
/*===                           |        ===*/
```

```
/*=======================================*/

case set_active_page:

ACTIVE_PAGE = ax;
CRT_START = CRT_LENGTH * (ax & 0x00ff);
dx = peek40(cursor_posn+((ax & 0x00ff) << 1));
load_6845_word(12,CRT_START / 2);
load_6845_word(14,(CRT_START +
((((dx >> 8) * CRT_COLS ) + (dx & 0xff) ) << 1)
>>1 ));

break;

/*==========================================*/
/*===                                    ===*/
/*=== AH = 6 = scroll up                 ===*/
/*===                                    ===*/
/*===   AL = # rows to scroll (0 = blank ) ===*/
/*===   CX = row,col of upper left corner  ===*/
/*===   DX = row,col of lower right corner ===*/
/*===   BH = attribute for blanked line    ===*/
/*===                                    ===*/
/*==========================================*/

case scroll_up:
if (check_mode() == is_graphics)
{
 if ((ax & 0xff) == 0)    /* blank the field */
 {
  myblock->i = (dx >> 8) - (cx >> 8);
  for (myblock->line_count = 0; myblock->line_count <= myblock->i; myblock->line_count++)
  {
   for (myblock->row = (cx >> 8); myblock->row <= (dx >>8); myblock->row++)
   {
    clear_graphics_row(myblock->video_segment,
    ((myblock->row * 320) + (cx & 0xff)),
    (dx & 0xff) - (cx & 0xff) + 1,0);
   }
  }
 }
 else   /* scroll the field */
 {
  /* i will = the # of rows to scroll */
  myblock->i = ax & 0xff;

  for (myblock->line_count = 0;myblock->line_count < myblock->i;myblock->line_count++)
  {
   for (myblock->row = (cx >> 8); myblock->row < (dx >>8); myblock->row++)
   {
    move_graphics_row(myblock->video_segment,
    ((myblock->row + 1) * 320) + (cx & 0xff),
    (myblock->row * 320) + (cx & 0xff),
    (dx & 0xff) - (cx & 0xff)+1);
   }
   myblock->row = dx >> 8;
   clear_graphics_row(myblock->video_segment,
   (myblock->row * 320) + (cx & 0xff),
   (dx & 0xff) - (cx & 0xff) + 1,0);
  }
 }
}
else  /* alpha mode scroll */
{
 if ((ax & 0xff) == 0)    /* blank the field */
 {
  myblock->i = (dx >> 8) - (cx >> 8);
  for (myblock->line_count = 0; myblock->line_count <= myblock->i; myblock->line_count++)
  {
   for (myblock->row = (cx >> 8); myblock->row <= (dx >>8); myblock->row++)
   {
    clear_row(myblock->video_segment,
    ((myblock->row * CRT_COLS) + (cx & 0xff)) * 2,
    (dx & 0xff) - (cx & 0xff) + 1,
    (bx & 0xff00) | 0x20);
   }
  }
 }
 else   /* scroll the field */
 {
  myblock->i = ax & 0xff;
  for (myblock->line_count = 0; myblock->line_count < myblock->i; myblock->line_count++)
  {
   for (myblock->row = (cx >> 8); myblock->row < (dx >>8); myblock->row++)
   {
    move_row(myblock->video_segment,
    (((myblock->row + 1) * CRT_COLS)+(cx & 0xff))*2,
    ((myblock->row * CRT_COLS)+(cx & 0xff))*2,
    (dx & 0xff)-(cx & 0xff)+1);
   }
   myblock->row = dx >> 8;
   clear_row(myblock->video_segment,
   ((myblock->row * CRT_COLS)+(cx & 0xff))*2,
   (dx & 0xff)-(cx & 0xff)+1,
   ((bx & 0xff00) | 0x20));
  }
 }
}

break;

/*==================================================*/
/*=== AH = 7 = scroll down                      ===*/
/*===                                           ===*/
/*===   AL = # rows to scroll (0 = blank window) ===*/
/*===   CX = row,col of upper left corner        ===*/
/*===   DX = row,col of lower right corner       ===*/
/*===   BH = attribute for blanked line          ===*/
/*===                                           ===*/
/*==================================================*/

case scroll_down:

if (check_mode() == is_graphics)
{
 if ((ax & 0xff) == 0)    /* blank the field */
 {
  myblock->i = (dx >> 8) - (cx >> 8);
  for (myblock->line_count = 0; myblock->line_count <= myblock->i; myblock->line_count++)
  {
   for (myblock->row = (cx >> 8); myblock->row <= (dx >>8); myblock->row++)
   {
    clear_graphics_row(myblock->video_segment,
    ((myblock->row*320)+(cx & 0xff)),
    (dx & 0xff)-(cx & 0xff)+1,0);
   }
  }
 }
 else   /* scroll the field */
 {
  for (myblock->line_count = 0; myblock->line_count < (ax & 0xff);
  myblock->line_count++)
  {
   for (myblock->row = (dx >> 8); myblock->row > (cx >>8); myblock->row--)
   {
    move_graphics_row(myblock->video_segment,
    ((myblock->row - 1) * 320) + (cx & 0xff),
    (myblock->row * 320) + (cx & 0xff),
    (dx & 0xff)-(cx & 0xff)+1);
   }
   myblock->row = cx >> 8;
   clear_graphics_row(myblock->video_segment,
   (myblock->row * 320) + (cx & 0xff),
   (dx & 0xff)-(cx & 0xff)+1,0);
  }
 }
}
else
{
 if ((ax & 0xff) == 0)    /* blank the field */
 {
  for (myblock->row = (dx >> 8); myblock->row < (cx >>8); myblock->row--)
  {
   for (myblock->col=(cx & 0xff); myblock->col< (dx & 0xff); myblock->col++)
   {
    pokeb(myblock->video_segment,
    (((myblock->row * CRT_COLS) + myblock->col)*2),0x20);
   }
  }
 }
 else   /* scroll the field */
 {
  for (myblock->line_count = 0; myblock->line_count < (ax & 0xff);
  myblock->line_count++)
  {
   for (myblock->row = (dx >> 8); myblock->row > (cx >>8); myblock->row--)
   {
    move_row(myblock->video_segment,
    (((myblock->row - 1) * CRT_COLS)+(cx & 0xff))*2,
    ((myblock->row * CRT_COLS)+(cx & 0xff))*2,
    (dx & 0xff)-(cx & 0xff)+1);
   }
```

```
    myblock->row = cx >> 8;

    clear_row(myblock->video_segment,
    ((myblock->row * CRT_COLS + (cx & 0xff))*2),
    (dx & 0xff)-(cx & 0xff)+1,
    ((bx & 0xff00) | 0x20));
   }
  }
 }

 break;

 /*============================================*/
 /*===                                      ===*/
 /*=== AH = 8 = read attr and char at cursor ===*/
 /*===                                      ===*/
 /*===   BH = page #        | AL = char     ===*/
 /*===    for alpha only    | AH = attrib   ===*/
 /*============================================*/

 case read_ac_current:

 if (check_mode() == is_graphics)
 {
  for (myblock->j = 0; myblock->j < 128; myblock->j++)
  {
   myblock->temp = true;
   for (myblock->i=0; myblock->i <= 7; myblock->i++)
   {
    /* if mode is 640 x 200 */
    if(CRT_MODE == 6)
    {
     if(peekb(myblock->video_segment,
     (myblock->row * 320) + ((myblock->i & 0xfe) * 40) + myblock->col +
     ((myblock->i & 1) * 0x2000)) != peekbcs(&video_font[ax & 0x007f][myblock->i]))
     myblock->temp = false;
    }
    else /* 320 x 200 mode */
    {
     if(peek(myblock->video_segment,
     (myblock->row * 320) + ((myblock->i & 0xfe) * 40) + (myblock->col * 2) + ((myblock->i & 1) *
0x2000)) !=
     expand_byte(peekbcs(&video_font[ax & 0x007f][myblock->i]),3))
     myblock->temp = false;
    }
    if (myblock->temp == true)
    {
     ax = (ax & 0xff00) | myblock->j;
     break;  /* get out with the find */
    }
   }
  }
 }
 else                          /* --- alpha read --- */
 {
  ax = peek(myblock->video_segment,find_page_position(bx >> 8));
 }

 break;

 /*============================================*/
 /*===                                      ===*/
 /*=== AH = 9 = write attr and char at cursor ===*/
 /*===                                      ===*/
 /*===   BH = page #    |                   ===*/
 /*===     for alpha only|                  ===*/
 /*===   CX = # write   |                   ===*/
 /*===   AL = character |                   ===*/
 /*===   BL = attribute |                   ===*/
 /*===                  |                   ===*/
 /*============================================*/

 case write_ac_current:

 if(check_mode() == is_graphics)
 {
  myblock->row = peekb40(cursor_posn+((bx >> 8) << 1) +1);
  myblock->col = peekb40(cursor_posn+((bx >> 8) << 1));
  for (myblock->i=0; myblock->i <= 7; myblock->i++)
  {
   /* if the mode is 640 x 200 */
   if(CRT_MODE == 6)
   {
    pokeb(myblock->video_segment,
    (myblock->row * 320) + ((myblock->i & 0xfe) * 40) +
    myblock->col +
   ((myblock->i & 1) * 0x2000),
   peekbcs(&video_font[ax & 0x007f][myblock->i]));
  }
  else
  {

   /* if the mode is 320 x 200 */

   poke(myblock->video_segment,
   (myblock->row * 320) + ((myblock->i & 0xfe) * 40) +
   (myblock->col * 2) +
   ((myblock->i & 1) * 0x2000),
   expand_byte(peekbcs(&video_font[ax & 0x007f][myblock->i]),
   bx & 0x83));
  }
 }
}
else
{
 myblock->temp = find_page_position(bx >> 8);
 for (myblock->i = 0; myblock->i < cx; myblock->i++)
 {
  poke(myblock->video_segment,myblock->temp+(2 * myblock->i),((bx << 8) | (ax & 0xff)));
 }
}

break;

/*==============================================*/
/*===                                        ===*/
/*=== AH = 10 = write char at cursor position ===*/
/*===                                        ===*/
/*===   BH = page #                          ===*/
/*===     for alpha only|                    ===*/
/*===   CX = # to write                      ===*/
/*===   AL = character                       ===*/
/*===                                        ===*/
/*==============================================*/

case    write_c_current:

if(check_mode() == is_graphics)
{
 myblock->row = peekb40(cursor_posn+((bx >> 8) << 1)+1);
 myblock->col = peekb40(cursor_posn+((bx >> 8) << 1));
 for (myblock->i=0; myblock->i <= 7; myblock->i++)
 {
  /* if the mode is 640 x 200 */
  if(CRT_MODE == 6)
  {
   pokeb(myblock->video_segment,
   (myblock->row * 320) + ((myblock->i & 0xfe) * 40) +
   myblock->col +
   ((myblock->i & 1) * 0x2000),
   peekbcs(&video_font[ax & 0x007f][myblock->i]));
  }
  else
  {

   /* if the mode is 320 x 200 */

   poke(myblock->video_segment,
   (myblock->row * 320) + ((myblock->i & 0xfe) * 40) +
   (myblock->col * 2) +
   ((myblock->i & 1) * 0x2000),
   expand_byte(peekbcs(&video_font[ax & 0x007f][myblock->i]),3));
  }
 }
}
else
{
 myblock->temp = find_page_position(bx >> 8);
 for (myblock->i = 0; myblock->i < cx; myblock->i++)
 {
  pokeb(myblock->video_segment,myblock->temp + (2 * myblock->i),ax);
 }
}

break;

/*==============================================*/
/*===                                        ===*/
/*=== AH = 11 = set color for MRES graphics  ===*/
/*===                                        ===*/
/*===   BH = 0                               ===*/
/*===        BL = background color           ===*/
```

```
/*===                                          ===*/
/*===   BH = 1                                 ===*/
/*===        BL = pallette select              ===*/
/*===                                          ===*/
/*==============================================*/

case set_color:
/* get current pallette value */
if((bx >> 8) == 0) /* this is color 0 */
{
 /* turn off low 5 bits of current color */
 CRT_PALLETTE &= 0xe0;
 CRT_PALLETTE |= (bx & 0x1f);
 /* turn off high three bits of input */
}
else /* select pallette */
{
 CRT_PALLETTE &= 0xdf;  /* turn off pallete select bit */
 if((bx & 0x01) != 0)
 CRT_PALLETTE |= 0x20; /* turn on pallette select bit */
}
outportb((ADDR_6845) + 5,CRT_PALLETTE);
break;

/*===========================================*/
/*===                                     ===*/
/*=== AH = 12 = write pixel               ===*/
/*===                                     ===*/
/*===   DX = row 0-199                    ===*/
/*===   CX = column 0-639                 ===*/
/*===   AL = pixel value (1 or 2 bits)    ===*/
/*===      bit 7 = 1 = XOR lower bits     ===*/
/*===                                     ===*/
/*===========================================*/

case write_pixel:

/* calc row address for even/odd */
myblock->row = dx * 40 + ((dx & 1) * (0x2000-40));
if(CRT_MODE == 6)     /* write one 1 bit */
{
 /* col variable is used for bit mask */
 myblock->col = 1 << (cx % 8);
 /* calc address for 1 bit in 8 */
 myblock->row += cx / 8;

 pokeb(myblock->video_segment,myblock->row,((peekb(myblock->video_segment,myblock->row)
 & ~myblock->col) |
 ((ax & 0x0001) << (cx % 8))));

}
else                            /* write 2 bits */
{
 /* col variable is used for bit mask */
 myblock->col = 3 << (cx % 4);

 /* calc address for 2 bits in 8 */
 myblock->row += cx/4;

 pokeb(myblock->video_segment,myblock->row,((peekb(myblock->video_segment,myblock->row)
 & ~myblock->col) |
 ((ax & 0x0003) << (cx % 4))));

}

break;

/*==========================================*/
/*===                                    ===*/
/*=== AH = 13 = read pixel               ===*/
/*===                                    ===*/
/*===   DX = row 0-199     | AL = pixel  ===*/
/*===   CX = column 0-639  |             ===*/
/*===                      |             ===*/
/*==========================================*/

case read_pixel:

/* calc row address for even/odd */
myblock->row = dx * 40 + ((dx & 1) * (0x2000-40));


if(CRT_MODE == 6)     /* read one 1 bit */
{

 /* calc address for 1 bit in 8 */
 myblock->row += cx/8;

 ax = (ax & 0xff00) |
 ((peekb(myblock->video_segment,myblock->row) >> (cx % 8)) & 1);

}
else                              /* read 2 bits */
{

 /* calc address for 2 bits in 8 */
 myblock->row += cx/4;

 ax = (ax & 0xff00) |
 ((peekb(myblock->video_segment,myblock->row) >> (cx % 4)) & 3);

}

break;

/*=======================================*/
/*===                                 ===*/
/*=== AH = 14 = write tty             ===*/
/*===                                 ===*/
/*===   AL = character to write       ===*/
/*===   BL = fground color in graphics ===*/
/*===   use active page               ===*/
/*=======================================*/

case write_tty:

if(redirect_flag != 0)
{
 myblock -> ax = ax & 0x00ff;
 myblock -> dx = 0;
 sys_int(0x17,myblock);
}
/* get current cursor */
myblock->row = peekb40(cursor_posn + (ACTIVE_PAGE << 1)+1);
myblock->col = peekb40(cursor_posn + (ACTIVE_PAGE << 1));

switch (ax & 0x00ff)
{
 case bell:
 beep();
 break;

 case backspace:
 if (myblock->col !=0) myblock->col--;
 break;

 case carriage_return:
 myblock->col = 0;
 break;

 case line_feed:
 if (myblock->row == 24)
 {
  /* find the fill attribute */
  myblock -> ax = 0x0800;
  // set bh to active page
  myblock -> bx = ACTIVE_PAGE;
  myblock -> bx = myblock -> bx << 8;
  sys_int(VIDEO_IO,myblock);
  // set bh to blank line attribute
  myblock -> bx = (myblock -> ax & 0xff00);

  myblock -> ax = 0x0601;   /* scroll call */
  myblock -> cx = 0;
  myblock -> dx = (24 << 8) | (CRT_COLS - 1);
  sys_int(VIDEO_IO,myblock);
 }
 else
 {
  myblock->row++;
 }

 break;

 default:   /* any other character */

 myblock -> ax = (write_c_current << 8) | (ax & 0x00ff);
 myblock -> bx = ACTIVE_PAGE;
 myblock -> bx = myblock -> bx << 8;
 myblock -> cx = 1;
 sys_int(VIDEO_IO,myblock);
 myblock->col++;
 if (myblock->col >= CRT_COLS)
 {
  myblock->col = 0;
  if (myblock->row == 24)
  {
   /* find the fill attribute */
   myblock -> ax = 0x0800;
   myblock -> bx = ACTIVE_PAGE;
   myblock -> bx = myblock -> bx << 8;
   sys_int(VIDEO_IO,myblock);
   myblock -> bx = (myblock -> ax & 0xff00);

   myblock -> ax = 0x0601;
   myblock -> cx = 0;
   myblock -> dx = (24 << 8) | (CRT_COLS - 1);
   sys_int(VIDEO_IO,myblock);
  }
  else
  {
   myblock->row++;
  }
 }
 break;
}

myblock->temp = (myblock->row << 8) | (myblock->col & 0x00ff);
poke40(cursor_posn+((ACTIVE_PAGE) << 1),myblock->temp);

load_6845_word(14,(CRT_START +
((myblock->row * CRT_COLS) + myblock->col ) << 1) >>1 );

break;

/*=========================================*/
/*===                                   ===*/
/*=== AH = 15 = return video state      ===*/
/*===                                   ===*/
/*===   AH = # of cols                  ===*/
/*===   AL = mode                       ===*/
/*===   BH = page                       ===*/
/*===                                   ===*/
/*=========================================*/

case return_video_state:

ax = (CRT_COLS << 8) | CRT_MODE;
bx = (bx & 0x00ff) | (ACTIVE_PAGE << 8);

break;

/*=================================================*/
/*===                                           ===*/
/*=== AH = 19 = write string                    ===*/
/*===                                           ===*/
/*===   ES:BP = pointer to string               ===*/
/*===   CX = length of string                   ===*/
/*===   DX = cursor position                    ===*/
/*===   BH = page #                             ===*/
/*===                                           ===*/
/*===    AL = 0 = write string - don't move cursor ===*/
/*===  BL = attribute                           ===*/
/*===    AL = 1 = write string - move cursor    ===*/
/*===  BL = attribute                           ===*/
/*===    AL = 2 = write char and attribute      ===*/
/*===    - don't move cursor                    ===*/
/*===    AL = 3 = write char and attribute      ===*/
/*===    - move cursor                          ===*/
/*===                                           ===*/
/*=================================================*/

case write_string:

/* if count > 0 and opcode 0-3 */
if ((cx != 0) && ((ax & 0x00ff) <= 3))
{
 /* save current cursor for this page */
 myblock->temp_cursor = peek40(cursor_posn+(bx >> 8)<<1);

 /* set cursor to callers position */
 myblock -> ax = (set_cursor_position << 8);
 myblock -> dx = dx;
 myblock -> bx = bx;
 sys_int(VIDEO_IO,myblock);
 myblock->row = myblock -> dx >> 8;
 myblock->col = myblock ->dx & 0x00ff;
```

```
      for (myblock->i = 0; myblock->i < cx; myblock->i++)
      {
       myblock->outchar = peekb(es,bp+myblock->i);

       if ((myblock->outchar == backspace) ||
       (myblock->outchar == carriage_return) ||
       (myblock->outchar == line_feed))
       {
        myblock -> ax = (write_tty << 8) | myblock->outchar;
        myblock -> bx = bx;
        sys_int(VIDEO_IO,myblock);
        myblock->row = (peek40(cursor_posn+(bx >> 8)<<1)) >> 8;
        myblock->col = peek40(cursor_posn+(bx >> 8)<<1);
       }
       else
       {
        myblock -> cx = 1;
        myblock -> bx = bx;
        if (ax & 0x00ff > 2)
        {
         myblock -> bx = peekb(es,bp);
         bp++;
        }
        myblock -> ax = (write_ac_current << 8) | myblock->outchar;
        sys_int(VIDEO_IO,myblock);
        myblock->col++;
        if (myblock->col >> (CRT_COLS))
        {
         myblock->row++;
         if (myblock->row >=25)
         {
          myblock->col = 0;
          myblock -> ax = (write_tty << 8) | line_feed;
          sys_int(VIDEO_IO,myblock);
          myblock->row--;
         }
        }
        /* update cursor */
        myblock -> ax = (set_cursor_position << 8);
        myblock -> dx = (myblock->row << 8) | myblock->col;
        sys_int(VIDEO_IO,myblock);
       }
      }
      /* restore original cursor */
      if ((ax & 1) == 0)
      {
       poke40((cursor_posn + ((bx >>8) << 1)),myblock->temp_cursor);
       myblock -> ax = (set_cursor_position << 8);
       myblock -> dx = myblock->temp_cursor;
       sys_int(VIDEO_IO,myblock);
      }
     }
     break;

    /*============================================*/
    /*===                                      ===*/
    /*=== default case for invalid opcodes     ===*/
    /*===                                      ===*/
    /*============================================*/
   }
  }
  release_block(myblock);
 }


 /*============================================*/
 /*===                                      ===*/
 /*===          SUPPORTING FUNCTIONS        ===*/
 /*===                                      ===*/
 /*============================================*/

 // find position - returns offset page*row*col

 unsigned find_position()
 {
  return(find_page_position(ACTIVE_PAGE));
 }

 unsigned find_page_position(unsigned char page)
 {
  unsigned temp,length;
  unsigned char row,col;
  temp = peek40(cursor_posn + (page << 1));
  row = temp >> 8;
  col = temp;
 length = CRT_LENGTH;
 temp = length * page;
 return(temp + position(row,col));
}

/*---
**--- calc buffer address of character at AX = row,col
**--- returns ax offset of character in buffer
*/

unsigned position(unsigned char row,unsigned char col)
{
 return(((row * CRT_COLS)+col)<<1);
}

/*---
**---  scroll end
*/

void scroll_end()
{
 if (CRT_MODE != 7) outportb(0x3d8,CRT_MODE_SET);
}

/*---
**--- convert cursor position to buffer offset
*/

unsigned cursor_to_offset()
{
 return(((peek40(cursor_posn) >> 8) * CRT_COLS * 4) + CURSOR_POSN & 0x0ff);
}

/*---
**--- convert input row-col to buffer offset
*/

/* input is row,col position */
unsigned calc_offset(unsigned input)
{
 return(((input >> 8) * CRT_COLS * 4) + input & 0x0ff);
}

/*---
**--- return alpha or graphics mode flag
*/

unsigned check_mode()
{
 if ((CRT_MODE >= 4) && (CRT_MODE <= 6)) return(is_graphics);
 return(is_alpha);
}

/*---
**---  expand the byte for medium res.
**---  double the input byte to an output integer
*/

unsigned expand_byte(unsigned char the_byte, unsigned char color)
{
 register temp = 0;
 unsigned char mask = 0x80;
 while (mask != 0)
 {
  temp = temp << 2;
  if ((the_byte & mask) != 0)
  if((color & 0x80) == 0)
         temp |= (color & 0x03);
  else
         temp ^= (color & 0x03);
  mask = mask >> 1;
 }
 return(((temp >> 8) & 0x00ff) | ((temp <<8 ) & 0xff00));
}

/*-----------------------------------------------*/
/*---                                         ---*/
/*---  load a word into the 6845 reg 'n' and 'n'+1 --*/
/*---                                         ---*/
/*-----------------------------------------------*/

void load_6845_word (unsigned address,unsigned value)
{
 load_6845_byte(address, value >> 8);
 load_6845_byte(address + 1, value);
}
```

```
/*---------------------------------------------------*/
/*---                                             ---*/
/*---  load a byte into the 6845 reg 'n'          ---*/
/*---                                             ---*/
/*---------------------------------------------------*/

void load_6845_byte(unsigned address,unsigned char value)
{
 outportb(ADDR_6845, address);
 outportb(((ADDR_6845) + 1),value);
}

unsigned swap(unsigned aword)
{
 return((aword >> 8) | (aword << 8));
}
```

EIGHT
The Floppy Disk Driver

The Floppy Disk Driver is used to operate the floppy disk.

This driver structure has been organized to support up to four (4) floppy disk drives. The normal AT style disk controller does not provide the motor and select logic to do so. Two possible options are to:

1. Install a second controller at an alternate address and sense which port address should be referenced for each drive. This would entail declaring a "port" variable which would be used on the inport and outport commands.

2. Install a controller which does support four drive configurations. If such a controller is not commercially available, one may design and construct such a device by duplicating the standard functions and adding the required motor and select logic. Since newer LSI based floppy disk controllers (such as the Intel 82072) simplify this task, one may wish to consider this alternative.

Presuming that the new perpendicular-recording 4 Mbyte floppy disk drives will become desired options, the disk parameter tables are organized for easy expansion. By inspecting the defined tables in the source file, one will observe that some entries are already included for the 4 Mb drives. Take note that these drive parameters (and the 4Mb drives) have not been installed or tested on the Bios development system, so that some additional development work will be required to implement the 4 Mb drive type.

The standard location in the CMOS RAM for storing the drive types for Drive A: and B: is cell 10 hex. We have chosen the "reserved" cell 11 hex for storing the type byte for floppy Drives C: and D:. If these should conflict with some other use, they may be re-assigned as you wish.

For those of you who are creating diskless systems, our Annabooks PromKit publication provides additional information on how the standard floppy disk driver may be replaced / chained / enhanced so a different physical device may be used for a logical floppy disk.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios enhanced floppy disk driver
*
* Version: 1.02
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: atdisk.c
*
* Functional Description:
*
* Arguments:
*
* Return:
*
* Version History:
* 1.01
*   Moved misplaced parm table entry for 720k drive from index 2 to index 3
* 1.02
*  Replaced peeks and pokes with casts
*  Moved variables to myblock
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

void fdisk_setup(void);
void interrupt cdecl far disk_io(void);
void interrupt cdecl far disk_isr(void);

unsigned char send_fdc(unsigned char);
void send_rate(unsigned);
unsigned retry(unsigned);
unsigned char med_change(unsigned);
unsigned char calc_sectors(unsigned);
unsigned char cmos_type(unsigned char);
unsigned char get_parm(unsigned,unsigned char);
unsigned char wait_int(void);
unsigned recal(unsigned);
unsigned char seek(unsigned);
void wait_for_head(unsigned);
unsigned char read_id(unsigned);
unsigned char get_fdc_status(unsigned);
unsigned char read_dskchng(unsigned);
unsigned char results(void);
unsigned char chk_stat_2(void);
void send_specify_command(unsigned);
void FDC_reset(unsigned);
void delay_call(unsigned,unsigned);
unsigned char dma_setup(unsigned,unsigned,unsigned);
void purge_fdc(void);
unsigned char fdc_init(unsigned);

/* G L O B A L   C O N S T A N T S */

extern _disk_io;
extern _disk_isr;

/* G L O B A L   V A R I A B L E S */

/* L O C A L   D E F I N I T I O N S */

#define FDC_STATUS   0x42
#define DISK_ID    0x90 /* hold drive/media type  */
#define LAST_TRACK 0x94 /* 94-97 holds last track number */

#define RATE_500 0x00
#define RATE_300 0x01
#define RATE_250 0x02
#define RATE_1000       0x03
#define INT_FLAG BIT7
#define MOTOR_WAIT 0x25

#define BAD_CMD  0x01
#define BAD_ADDR_MARK 0x02
#define WRITE_PROTECT 0x03
#define RECORD_NOT_FND 0x04
#define MEDIA_CHANGE 0x06
#define BAD_DMA  0x08
#define DMA_BOUNDARY 0x09
#define MED_NOT_FND 0x0c
#define BAD_CRC  0x10
#define BAD_FDC  0x20
#define BAD_SEEK 0x40
#define TIME_OUT 0x80

/* function code definitions */

#define RESET  0x00
#define READ_STATUS 0x01
#define READ_SECTORS 0x02
#define WRITE_SECTORS 0x03
#define VERIFY_SECTORS 0x04
#define FORMAT_TRACK 0x05
#define DISK_PARMS 0x08
#define DISK_TYPE 0x15
#define DISK_CHANGE 0x16
#define FORMAT_SET 0x17
#define SET_MEDIA 0x18
```

```
#define BAD_FUNCTION 0x19

#define TRY  0xff

/* FDC port definitions */

#define DOR_PORT   0x3f2
#define MSR_PORT  0x3f4
#define DATA_PORT  0x3f5
#define DIR_PORT  0x3f7
#define DRR_PORT  0x3f7

#define RQM  BIT7
#define DIO  BIT6
#define BUSY  BIT4
#define DSKCHANGE_BIT BIT7

/* DMA port definitions */

#define DMA_MASK 0x0a
#define DMA_MODE 0x0b
#define DMA_FLFF 0x0c
#define DMA_ADR  0x04
#define DMA_BASE 0x05
#define DMA_PAGE 0x81

/* DMA literal definitions */

#define DMA_ON  0x02
#define DMA_OFF  0x06
#define DMA_RX_MODE 0x46
#define DMA_TX_MODE 0x4a
#define DMA_VRFY_MODE 0x42
#define TX_DIR  0x01
#define RX_DIR  0x00

/* FDC commands */

#define FDC_READ 0xe6
#define FDC_WRITE 0xc5
#define FDC_FORMAT 0x4d
#define FDC_READID 0x4a

/*=== These are the floppy disk parameter tables =====
*
* The tables are organized as an Array of up to 8 Drive types, each supporting up to 8 media types.
* Each item is 16 bytes long
*
*/
/* DISK TABLE indexes */

#define DT_SPEC1 0  // specify command 1
#define DT_SPEC2 1  // specify command 2
#define DT_OFF_TIM 2          // motor off count
#define DT_BYT_SEC 3          // bytes/sector
#define DT_SEC_TRK 4          // sectors/track
#define DT_GAP 5    // gap
#define DT_DTL 6    // dtl
#define DT_GAP3 7   // gap 3 for format command
#define DT_FIL_BYT 8          // fill byte for format command
#define DT_HD_TIM 9 // head settle time
#define DT_STR_TIM 10         // motor start time
#define DT_MAX_TRK 11         // max # of tracks for drive
#define DT_RATE 12  // data rate
#define DT_TYPE               // drive and media type (not used)
#define DT_STEP 14  // double step flag

#define drive_established 0x08
#define drive_field 0x07
#define drive_none  00
#define drive_360  01
#define drive_12  02
#define drive_720  03
#define drive_14  04
#define drive_28  05

#define media_established 0x80
#define media_field 0x70
#define media_none  0x00
#define media_360  0x10
#define media_12  0x20
#define media_720  0x30
#define media_14  0x40
#define media_28  0x50
// media definitions for transition table
```

```
#define m_none  0x0
#define m_360  0x1
#define m_12  0x2
#define m_720  0x3
#define m_14  0x4
#define m_28  0x5

#define m_wait 0x25

/* L O C A L   C O N S T A N T S */

const unsigned char transition_table[8][4]= {
// This table covers the sequence of media type to try for
// each drive type in establishing the media in the drive.
// The media is defaulted to the first item in this table
// for a particular drive type. When we attempt retries, we
// step through the possible media types until we come
// to "none" marking the end of the table.

/* drive type 0 (None) */ {m_none,m_none,m_none,m_none},
/* drive type 1 ( 360) */ {m_360, m_none,m_none,m_none},
/* drive type 2 ( 1.2) */ {m_12,  m_360, m_none,m_none},
/* drive type 3 ( 720) */ {m_720, m_none,m_none,m_none},
/* drive type 4 (1.44) */ {m_14,  m_720, m_none,m_none},
/* drive type 5 (2.88) */ {m_28,  m_14,  m_720, m_none},
/* drive type 6 unused */ {m_none,m_none,m_none,m_none},
/* drive type 7 unused */ {m_none,m_none,m_none,m_none},
};

const unsigned char fdisk_table[8][8][16]= {
/* this entry is for compatibility with the XT disk driver */
{
 /* ---- drive type 0 = None (Default to 360) ----*/

 {0x0af,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,39,RATE_250,drive_360|media_360,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{
 /* ---- drive type 1 = 360 ----*/

 /* 0 = no media in 360 kb drive */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 1 = 360 kb media in 360 kb drive */
 {0x0af,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,39,RATE_250,drive_360|media_360,0,0},

 /* 2-7 not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},


{
 /* ---- drive type 2 = 1.2 ------*/

 /* 0 = no media in 1.2 mb drive */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 1 = 360 kb media in 1.2 mb drive */
 {0xaf,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,39,RATE_300 ,drive_12|media_360,1,0},

 /* 2 = 1.2 mb media in 1.2 mb drive */
 {0xaf,2,m_wait,2,15,0x1b,-1,0x54,0x0f6,15,8,79,RATE_500 ,drive_12|media_12,0,0},

 /* 3-7 = not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},


{
 /* ---- drive type 3 = 720 ------ */
 /* 0 = no media in 720 kb drive */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 1-2 = not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 3 = 720 kb media in 720 kb drive */
 {0x0af,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,79,RATE_250 ,drive_720|media_720,0,0},

 /* 4-7 = not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{
 /* ---- drive type 4 = 1.44 ------*/

 /* 0 = no media in 1.44 mb drive */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 1-2 =  not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 3 =  720 kb media in 1.44 mb drive */
 {0xaf,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,79,RATE_250 ,drive_14|media_720,0,0},

 /* 4 = 1.44 kb media in 1.44 mb drive */
 {0xaf,2,m_wait,2,18,0x1b,-1,0x6c,0x0f6,15,8,79,RATE_500 ,drive_14|media_14,0,0},

 /* 5-7 = not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{

 /*---- drive type 5 = 2.88 ------*/

 // The 2.88 Meg drives have not been tested.
 // These tables are included as a help to integrating 2.88 drives into your system.
 // The correct parameters must be determined in conjunction with the drive manufacturers
 // specifications.

 /* 0 = no media in 2.88 mb drive */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 1-2 not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 3 = 720 kb media in 2.88 mb drive */
 {0xaf,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,79,RATE_250 ,drive_28|media_720,0,0},

 /* 4 = 1.44 kb media in 2.88 mb drive */
 {0xaf,2,m_wait,2,18,0x1b,-1,0x6c,0x0f6,15,8,79,RATE_500 ,drive_28|media_14,0,0},

 /* 5 = 2.88 kb media in 2.88 mb drive */
 {0xaf,2,m_wait,2,36,0x1b,-1,0x53,0x0f6,15,8,79,RATE_1000,drive_28|media_28,0,0},

 /* 6-7 not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{
 /*---- drive type 6 = reserved ------*/

 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{
 /*---- drive type 7 = reserved ------*/
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

};

/* P R O G R A M */

/**********************************************************
*
*  FOSCO Enhanced Floppy Disk Driver
*
*  Copyright FOSCO 1988, 1989
*
**********************************************************/
/*

This driver uses a drive/media id byte as follows:

bit 7 0/1 = media not established / established

bit 6-4 media type
000 = no established media
001 = 360 (40/80 track)
010 = 1.2 (80 track)
011 = 720 (80 track)
100 = 1.4 (80 track)
101 = 2.8 (80 track)
110 - 111 reserved

bit 3 = 0/1 = drive not established / established

bit 2-0 Drive type
000 no established drive type
001 = 360 (40 track)
010 = 1.2 (80 track)
011 = 720 (80 track)
100 = 1.4 (80 track)
101 = 2.8 (80 track)
110 - 111 reserved

The drive/media bytes are located at 40:90-93 for up to four drives.*/

/*====================================================*/
/*=================== START OF CODE ===================*/
/*====================================================*/

variables
unsigned char fd_drive_type;
unsigned char fd_state;
unsigned char fd_fdc_command;
unsigned char fd_dma_command;
unsigned char fd_sector_track;
unsigned char fd_media_type;
unsigned char fd_fcode;
unsigned char fd_drive;
unsigned char fd_nr_sectors;
unsigned char fd_head;
unsigned char fd_sector;
unsigned char fd_track;
unsigned fd_max_track;
unsigned fd_max_sectors;
unsigned fd_i;
unsigned fd_try_count;
unsigned fd_media_index;
unsigned char end_track,end_head,end_sector,num_sectors,sectors_track;
unsigned dma_address;
unsigned long lcheck;
unsigned nibble, upper_byte, lower_byte, length;
unsigned char error_byte,time_out_flag;
end_variables fdisk_regs;

/*-------------- setup the controller -----------*/

void fdisk_setup ()
{
 fdisk_regs *myblock;
 myblock = acquire_block(FLOPPY);

 link_interrupt(0x13,&_disk_io);
 link_interrupt(0x0e,&_disk_isr);
 set_vector(0x1e,bios_cs(),&fdisk_table[0][0][0]);

 outportb(0x21,inportb(0x21) & ~BIT6);     // unmask int's
 SEEK_STATUS = 0;   /* clear seek status */
 MOTOR_COUNT = 0;   /* clear motor count */
 MOTOR_STATUS = 0;  /* clear motor status */
 DISK_STATUS = 0;   /* clear disk status */

 HF_CNTRL |= 1;
 RTC_WAIT_FLAG |= 1;
 pokeb40(DISK_ID+0,0);    /* clear drive 0 state */
 pokeb40(DISK_ID+1,0);    /* clear drive 1 state */
 pokeb40(DISK_ID+2,0);    /* clear drive 2 state */
 pokeb40(DISK_ID+3,0);    /* clear drive 3 state */

 pokeb40(DISK_ID+0,cmos_type(0));
 pokeb40(DISK_ID+1,cmos_type(1));
 pokeb40(DISK_ID+2,cmos_type(2));
 pokeb40(DISK_ID+3,cmos_type(3));

 RTC_WAIT_FLAG &=  0xfe;      /* allow for rtc wait */
 MOTOR_COUNT = get_parm(myblock,2);     /* set motor count */

 myblock -> dx = 0x0000;
 myblock -> ax = 0x0000;  /* reset the FDC */
 sys_int(0x13,myblock);
 release_block(myblock);
}

/*=========================================================*/
/*=========================================================*/
/*=========================================================*/
/*======== BEGINNING OF MAIN ROUTINES  ==================*/
/*=========================================================*/
/*=========================================================*/
/*=========================================================*/

/* handle the floppy disk function service call */

void interrupt cdecl far disk_io(interrupt_registers)
{
 fdisk_regs *myblock;
// disable();                // some XT type disk controllers will pre-enable
 myblock = acquire_block(FLOPPY);
// enable();

 snapshot_in(FLOPPY,&es);

 myblock->fd_fcode = ax >> 8;
 myblock->fd_drive = dx & 0x00ff;
 myblock->fd_nr_sectors = ax & 0x00ff;
 myblock->fd_head = dx >> 8;
 myblock->fd_sector = cx & 0x00ff;
 myblock->fd_track = cx >> 8;

 flags &= ~0x0001;  /* clear the carry flag for returns */

 DISK_STATUS = 0;  /* clear status */

 if (((ax >> 8) != 0) && ((dx & 0x00ff) > 3))
 {  // bad function code because of drive number
  DISK_STATUS = BAD_CMD; ax = (BAD_CMD << 8) | (ax & 0xff);
  flags |= 1;  /* set return error flag */
 }
 else
 {
  /* make sure any residual status is unloaded */
  purge_fdc();
  /* set the data rate register to a default value */
  outportb(DRR_PORT,RATE_250);

  switch (myblock->fd_fcode)
  {
   case RESET :  /* reset the disk controller */
   watch_string(FLOPPY,"Reset");
   FDC_reset(myblock);
   ax = (DISK_STATUS << 8) || (ax & 0xff);
   if ((ax & 0xff00) != 0) flags |= 1;
   break;

   case READ_STATUS:
   watch_string(FLOPPY,"Read Status");
   ax = DISK_STATUS << 8;
   if ((ax & 0xff00) != 0) flags |= 1;
   break;
```

```
    case READ_SECTORS: /* read sectors */
    watch_string(FLOPPY,"Read sectors");
    MOTOR_STATUS &= ~INT_FLAG;
    myblock->fd_fdc_command = FDC_READ;
    myblock->fd_dma_command = DMA_RX_MODE;
    goto read_write_verify;

    case WRITE_SECTORS:
    watch_string(FLOPPY,"Write sectors");
    MOTOR_STATUS |= 0x80;
    myblock->fd_fdc_command = FDC_WRITE;
    myblock->fd_dma_command = DMA_TX_MODE;
    goto read_write_verify;

    case VERIFY_SECTORS:
    watch_string(FLOPPY,"Verify sectors");
    MOTOR_STATUS &= ~INT_FLAG;
    myblock->fd_fdc_command = FDC_READ;
    myblock->fd_dma_command = DMA_VRFY_MODE;
    goto read_write_verify;

read_write_verify:

     // if a media change sensed, then de-establish media
    if (med_change(myblock) == error)
    {
     andb40(DISK_ID+myblock->fd_drive,0x0f);
     myblock->fd_media_index = 0;
    }
    myblock->fd_try_count = 3;
    do  // this is the major loop, try the major operation 3 times
    {
     watch_string(FLOPPY,"\n\rMajor Loop");

     // if media not established, then set media type = drive type
     // set media_index = 0, set media type by index

     if ((peekb40(DISK_ID+myblock->fd_drive) & media_established) == 0)
     {
      // set the media index for the first possible type of media
      myblock->fd_drive_type = peekb40(DISK_ID+myblock->fd_drive) & 0x07;

      // set the DISK_ID with the first media type to try
      pokeb40(DISK_ID+myblock->fd_drive,(peekb40(DISK_ID+myblock->fd_drive) &
           ~media_field) |
                   (peekbcs(&transition_table[myblock->fd_drive_type][0]) << 4));
      // set the media type according to the drive type [index = 0]
      myblock->fd_media_type = (peekb40(DISK_ID+myblock->fd_drive) >> 4) & 0x07;
     }
     do  // this is the minor loop
     // do until we run out of retrys
     {

      watch_string(FLOPPY," Minor Loop ");
      purge_fdc();     /* make sure any residual status is unloaded */

       // set up drive and media indexes before doing the dma setup,
       // which needs to know how many bytes per sector for calcing xfr length
       myblock->fd_drive_type = peekb40(DISK_ID+myblock->fd_drive) & 0x07;
       myblock->fd_media_type = (peekb40(DISK_ID+myblock->fd_drive) >> 4) & 0x07;
       watch_string(FLOPPY," Disk ID byte = ");
       watch_byte(FLOPPY,peekb40(DISK_ID+myblock->fd_drive));

      if (dma_setup(myblock,es,bx) == ok)
      {
       /* send the specify command */
       send_fdc(3);
       send_fdc(get_parm(myblock,0));
       send_fdc(get_parm(myblock,1));

       send_rate(myblock);

       if (fdc_init(myblock) == ok)
       {
        if (send_fdc(myblock->fd_track) == ok)
        {
         if (send_fdc(myblock->fd_head) == ok)
         {
          if (send_fdc(myblock->fd_sector) == ok)
          {
           if (send_fdc(get_parm(myblock,3)) == ok)
           {
            if (send_fdc(get_parm(myblock,4)) == ok)
            {
             if (send_fdc(get_parm(myblock,5)) == ok)
         {
          if (send_fdc(get_parm(myblock,6)) == ok)
          {
           if(get_fdc_status(myblock) == ok)
           {
            orb40(DISK_ID+myblock->fd_drive,media_established); break;  // get out with good
operation
           }
          }
         }
        }
       }
      }
     }
    }
   }
  }
 }
  while (retry(myblock) == error);
  watch_string(FLOPPY,"\n\rDISK STATUS = ");watch_byte(FLOPPY,DISK_STATUS);
 }
 while ((--myblock->fd_try_count > 0) &&
        ((peekb40(DISK_ID+myblock->fd_drive) & media_established) == 0) &&
                    ((DISK_STATUS & TIME_OUT) != TIME_OUT));
 ax = (DISK_STATUS << 8) | (calc_sectors(myblock));
 if ((ax & 0xff00) != 0) flags |= 1;
 break;

 case FORMAT_TRACK : /* format track */
 watch_string(FLOPPY,"Format track");
 myblock->fd_fdc_command = FDC_FORMAT;
 myblock->fd_dma_command = DMA_TX_MODE;
 // if media not established, then set media type = drive type
 if ((peekb40(DISK_ID+myblock->fd_drive) & media_established) == 0)
        pokeb40(DISK_ID+myblock->fd_drive,
                  (peekb40(DISK_ID+myblock->fd_drive) & 0x07) |
                         ((peekb40(DISK_ID+myblock->fd_drive) & 0x07) << 4));

 MOTOR_STATUS |= 0x80;      /* indicate write operation */
 if (med_change(myblock) == ok)     /* check for media change */
 {
  watch_string(FLOPPY,"\n\rNo media change");
  send_fdc(3);  /* send specify command */
  send_fdc(get_parm(myblock,0));
  send_fdc(get_parm(myblock,1));
  watch_string(FLOPPY,"\n\rSpecify Command Sent");
  send_rate(myblock);
  if (dma_setup(myblock,es,bx) == ok)
  {
   watch_string(FLOPPY,"\n\rDma Setup ok");
   if (fdc_init(myblock) == ok)
   {
    watch_string(FLOPPY,"\n\rFDC Init ok");
    if (send_fdc(get_parm(myblock,3)) == ok)
    {
     watch_string(FLOPPY,"\n\rSent parm 3");
     if (send_fdc(get_parm(myblock,4)) == ok)
     {
      watch_string(FLOPPY,"\n\rSent parm 4");
      if (send_fdc(get_parm(myblock,7)) == ok)
      {
       watch_string(FLOPPY,"\n\rSent parm 7");

       send_fdc(get_parm(myblock,8)); /* send command */
       watch_string(FLOPPY,"\n\rFDC Commands all sent");

       get_fdc_status(myblock);
       watch_string(FLOPPY,"\n\rFDC status rx'd");

      }
     }
    }
   }
  }
 }
 ax = DISK_STATUS << 8;
 if ((ax & 0xff00) != 0) flags |= 1;
 break;

 case DISK_PARMS : /* read drive parameters */
 watch_string(FLOPPY,"Read disk parms");
 ax = bx = cx = dx = es = 0;    /* reset all registers */
 if (myblock->fd_drive > 0x80)
 {
  ax = (ax & 0xff) | (BAD_CMD << 8); flags |= 1;  /* set return error flag */
 }
```

```
else
{
 if ((EQUIP_FLAG & 0x01) != 0) /* there are drives present */
 {
  di = dx; // save old dx temporarily
  dx = EQUIP_FLAG >> 6;
  /* return # of drives in dx */
  if (di > dx) /* called for a non-existent drive */
  di = 0; /* return with null parameters */
  else
  {
   di = 0;
   if (cmos_type(myblock->fd_drive) != 0) /* get parms for this type */
   {
    myblock->fd_drive_type = myblock->fd_media_type = cmos_type(myblock->fd_drive) & 0x07;

    di = &fdisk_table[myblock->fd_drive_type] [myblock->fd_media_type] [0];
    myblock->fd_sector_track = peekbcs(&fdisk_table[myblock->fd_drive_type] [myblock-
>fd_media_type] [DT_SEC_TRK]);
    myblock->fd_max_track    = peekbcs(&fdisk_table[myblock->fd_drive_type] [myblock-
>fd_media_type] [DT_MAX_TRK]);
    cx = ((myblock->fd_max_track << 8) & 0xff00) | (myblock->fd_sector_track & 0x3f);
    dx = (1 << 8) | dx;
    bx = cmos_type(myblock->fd_drive) & 0x07;
    es = bios_cs();       // needs to be our code segment
   }
  }
 }
}
break;

case DISK_TYPE :  /* read disk type */
watch_string(FLOPPY,"Read disk type");

ax &= 0x00ff;  /* no drive */

if ((peekb40(DISK_ID+myblock->fd_drive) & drive_field) != drive_none)
{
 if ((peekb40(DISK_ID+myblock->fd_drive) & drive_field) == drive_360)
 {
  ax |= 0x0100; /* 40 track, no change line */
 }
 else
 {
  ax |= 0x0200; /* 80 track, change line */
 }
}
break;

case DISK_CHANGE :     /* return change line condition */
watch_string(FLOPPY,"Read disk change line");
/* if no drive - then return a timeout condition */
if ((peekb40(DISK_ID+myblock->fd_drive) & drive_field) == drive_none)
{
 DISK_STATUS |= TIME_OUT;
}
else
{
 /* if 360 k drive - always return disk change */
 if ((peekb40(DISK_ID+myblock->fd_drive) & drive_field) == drive_360)
 DISK_STATUS = MEDIA_CHANGE;
 else
 /* if 720, 1.4, or 2.8 drive - read the change line */
 if (read_dskchng(myblock) != 0 )
 DISK_STATUS = MEDIA_CHANGE;
}
ax = (DISK_STATUS << 8) | (ax & 0xff);
if ((ax & 0xff00) != 0) flags |= 1;
break;

case FORMAT_SET : /* set disk type */
watch_string(FLOPPY,"Set disk type");

/* set drive to requested  format/media */
myblock->fd_drive_type = peekb40(DISK_ID+myblock->fd_drive) & 0x07;

switch (ax & 0x00ff) /* use requested type for switch */
{
 case 1: /* 360/360 */
 if(myblock->fd_drive_type == drive_360)
 pokeb40(DISK_ID+myblock->fd_drive,(media_360 | drive_360 | media_established));
 break;

 case 2: /* 360/1.2 */
 if(myblock->fd_drive_type == drive_12)
 pokeb40(DISK_ID+myblock->fd_drive,(media_360 | drive_12 | media_established));
```

```
     break;

     case 3: /* 1.2/1.2 */
     if(myblock->fd_drive_type == drive_12)
     pokeb40(DISK_ID+myblock->fd_drive,(media_12 | drive_12 | media_established));
     break;

     case 4: /* 720/720 */
     if(myblock->fd_drive_type == drive_720)
     pokeb40(DISK_ID+myblock->fd_drive,(media_720 | drive_720 | media_established));
     break;

     default:
     DISK_STATUS = BAD_CMD;  /* bad command */
     break;
    }
    ax = (DISK_STATUS << 8) | (ax & 0xff);
    if ((ax & 0xff00) != 0) flags |= 1;
    break;

    case SET_MEDIA :       /* set media type */
    watch_string(FLOPPY,"Set media type");
    // get the max tracks called for
    myblock->fd_max_track = (cx >> 8) | ((cx << 2) & 0x300);
    // get the max sectors called for
    myblock->fd_max_sectors = cx & 0x003f;
    if (cmos_type(myblock->fd_drive) != 0)
    {
     myblock->fd_drive_type = cmos_type(myblock->fd_drive) & 0x07;
     watch_string(FLOPPY,"\n\rDrive is ");watch_byte(FLOPPY,myblock->fd_drive);
     watch_string(FLOPPY,"\n\rDrive type is ");watch_byte(FLOPPY,myblock->fd_drive_type);
     //    myblock->fd_drive_type = peekb40(DISK_ID+myblock->fd_drive)& 0x07;
     // use the transition table for this drive
     for (myblock->fd_i= 0;peekbcs(&transition_table[myblock->fd_drive_type][myblock->fd_i]) !=
0;myblock->fd_i++)
     {
      myblock->fd_media_type = peekbcs(&transition_table[myblock->fd_drive_type][myblock->fd_i]);
      watch_string(FLOPPY,"\n\rMedia type is ");watch_byte(FLOPPY,myblock->fd_media_type);


      if ((myblock->fd_max_sectors == peekbcs(&fdisk_table[myblock->fd_drive_type][myblock-
>fd_media_type][DT_SEC_TRK])) && \
                         (myblock->fd_max_track == peekbcs(&fdisk_table[myblock->fd_drive_type][myblock-
>fd_media_type][DT_MAX_TRK])))
      {
       // return the disk parms ptr to the caller
       di = &fdisk_table[myblock->fd_drive_type][myblock->fd_media_type][0];
       es = bios_cs();
       // set the disk_id to the established supported media
       pokeb40(DISK_ID+myblock->fd_drive,myblock->fd_drive_type | (myblock->fd_media_type << 4) |
       drive_established | media_established);
       watch_string(FLOPPY,"\n\rMedia established, drive byte is: ");
       watch_byte(FLOPPY,peekb40(DISK_ID+myblock->fd_drive));
       // need a break to say OK here !!!
       goto set_media_exit;
      }
     }
     if (peekbcs(&transition_table[myblock->fd_drive_type][myblock->fd_i]) == 0)
     DISK_STATUS = MED_NOT_FND;
    }
    set_media_exit:
    ax = (DISK_STATUS << 8) | (ax & 0xff);
    if ((ax & 0xff00) != 0) flags |= 1;
    break;

    default : /* invalid function code - return bad code */
    watch_string(FLOPPY,"Bad Command");
    DISK_STATUS = BAD_CMD;
    ax = (BAD_CMD << 8) | (ax & 0xff);
    flags |= 1;  /* set return error flag */

   } /* end of switch */

  }

  snapshot_out(FLOPPY,&es);

  MOTOR_COUNT = get_parm(myblock,2); /* set motor count */
  release_block(myblock);

} /* end of disk_io */

/*=====================================================*/

/*=====================================================*/
```

```
/* reset the disk system */

void FDC_reset(fdisk_regs *myblock)
{
 unsigned i;
 unsigned char status;

 disable();
 status = (MOTOR_STATUS << 4) | 8;
 // if a motor is on, set the drive select bits accordingly
 switch (status & 0xf0)
 {
  /* motor 2 on */
  case 0x20: status |= 0x01; break;

  /* motor 3 on */
  case 0x40: status |= 0x02; break;

  /* motor 4 on */
  case 0x80: status |= 0x03; break;
 }
 outportb (DOR_PORT,status);  /* reset disk controller */
 SEEK_STATUS = 0;
 outportb (DOR_PORT,status | 0x04); // reset the reset bit
 enable();    /* enable interrupts */
 /* wait for interrupt */
 if (wait_int() == error) { DISK_STATUS |= BAD_FDC; return; }
 for (i = 0; i < 4; i++)   /* number of drives */
 {
  /* send command */
  if (send_fdc(8) == error) { DISK_STATUS |= BAD_FDC; return; }
  /* check results */
  if (results() != ok) { DISK_STATUS |= BAD_FDC; return; }
  if (peekb40(FDC_STATUS) != (i | 0xc0)) { DISK_STATUS |= BAD_FDC; return; }
 }
 send_specify_command(myblock);
}

/*========================================================

When the hardware interrupt occurs from the floppy disk,
this function sets bit 7 in the seek_status flag and sends
an end-of-interrupt command to the 8259.

--------------------------------------------------------*/

void interrupt cdecl far disk_isr (void)
{
 SEEK_STATUS |= INT_FLAG; outportb (0x20,0x20);   /* send eoi */
}

/*========================================================

Return the drive type for a specified drive.

--------------------------------------------------------*/


unsigned char cmos_type (unsigned char drive_nr)
{
 unsigned char drive_type = 0;
 switch(drive_nr)
 {
  case 0: drive_type = incmos(0x10) >> 4; break;
  case 1: drive_type = incmos(0x10) & 0x0f; break;
  case 2: drive_type = incmos(0x11) >> 4; break;
  case 3: drive_type = incmos(0x11) & 0x0f; break;
 }
 if (drive_type != 0) drive_type |= drive_established;
 return(drive_type);
}

/*========================================================

Send a byte to the FDC.

--------------------------------------------------------*/

unsigned char send_fdc(unsigned char parm)
{
 unsigned long i = set_timeout_count(5);         // wait at least 1 sec

 do
 {
  if ((inportb(MSR_PORT) & 0xc0) == RQM)
  {
   outportb(DATA_PORT,parm);
   delay_call(0,50); /* delay at least 50 useconds */
   return (ok);
  }
 }
 while (TIMER_LONG < i);
 DISK_STATUS |= TIME_OUT;
 return (error);
}

/*=========================================================

Get the indexed value from the disk parameter table.

---------------------------------------------------------*/

unsigned char get_parm(fdisk_regs *myblock,unsigned char index)
{
 return(peekbcs(&fdisk_table[myblock->fd_drive_type][myblock->fd_media_type][index]));
}

/*=========================================================

Send the specify command to the FDC.

---------------------------------------------------------*/

void send_specify_command (fdisk_regs *myblock)
{
 send_fdc(3);
 send_fdc(get_parm(myblock,0));
 send_fdc(get_parm(myblock,1));  /* send command */
}
/*=======================================================*/
/*=======================================================*/
/*=======================================================*/

/*=====================================================

Turn motor on and wait for motor start up time if this is a
write operation. Proceed immediately if a read operation.
1. save the last state of the motor.
2. turn on the selected motor.
3. if not a write, exit immediately.
3. if a write, if the same motor was already on, exit, else wait for the delay.

---------------------------------------------------------*/

void motor_on(fdisk_regs *myblock)
{
 unsigned char this_motor, last_motor, wait;

 disable();
 MOTOR_COUNT = 0xff;
 /* hit timer with max on-count */
 last_motor = MOTOR_STATUS & 0x0f;
 wait = MOTOR_STATUS & 0x80;
 MOTOR_STATUS = this_motor = (1 << myblock->fd_drive);
 outportb(DOR_PORT,(this_motor << 4) | 0x0c | myblock->fd_drive);
 enable();
 if((last_motor != this_motor) && (wait != 0))
 {
  // delay for a write and a motor start
  // force a delay in 1/8 seconds
  // delay = (resolution,count) in milliseconds */

  delay_call(125,get_parm(myblock,10));
  MOTOR_COUNT = 0xff;
  /* hit timer with max on-count */
  watch_string(FLOPPY,"\n\rMotor on Delay = ");
  watch_byte(FLOPPY,get_parm(myblock,10));
 }
}

/*=======================================================

Wait for the hardware interrupt to occur. Time-out and return
if no interrupt.

---------------------------------------------------------*/

unsigned char wait_int (void)
{
 /* this time out should be 3 secs */
 unsigned long i = set_timeout_count(3 * 20);
 disable();
 outportb(0x21,inportb(0x21) & ~BIT6);
 enable();
 do
 {
  if((SEEK_STATUS & INT_FLAG) != 0) { SEEK_STATUS &= ~INT_FLAG; return (ok); }
 }
 while (TIMER_LONG < i);
 DISK_STATUS |= TIME_OUT; SEEK_STATUS &= ~INT_FLAG;
 return (error);
}

/*=========================================================

Send the data-transfer-rate command to controller

---------------------------------------------------------*/

void send_rate(fdisk_regs *myblock)
{
 outportb(DRR_PORT,peekbcs(&fdisk_table[peekb40(DISK_ID+myblock->fd_drive) & 0x07]
 [(peekb40(DISK_ID+myblock->fd_drive) >> 4) & 0x07][DT_RATE]));
}

/*=========================================================

Process the interrupt received after recalibrate or seek.

---------------------------------------------------------*/

unsigned char chk_stat_2 (void)
{
 if (wait_int() == error) return (error);
 if (send_fdc(8) == error) return (error);
 if (results() == error) return (error);
 if ((peekb40(FDC_STATUS) & 0x60) == 0x60)  /* normal termination */
 {
  watch_string(FLOPPY,"\n\rSeek Error, fdc status was - ");
  watch_byte(FLOPPY,peekb40(FDC_STATUS));
  DISK_STATUS |= BAD_SEEK;
  return (error);
 }
 return (ok);
}

/*=========================================================

Initialize dma controller for read, write, verify operations.

---------------------------------------------------------*/

unsigned char dma_setup(fdisk_regs *myblock,unsigned es,unsigned bx)
{
 disable();    /* disable interrupts */
 outportb(DMA_FLFF,0x00); /* set the first/last ff */
 outportb(DMA_MASK,DMA_OFF);  /* disable channel while loading */

 outportb(DMA_MODE,myblock->fd_dma_command); /* output mode byte */
 if (myblock->fd_dma_command == DMA_VRFY_MODE) /* dma verify command ? */
 {     /* yes */
  myblock->dma_address = 0;
  myblock->lower_byte = 0;
  myblock->upper_byte = 0;
  myblock->nibble = 0;
 }
 else
 {
  myblock->dma_address = es + (bx >> 4);
  myblock->nibble = myblock->dma_address >> 12;
  myblock->upper_byte = myblock->dma_address >> 4;
  myblock->lower_byte = (myblock->dma_address << 4) | (bx & 0x000f);
 }

 myblock->length = ((myblock->fd_nr_sectors * 128) << get_parm(myblock,3)) - 1;
 watch_string(FLOPPY,"\n\rTransfer length = ");
 watch_word(FLOPPY,myblock->length);
 outportb(DMA_ADR,myblock->lower_byte);  /* output low address */
 outportb(DMA_ADR,myblock->upper_byte);  /* output high address */
 outportb(DMA_PAGE,myblock->nibble);  /* output page address */
 outportb(DMA_BASE,myblock->length & 0xff); // output low byte length
 outportb(DMA_BASE,myblock->length >> 8);  // output high byte length
 enable();    /* enable interrupts */

 outportb(DMA_MASK,DMA_ON);  /* init disk channel */
 myblock->lcheck = 0;
 myblock->lcheck = ((myblock->upper_byte & 0xff) << 8) | (myblock->lower_byte & 0xff);
 myblock->lcheck = myblock->lcheck + (unsigned long) myblock->length;
 if (myblock->lcheck & 0xffff0000)  /* test for overflow */
 {     /* overflow */
  DISK_STATUS = DMA_BOUNDARY;
  watch_string(FLOPPY,"\n\rDMA overflow");
  return (error);
 }
 return (ok);
}

/*==========================================================

Move the head to the selected track.

---------------------------------------------------------*/

unsigned char seek(fdisk_regs *myblock)
{
//unsigned char drive_type,media_type;
 if ((SEEK_STATUS & (1 << myblock->fd_drive)) == 0) /* need recal */
 {
  SEEK_STATUS |= (1 << myblock->fd_drive); /* mark recal will be done */

  /* try 2 attemps at recalibrate, then if failed, return error */
  if (recal(myblock) == error)
  {
   DISK_STATUS = 0;
   if (recal(myblock) == error) return (error);
  }
  pokeb40(LAST_TRACK+(myblock->fd_drive),0); /* clear track number */
  /* if we want track zero, then just wait for head and exit */
  if (myblock->fd_track == 0) { wait_for_head(myblock); return (ok); }
 }
 if (peekbcs(&fdisk_table[myblock->fd_drive_type][myblock->fd_media_type][DT_STEP]) != 0)
         myblock->fd_track *= 2;
 if (peekb40(LAST_TRACK+myblock->fd_drive) == myblock->fd_track)
          return(ok);
  /* update new position */
 pokeb40(LAST_TRACK+myblock->fd_drive,myblock->fd_track);
 if (send_fdc(0x0f) == error) return (error);
 if (send_fdc(myblock->fd_drive) == error) return (error);
 if (send_fdc(myblock->fd_track) == error) return (error);
 if (chk_stat_2() == error) return (error);
 wait_for_head(myblock);
 return (ok);
}

/*=========================================================

Recalibrate the drive.

---------------------------------------------------------*/

unsigned recal(fdisk_regs *myblock)
{
 if (send_fdc(7) == error) return (error);
 if (send_fdc(myblock->fd_drive) == error) return (error);
 /* send command */
 if (chk_stat_2() == error) return (error);
 /* return with error */
 return (ok);       /* return with no error */
}

/*==========================================================

Determine whether a retry is necessary.
Returning an OK says don't retry any more
Returning an ERROR says retry again

---------------------------------------------------------*/

unsigned retry(fdisk_regs *myblock)
{
                  unsigned char media_type;
 /* if operation timed out - say no retry */
 if ((DISK_STATUS & TIME_OUT ) == TIME_OUT) return (ok);
 /* if media is established - say no retry */
 if ((peekb40(DISK_ID+myblock->fd_drive) & media_established) != 0) return (ok);

 /* we have to step through the media */

 // get the next possible media type for this drive type
 media_type = peekbcs(&transition_table[myblock->fd_drive_type][++myblock->fd_media_index]);
 if (media_type == media_none)  // then at end of possibles
 {
  // dis-establish media
  // return - no more
  andb40(DISK_ID+myblock->fd_drive,~(media_established | media_field));
  return(ok);
 }

 // insert the new media type into the DISK_ID and return saying try again
 pokeb40(DISK_ID+myblock->fd_drive,(peekb40(DISK_ID+myblock->fd_drive) & ~media_field) |
 (media_type << 4));

 DISK_STATUS = 0;
 pokeb40(FDC_STATUS,0);
 return(error); /* return saying retry */
}

/*========================================================

Read anything from the controller following an interrupt.
This may include up to seven bytes of status.

--------------------------------------------------------*/

unsigned char results (void)
{
 unsigned long i;
 unsigned char count,flag;

 for (count = 0; count < 7; count++)    // get up to seven bytes
 {
  flag = error;
  i = set_timeout_count(2); // set delay to 50-100 ms.

  do
  {
   if ((inportb(MSR_PORT) & 0xc0) == 0xc0) /* data available from FDC */
   {
    pokeb40(FDC_STATUS+count,inportb(DATA_PORT)); /* save status */
    delay_call(0,100); /* at least 100 us for the FDC */
    flag = ok;
   }
   /* check busy bit */
   if ((inportb(MSR_PORT) & BUSY) == 0) return (ok);
  }
  while (TIMER_LONG < i);
  if (flag == error) { DISK_STATUS |= TIME_OUT; return (error); }
 }
 DISK_STATUS |= BAD_FDC; /* too many bytes */
 return (error);
}

/*========================================================

Purge the FDC of any status it is waiting to send.

--------------------------------------------------------*/

void purge_fdc (void)
{
 while ((inportb(MSR_PORT) & 0xc0) == 0xc0)
 {
  inportb(DATA_PORT);  /* read status */
  delay_call(0,50); /* at least 50 us for the FDC */
 }
}

/*========================================================

Read the state of the disk change line.

--------------------------------------------------------*/

unsigned char read_dskchng(fdisk_regs *myblock)
{
 motor_on(myblock);   /* turn motor on */
 return (inportb(DIR_PORT) & DSKCHANGE_BIT);
}

/*========================================================

Execute the FDC read id command.

--------------------------------------------------------*/

unsigned char read_id (fdisk_regs *myblock)
```

```
{
 if (send_fdc(FDC_READID) == error) return(error);
 if (send_fdc(myblock->fd_head << 2 | myblock->fd_drive) == error) return (error);
 return (get_fdc_status(myblock));
}

/*=======================================================

Wait for the head to settle after a seek.

-------------------------------------------------------*/

void wait_for_head(fdisk_regs *myblock)
{
 unsigned char wait;

 wait = get_parm(myblock,9);   /* get head settle */
 if ((MOTOR_STATUS & 0x80) != 0)  // if write
 {     /* yes */
  if (wait == 0)   /* wait zero ? */
  {     /* yes */
   if ((peekb40(DISK_ID+(myblock->fd_drive)) >> 4) & 0x07)
   wait = 0x14;   /* default 360 head time */
   else
   wait = 0x0f;   /* default others head time */
  }
 }
 else
 {
  if (wait == 0)   /* is wait zero ? */
  return;    /* yes, return */
 }
 delay_call(1,wait); /* delay n milliseconds */
}

/*==========================================================

Checks for a media change, reset media changes and check
media changes.

Returns:
ok = media not changed, error = media changed

-------------------------------------------------------*/

unsigned char med_change(fdisk_regs *myblock)
{
 if (read_dskchng(myblock) == 0) return (ok);
 /* clear media established and media type */
 andb40(DISK_ID+myblock->fd_drive,~media_established);

 disable();
 MOTOR_STATUS &= ~(1 << myblock->fd_drive);
 /* turn off motor status */
 enable();
 motor_on(myblock);
 FDC_reset(myblock);
 myblock->fd_track = 0;
 seek(myblock);
 myblock->fd_track = 1;
 seek(myblock);
 DISK_STATUS = MEDIA_CHANGE;
 if (read_dskchng(myblock) != 0) DISK_STATUS = TIME_OUT;
 return (error);
}

/*=========================================================

Seek to the requested track and initialize the controller

-------------------------------------------------------*/

unsigned char fdc_init(fdisk_regs *myblock)
{
 motor_on(myblock);
 if (seek(myblock) == error) return (error);
 if (send_fdc(myblock->fd_fdc_command) == error) return(error);
 if (send_fdc(((myblock->fd_head <<2) & BIT2) | myblock->fd_drive) == error)
 return(error);
 return (ok);
}
/*=======================================================

Wait until an operation is complete, then accept the status
from the controller.
```

```
---------------------------------------------------------*/

unsigned char get_fdc_status(fdisk_regs *myblock)
{
 myblock->time_out_flag = wait_int();
 if (results() == error) return (error);
 if (myblock->time_out_flag == error)
 {
  if (DISK_STATUS == 0) return(ok); else return(error);
 }
 if ((peekb40(FDC_STATUS) & 0xc0) == 0)
 {
  if (DISK_STATUS == 0) return(ok); else return(error);
 }
 if ((peekb40(FDC_STATUS) & 0xc0) != 0x40) { DISK_STATUS |= BAD_FDC; return(error); }

 myblock->error_byte = peekb40(FDC_STATUS + 1);
 /* get controller status */
 if      ((myblock->error_byte & BIT7) != 0) {DISK_STATUS |= RECORD_NOT_FND;}
 else if ((myblock->error_byte & BIT5) != 0) {DISK_STATUS |= BAD_CRC;}
 else if ((myblock->error_byte & BIT4) != 0) {DISK_STATUS |= BAD_DMA;}
 else if ((myblock->error_byte & BIT2) != 0) {DISK_STATUS |= RECORD_NOT_FND;}
 else if ((myblock->error_byte & BIT1) != 0) {DISK_STATUS |= WRITE_PROTECT;}
 else if ((myblock->error_byte & BIT0) != 0) {DISK_STATUS |= BAD_ADDR_MARK;}
 else    {DISK_STATUS |= BAD_FDC;}
 if (DISK_STATUS != 0) return(error);
 return(ok);
}

/*=========================================================

calculate number of sectors that were actually transferred.

returns:
number of sectors transferred

---------------------------------------------------------*/

unsigned char calc_sectors (fdisk_regs *myblock)
{
 if (DISK_STATUS != 0) return (0);

 myblock->sectors_track = get_parm(myblock,4);
 myblock->end_track = peekb40(FDC_STATUS + 3);
 myblock->end_head = peekb40(FDC_STATUS + 4);
 myblock->end_sector = peekb40(FDC_STATUS + 5);
 myblock->num_sectors = 0;
 if (myblock->end_track != myblock->fd_track)
          myblock->num_sectors += myblock->sectors_track;
 if (myblock->end_head != myblock->fd_head)
          myblock->num_sectors += myblock->sectors_track;
 myblock->num_sectors += (myblock->end_sector - myblock->fd_sector);
 return(myblock->num_sectors);
}

/*======================================================*/
```

NINE

The Hard Disk Driver

The Hard Disk Driver is used to operate an optional hard disk drive. This driver supports the standard AT type of hard disk controller. Other disk interfaces such as SCSI, ESDI, and SMD may require drastically different hardware level interaction, but the Bios register interface will remain the same as in this driver. If you are developing any of these drivers, this module may be used as a starting point, since the functional requirements at the Bios function call interface remain the same.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios hard disk driver
*
* Version: 1.04
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: athard.c
*
* Language MSC 5.1
*
* Functional Description:
*
* Interrupt 13h - hard disk driver
*
* AH = 0 - do a reset on the controller, set recal needed
*
* AH = 1 - read the last operation status into AL
*
*-------------- for the following operations ------------
*
* DL = drive number (0x80-0x81)
* DH = head number
* CH = track number
* CL = sector number (except for Format)
* AL = number of sectors (except for Format)
* ES:BX - data buffer address (except for Format)
*
*
* AH = 2 - read sectors
*
* AH = 3 - write sectors
*
* AH = 4 - verify sectors
*
* The above commands return in AL the # of sectors succesfull
*
* AH = 5 - format track
*  ES:BX points to  (4 sector bytes) * # of sectors
*  Each field (1 per sector) =
*   C = Cylinder (track ) number
*   H = head number
*   R = sector number
*   N = number of bytes per sector encoded as:
*    00 =  128 bytes per sector
*    01 =  256 bytes per sector
*    02 =  512 bytes per sector
*    03 = 1024 bytes per sector
*
* AH = 6 - not used
*
* AH = 7 - not used
*
* AH = 8 - Return the Drive Parameters
*
* AH = 9 - Initialize the drive pair
*
* AH = 10 - Read Long
*
* AH = 11 - Write Long
*
* AH = 12 - Seek
```

```
*
* AH = 13 - Alternate disk reset
*
* AH = 14 - not used
*
* AH = 15 - not used
*
* AH = 16 - test drive ready
*
* AH = 17 - Recalibrate
*
* AH = 18 - not used
*
* AH = 19 - not used
*
* AH = 20 - controller diagnostic
*
* AH = 21 - read drive type
*
*
* Returns:
* Carry Flag Clear = good operation - AH = 0
* Carry Flag Set = Failure - AH = failure code
*
* Arguments:
*
* Return:
*
* Version History:
* 1.01
*  DMA overrun check was being called with the cx register instead of the ax register. Changed
*  calls to pass ax.
* 1.02
* Corrected previous error in check_overrun, (test for nr sectors = 0x7f).
* Corrected error in check_error_byte function
* 1.03
*  Changed init routine to check for any disks present before linking
*  vector 40h.
* 1.04
*  Replaced peeks and pokes with casts
*  Moved variables to myblock
*  Changed setup to skip vectoring if no hard disks present
*  Changed most function calls to pass only myblock pointer for arguments
*  Corrected error in FORMAT case (using wrong block index)
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

/*================ function prototypes ===================*/

unsigned hdisk_setup(void);
void interrupt cdecl far hdisk_io(interrupt_registers);
void interrupt cdecl far hdisk_isr (void);

unsigned char calc_hsectors(unsigned char,unsigned char,unsigned char);
unsigned wait_for_int(void);
unsigned not_busy(void);
unsigned check_overrun(unsigned,unsigned,unsigned);
unsigned wait_for_data(void);
unsigned check_hd_status(void);
unsigned check_status_byte(void);
unsigned check_error_byte(void);
unsigned load_command(unsigned);
unsigned hi_regs(unsigned long);

/* G L O B A L   V A R I A B L E S */

/* G L O B A L   C O N S T A N T S */

extern const unsigned char hdisk_table;

/* L O C A L   D E F I N I T I O N S */

#define OUTPUT_REQ_BIT 0x08
#define ecc_mode 0x02

#define send_command load_command(myblock)

//--------- bios ram usage definitions ----------

#define bad_bat 0x80
```

```
#define bad_chksm 0x40

//--------- error code definitions ------------

#define NO_ERROR 0x00
#define BAD_CMD  0x01
#define BAD_ADDRESS 0x02
#define BAD_SECTOR 0x04
#define BAD_RESET  0x05
#define BAD_PARMS 0x07
#define BAD_DMA  0x09
#define BAD_SECTOR_FLAG 0x0a
#define BAD_CYLINDER 0x0b
#define BAD_FORMAT 0x0d
#define BAD_CONTROL 0x0e
#define BAD_DMA_ARB 0x0f
#define BAD_ECC  0x10
#define BAD_ECC_OK 0x11
#define BAD_HDC  0x20
#define BAD_SEEK  0x40
#define TIME_OUT  0x80
#define NOT_READY 0xaa
#define UNDEF_ERR 0xbb
#define BAD_WRITE 0xcc
#define BAD_STATUS 0xe0
#define BAD_SENSE 0xff

//--------- function code definitions ---------

#define RESET  0x00
#define READ_STATUS 0x01
#define READ_SECTORS 0x02
#define WRITE_SECTORS 0x03
#define VERIFY_SECTORS 0x04
#define FORMAT_TRACK 0x05
#define BAD_FUNCTION 0x06
#define GET_PARAMETERS 0x08
#define INIT_DRIVE 0x09
#define READ_LONG 0x0a
#define WRITE_LONG 0x0b
#define SEEK 0x0c
#define ALT_RESET 0x0d
#define TEST_DRIVE_READY 0x10
#define RECALIBRATE 0x11
#define DIAGNOSTIC 0x14
#define GET_DISK_TYPE 0x15
#define PARK_HEADS 0x19
#define FORMAT_UNIT 0x1a
#define BAD_COMMAND 0x1b

#define TRY  0xff

//----------- I/O port definitions -----------

#define HD_PORT 0x1f0

// controller task register file (ports) (frame)

// for a port write -

// 0 - write data
// 1 - pre-comp
// 2 - sector count
// 3 - sector number
// 4 - low cyl
// 5 - high cyl
// 6 - size/drive/head
// 7 - command register

// for a port read
// 0 - read data
// 1 - error register
// 2 - sector count
// 3 - sector number
// 4 - low cyl
// 5 - high cyl
// 6 - size/drive/head
// 7 - status register

#define HD_REG_PORT 0x3f6

//----- controller internal command codes ------------

#define HDC_RECAL 0x10
#define HDC_READ 0x20
#define HDC_READ_LONG 0x22
```

```
#define HDC_WRITE 0x30
#define HDC_WRITE_LONG 0x32
#define HDC_VERIFY 0x40
#define HDC_FORMAT 0x50
#define HDC_INIT 0x60
#define HDC_SEEK 0x70
#define HDC_DIAG 0x90
#define HDC_SET_PARM 0x91

/* L O C A L   C O N S T A N T S */

//const char ndisk_table[1][1][16];

/* P R O G R A M */

variables
unsigned char hd_frame[8];    /* The command block is built in this frame */
unsigned hd_table_seg;                  /* pointers to drive parameter table */
unsigned hd_table_off;
unsigned hd_xfr_seg;                    /* pointers to the transfer area */
unsigned hd_xfr_off;
end_variables hdregs;

/*-------------- setup the controller -----------*/

unsigned hdisk_setup()
{
 unsigned error_code = ok;
 hdregs *myblock;
 unsigned char drive_type;
 myblock = acquire_block(HARD);

 HD_NUM = 0;    /* clear number of drives */
 HD_CONTROL = 0;/* clear control byte */
 HD_STATUS1 = 0;/* clear disk status */

 if (incmos(0x8e) & (bad_bat | bad_chksm) != 0)
 {
  outcmos(0x8e,incmos(0x8e) & ~0x10); error_code = error;
 }
 else
 {
  /* get disk type for drive 1 from cmos */

  drive_type = incmos(0x12) >> 4;
  if (drive_type != 0) /* there are 1 or more drives */
  {

   VECTOR_40 = VECTOR_13;     /* move floppy vector to interrupt 40 */

   link_interrupt(0x13,hdisk_io); /* link 13 for functions and 76 (IRQ14) for isr */

   link_interrupt(0x76,hdisk_isr);

   set_vector(0x41,bios_cs(),&hdisk_table); /* set default table addresses to type 0 */
   set_vector(0x46,bios_cs(),&hdisk_table);

   outportb(0xa1,inportb(0xa1) & ~0x40); /* enable IRQ 14 */

   outportb(0x21,inportb(0x21) & ~0x05); /* enable IRQ 2 in slave mode */

   if(drive_type == 15) drive_type = incmos(0x19); /* get extended type */

   /* set pointer to table[type] into int 41 vector */
   poke(0x00,0x41 * 4,&hdisk_table+((drive_type-1)*16));
   HD_NUM = 1; // say there is one drive

   drive_type = incmos(0x12) & 0x0f;
   if (drive_type != 0) /* there are 1 or more drives */
   {
    if(drive_type == 15) drive_type = incmos(0x1a); /* get extended type */

    /* set pointer to table[type] into int 46 vector */
    poke(0x00,0x46 * 4,&hdisk_table+((drive_type-1)*16));
    HD_NUM = 2; // say there are two drives
   }

   /* test the hard disk controller */
   myblock->dx = 0x0080;
   myblock->ax = DIAGNOSTIC << 8;
   sys_int(0x13,myblock);
   if ((myblock->flags & carry_bit) != 0)
   {
    error_code = 0x1782; /* return error code */
   }
   /* do a reset on the drive(s) */
```

```
    myblock->dx = 0x0080;
    myblock->ax = RESET << 8;
    sys_int(0x13,myblock);
    if ((myblock->flags & carry_bit) != 0)
    {
     error_code = 0x1790; /* return error code */
    }
   }
  }
 release_block(myblock);
 return(error_code);
}
/*===============================================*/
/*======== BEGINNING OF MAIN ROUTINES ============*/
/*===============================================*/

/* handle the floppy disk function service call */

void interrupt cdecl far hdisk_io (interrupt_registers)
{
 hdregs *myblock;

 if((dx & 0x0080) == 0)                /* if dl <= 7f, then divert to floppy disk */
 {
  // fdisk_chain is an assembly language routine which
  // unstacks the pushed registers, then executes an int 40
  // to the floppy driver. This prevents the stack from
  // becoming too deep. DOS has a problem with its stack
  // being too short to support many nested,interrupts.
  fdisk_chain(); // does not return here !!!
 }

 enable();
 myblock = acquire_block(HARD);

 snapshot_in(HARD,&es);

 flags &= ~0x0001;  /* clear the carry flag for returns */

 HD_STATUS1 = 0;  /* clear status */

 // set precomp - varies for drive zero or one !!!
 if ((dx & 0xff) > 0x81) ax = BAD_COMMAND << 8;   /* bad drive number */

 if(HD_NUM == 0)  ax = BAD_COMMAND << 8;          /* no hard drives */

 if((dx & 0x7f) > (HD_NUM - 1)) ax = BAD_COMMAND << 8; /* bad drive number */

 if ((dx & 0x01) == 0)  // drive 0:
 {
  myblock->hd_table_seg = peek(0x00,0x0106);
  myblock->hd_table_off = peek(0x00,0x0104);
 }
 else    // drive 1:
 {
  myblock->hd_table_seg = peek(0x00,0x011a);
  myblock->hd_table_off = peek(0x00,0x0118);
 }
 // set the control byte
 HD_CONTROL = (HD_CONTROL & 0xc0) | peekb(myblock->hd_table_seg,myblock->hd_table_off+8);
 // set the register port
 outportb(HD_REG_PORT,peekb(myblock->hd_table_seg, myblock->hd_table_off+8));
 // build the frame
 myblock->hd_frame[1] = peek(myblock->hd_table_seg, myblock->hd_table_off+5) >> 2;
 myblock->hd_frame[2] = ax & 0x00ff;                // sector count
 myblock->hd_frame[3] = cx & 0x3f;                  // sector number
 myblock->hd_frame[4] = cx >> 8;                    // cyl low
 myblock->hd_frame[5] = (cx >> 6) & 0x03;           // cyl high
 myblock->hd_frame[6] = ((dx & 0x01) << 4) | ((dx >> 8) & 0x0f) | 0xa0;
 myblock->hd_frame[7] = 0;                          // pre-clear this item

 // normalize es:bx to xfr_seg:xfr_off = xxxx:000x
 myblock->hd_xfr_seg = es + (bx >> 4);
 myblock->hd_xfr_off = bx & 0x000f;

 /* now execute the CASE for the selected function */

 switch (ax >> 8)
 {
  case RESET:  /* reset the disk controller */
  watch_string(HARD,"RESET");
  myblock->ax = 0; // do a reset on the Floppy controller first
  sys_int(0x40,myblock);
  if ((dx & 0x00ff) > 0x81)  // not legal drive #
  {
   ax = myblock->ax;   // pass results back to caller
 flags = myblock->flags;
}
else     // do the hard drive too
{
 // enable the IRQ
 outportb(0xa1,inportb(0xa1) & 0xbf);
 // reset the controller
 outportb(HD_REG_PORT, 4);

 // DELAY 5-10 USECS
 delay_call(0,10);

 // set the head, and clear the reset bit
 outportb(HD_REG_PORT,HD_CONTROL & 0x0f);
 if(not_busy() == error)
 {
  HD_STATUS1 = BAD_RESET;
  watch_string(HARD,"NOT BUSY BAD RESET");
 }
 else
 {
  watch_string(HARD,"Not busy was OK");
  if(inportb(HD_PORT+1) != 0x01)
  {
   HD_STATUS1 = BAD_RESET;
   watch_string(HARD,"Port 1 not = 01 bad reset");
  }
  else
  {
   watch_string(HARD,"Init and Recal the drives");
   myblock->ax = INIT_DRIVE << 8;
   myblock->dx = 0x0080;
   sys_int(0x13,myblock);

   myblock->ax = RECALIBRATE << 8;
   myblock->dx = 0x0080;
   sys_int(0x13,myblock);
   // if two drives, do the next one
   if (HD_NUM > 1)
   {
    myblock->ax = INIT_DRIVE << 8;
    myblock->dx = 0x0081;
    sys_int(0x13,myblock);

    myblock->ax = RECALIBRATE << 8;
    myblock->dx = 0x0081;
    sys_int(0x13,myblock);
   }
   HD_STATUS1 = 0;  // clear the error bits
   watch_string(HARD,"Good reset");
  }
  HD_STATUS1 = 0;  // clear the error bits
  watch_string(HARD,"Good reset");
 }
}
break;

case READ_STATUS:
watch_string(HARD,"READ_STATUS");
ax = (ax & 0xff00) | HD_STATUS1;
break;

case READ_SECTORS:
watch_string(HARD,"READ_SECTORS");
myblock->hd_frame[7] = HDC_READ;
if(myblock->hd_frame[2] == 0) { HD_STATUS1 = BAD_CMD; break; }

watch_string(HARD,"\n\rChecking Overrun");
if(check_overrun(myblock->hd_frame[2],myblock->hd_xfr_off,myblock->hd_frame[7]) == ok)
{
 watch_string(HARD,"\n\rOverrun ok");

 if(send_command == ok)
 {
  watch_string(HARD,"\n\rSent command");
  // ouput_commands
  // do this loop for each sector
  do
  {
   if(wait_for_int() == ok)
   // wait for data ready interrupt
   {
    watch_string(HARD,"\n\rReady for data");
    // do the sector transfer
    rep_in(myblock->hd_xfr_seg, myblock->hd_xfr_off,HD_PORT,256);
    myblock->hd_xfr_off +=512;
```

```
     }
     watch_string(HARD,"\n\rXFR done");
     if (check_hd_status() == error) break;
     watch_string(HARD,"\n\rStatus was OK");
    }
    while(--myblock->hd_frame[2] != 0);
   }
  }

  break;

  case WRITE_SECTORS:
  watch_string(HARD,"WRITE_SECTORS");
  myblock->hd_frame[7] = HDC_WRITE;

  if(myblock->hd_frame[2] == 0) { HD_STATUS1 = BAD_CMD; break; }

  // check for transfer overrun
  if(check_overrun(myblock->hd_frame[2],myblock->hd_xfr_off,myblock->hd_frame[7]) == ok)
  {
   watch_string(HARD,"\n\rOverrun ok");
   if(send_command == ok)
   {
    watch_string(HARD,"\n\rSent command");

    // wait for data request
    if(wait_for_data() == ok)
    {
    watch_string(HARD,"\n\rReady for data");
     // do this loop for each sector
     do
     {
      // do the sector transfer
      rep_out(myblock->hd_xfr_seg,myblock->hd_xfr_off,HD_PORT,256);
      myblock->hd_xfr_off += 512;
      watch_string(HARD,"\n\rXFR done");
      if(wait_for_int() != ok) break; // wait for completion
      watch_string(HARD,"\n\rCompletion Interrupt rx'd");
      if (check_hd_status() == error) break;
      watch_string(HARD,"\n\rStatus was OK");
     }
     while(((HD_STATUS & 0X08) != 0) && (--myblock->hd_frame[2] != 0));
     if(inportb(HD_PORT+2) != 0)
{
watch_string(HARD,"\n\rXfer Aborted = original error code = ");
watch_word(HARD,HD_STATUS1);
HD_STATUS1 = UNDEF_ERR;
}
    }
   }
  }
  break;

  case VERIFY_SECTORS:
  watch_string(HARD,"VERIFY_SECTORS");
  myblock->hd_frame[7] = HDC_VERIFY;
  if(send_command == ok)
  {
   if (wait_for_int() == ok)
   {
    check_hd_status();
   }
  }
  break;

  case FORMAT_TRACK:
  watch_string(HARD,"FORMAT_TRACK");
  myblock->hd_frame[7] = HDC_FORMAT;
  myblock->hd_frame[2] = peekb(myblock->hd_table_seg, myblock->hd_table_off+14);
  if(send_command == ok)
  {
   watch_string(HARD,"\n\rSent command");

   // wait for data request interrupt
   if(wait_for_data() == ok)
   {
    watch_string(HARD,"\n\rReady for data");
    // do this loop for each sector
    do
    {
     // do the sector transfer
     rep_out(myblock->hd_xfr_seg,myblock->hd_xfr_off,HD_PORT,256);
     myblock->hd_xfr_off += 512;
     watch_string(HARD,"\n\rXFR done");
     if(wait_for_int() != ok) break; // wait for completion
     watch_string(HARD,"\n\rCompletion Interrupt rx'd");
    if (check_hd_status() == error) break;
    watch_string(HARD,"\n\rStatus was OK");
   }
   while(((HD_STATUS & 0x08) != 0) && (--myblock->hd_frame[2] != 0));
   // if the controller is still asking for more data,then it is an error
   if(inportb(HD_PORT+2) != 0) HD_STATUS1 = UNDEF_ERR;
  }
 }
 break;


 case GET_PARAMETERS:
 watch_string(HARD,"GET_PARAMETERS");

 // build ch = max cyls, cl = max sectors
 // get lower 8 bits of max cyls
 ax = peek(myblock->hd_table_seg,myblock->hd_table_off)-1;
 cx = ax << 8;
 cx |= ((ax >> 2) & 0x00c0);

 // get max sectors
 cx |= (peekb(myblock->hd_table_seg,myblock->hd_table_off+14));

 // build dh = max heads, dl = # drives
 dx = (peekb(myblock->hd_table_seg,myblock->hd_table_off+2)-1) << 8;
 dx |= HD_NUM;

 ax = 0;
 break;

 case INIT_DRIVE:
 watch_string(HARD,"INIT_DRIVE");
 myblock->hd_frame[7] = HDC_SET_PARM;
 // set max heads
 myblock->hd_frame[6] = (myblock->hd_frame[6] & 0xf0) | \
 (peekb(myblock->hd_table_seg,myblock->hd_table_off+2)-1);
 // set sector count
 myblock->hd_frame[2] = peekb(myblock->hd_table_seg,myblock->hd_table_off+14);
 // set low cylinder = 0
 myblock->hd_frame[4] = 0;
 // set high cylinder = 0
 myblock->hd_frame[5] = 0;

 if(send_command == ok)
 {
  if(not_busy() == ok)
  {
   check_hd_status();
  }
 }
 break;

 case READ_LONG:
 watch_string(HARD,"READ_LONG");
 myblock->hd_frame[7] = HDC_READ_LONG;

 if(myblock->hd_frame[2] == 0) { HD_STATUS1 = BAD_CMD; break; }

 // check for transfer overrun
 // watch_string(HARD,"\n\rChecking Overrun");
 if(check_overrun(myblock->hd_frame[2],myblock->hd_xfr_off,myblock->hd_frame[7]) == ok)
 {
  //   watch_string(HARD,"\n\rOverrun ok");
  // command phase
  if(send_command == ok)
  {
   //    watch_string(HARD,"\n\rSent command");
   // ouput_commands
   // do this loop for each sector
   do
   {
    if(wait_for_int() == ok)
    // wait for data ready interrupt
    {
     watch_string(HARD,"\n\rReady for data");
     // do the sector transfer
     rep_in(myblock->hd_xfr_seg,myblock->hd_xfr_off,HD_PORT,256);
     myblock->hd_xfr_off +=512;

     // read the 4 extra bytes
     pokeb(myblock->hd_xfr_seg,myblock->hd_xfr_off++,inportb(HD_PORT));
     pokeb(myblock->hd_xfr_seg,myblock->hd_xfr_off++,inportb(HD_PORT));
     pokeb(myblock->hd_xfr_seg,myblock->hd_xfr_off++,inportb(HD_PORT));
     pokeb(myblock->hd_xfr_seg,myblock->hd_xfr_off++,inportb(HD_PORT));

    }
  //      watch_string(HARD,"\n\rXFR done");
  if (check_hd_status() == error) break;
  //      watch_string(HARD,"\n\rStatus was OK");
 }
 while(--myblock->hd_frame[2] > 0);
}
}

break;


case WRITE_LONG:
watch_string(HARD,"WRITE_LONG");
myblock->hd_frame[7] = HDC_WRITE_LONG;

if(myblock->hd_frame[2] == 0) { HD_STATUS1 = BAD_CMD; break; }

// check for transfer overrun
if(check_overrun(myblock->hd_frame[2],myblock->hd_xfr_off,myblock->hd_frame[7]) == ok)
{
 watch_string(HARD,"\n\rOverrun ok");
 if(send_command == ok)
 {
  watch_string(HARD,"\n\rSent command");

  // wait for data request interrupt
  if(wait_for_data() == ok)
  {
   watch_string(HARD,"\n\rReady for data");
   // do this loop for each sector
   do
   {
    // do the sector transfer
    rep_out(myblock->hd_xfr_seg,myblock->hd_xfr_off,HD_PORT,256);
    myblock->hd_xfr_off += 512;
    // ecc mode - send 4 more bytes
    outportb(HD_PORT,peek(myblock->hd_xfr_seg,myblock->hd_xfr_off++));
    outportb(HD_PORT,peek(myblock->hd_xfr_seg,myblock->hd_xfr_off++));
    outportb(HD_PORT,peek(myblock->hd_xfr_seg,myblock->hd_xfr_off++));
    outportb(HD_PORT,peek(myblock->hd_xfr_seg,myblock->hd_xfr_off++));
    watch_string(HARD,"\n\rXFR done");
    if(wait_for_int() != ok) break; // wait for completion
    watch_string(HARD,"\n\rCompletion Interrupt rx'd");
    if (check_hd_status() == error) break;
    watch_string(HARD,"\n\rStatus was OK");
   }
   while(((HD_STATUS & 0x08) != 0) && (--myblock->hd_frame[2] != 0));
   // if the controller still asking for more then an error.
   if(inportb(HD_PORT+2) != 0) HD_STATUS1 = UNDEF_ERR;
  }
 }
}
break;

case SEEK:
watch_string(HARD,"SEEK");
myblock->hd_frame[7] = HDC_SEEK;
if(send_command == ok)
{
 if(wait_for_int() == ok) check_hd_status();
 if(HD_STATUS1 == BAD_SEEK) HD_STATUS1 = 0;
}
break;

case ALT_RESET:
watch_string(HARD,"ALT_RESET");
break;

case TEST_DRIVE_READY:
watch_string(HARD,"TEST_DRIVE_READY");
if(not_busy() == ok)
{
 outportb(HD_PORT+6,myblock->hd_frame[6]);
 check_hd_status();
}
break;

case RECALIBRATE:
watch_string(HARD,"RECALIBRATE");
myblock->hd_frame[7] = HDC_RECAL;
if(send_command == ok)
{
 if(wait_for_int() == ok)
 {
  check_hd_status();
 }
```

```
   else
   {
    if(wait_for_int() == ok)
    check_hd_status();
   }
  }
  if(HD_STATUS1 == BAD_SEEK) HD_STATUS1 = 0;
  break;

  case DIAGNOSTIC:
  watch_string(HARD,"DIAGNOSTIC");
  /* make sure interrupts are disabled when changing the interrupt mask */
  disable();
  outportb(0xa1,inportb(0xa1) & ~ 0x40);
  outportb(0x21,inportb(0x21) & ~ 0x20);
  enable();
  if(not_busy() == ok) /* ok to start diag */
  {
   outportb(HD_PORT+7,HDC_DIAG);
   if (not_busy() == ok)   /* wait for completion */
   {
    if ((HD_ERROR = inportb(HD_PORT+1)) != 1)
    {
     HD_STATUS1 = BAD_HDC;
    }
   }
   else
   {
    HD_STATUS1 = TIME_OUT;
   }
  }
  break;

  case GET_DISK_TYPE:
  watch_string(HARD,"GET_DISK_TYPE");
  // cyls =    peek(hd_table_seg,hd_table_off)
  // heads =   peekb(hd_table_seg,hd_table_off+2)
  // sectors = peekb(hd_table_seg,hd_table_off+14)

  cx = hi_regs((peek(myblock->hd_table_seg,myblock->hd_table_off)-1) *
  peekb(myblock->hd_table_seg,myblock->hd_table_off+2) *
  peekb(myblock->hd_table_seg,myblock->hd_table_off+14));

  dx = (peek(myblock->hd_table_seg,myblock->hd_table_off)-1) *
  peekb(myblock->hd_table_seg,myblock->hd_table_off+2) *
  peekb(myblock->hd_table_seg,myblock->hd_table_off+14);

  ax = 0x0300;
  break;

  default : /* invalid function code - return bad code */
  HD_STATUS1 = BAD_CMD;
  ax = (BAD_CMD << 8) | (ax & 0xff);
  break;

 } /* end of switch */

 ax = (ax & 0x00ff) | (HD_STATUS1 << 8);
 if((ax & 0xff00) != 0) flags |= 1;

 if(watch(HARD))
 {
  // display port ins
  write_string("\n\rPort ins 1f1 = ");lbyte(inportb(HD_PORT+1));
  write_string(" 1f2 = "); lbyte(inportb(HD_PORT+2));
  write_string(" 1f3 = "); lbyte(inportb(HD_PORT+3));
  write_string(" 1f4 = "); lbyte(inportb(HD_PORT+4));
  write_string(" 1f5 = "); lbyte(inportb(HD_PORT+5));
  write_string(" 1f6 = "); lbyte(inportb(HD_PORT+6));
  write_string(" 1f7 = "); lbyte(inportb(HD_PORT+7));
  write_string("\n\r");
 }
 snapshot_out(HARD,&es);
 release_block(myblock);
}

/*====================================================*/

/* check for the controller not busy */
unsigned not_busy(void)
{
 /* get current count and add 5 secs to it */
 unsigned long delay_count = set_timeout_count(5 * 20);

 enable();
 // clear status byte
 HD_STATUS1 = NO_ERROR;

 do
 { if ((inportb(HD_PORT+7) & 0x80) == 0) return(ok); }
 while (TIMER_LONG < delay_count);

 // set time out error
 HD_STATUS1 = TIME_OUT;

 // return error condition to caller
 return(error);  /* time out */
}

/*=====================================================*/

void interrupt cdecl far hdisk_isr (void)
{
 HD_INT_FLAG = -1;
 outportb(0xa0,0x20);
 outportb(0x20,0x20);   /* send eoi */
}

/*======================================================

Wait for the hardware interrupt to occur. Time-out and
return if no interrupt.

-----------------------------------------------------*/

unsigned wait_for_int (void)
{
 /* this time out should be 3 secs */
 unsigned long delay_count = set_timeout_count(3 * 20);

 enable();

 do
 { if(HD_INT_FLAG != 0) { HD_INT_FLAG = 0; HD_STATUS1 = 0; return (ok); } }
 while ((TIMER_LONG < delay_count));
 HD_INT_FLAG = 0; HD_STATUS1 |= TIME_OUT;
 return(error);
}

/*======================================================

Wait for the data request. Time-out and
return if no request.

-----------------------------------------------------*/

unsigned wait_for_data (void)
{
 unsigned long delay_count = set_timeout_count(2 * 20);

 enable();
 /* this time out should be 2 secs */
 do
 { if((inportb(HD_PORT+7) & OUTPUT_REQ_BIT) != 0) { return(ok); } }
 while (TIMER_LONG < delay_count);
 HD_STATUS1 = TIME_OUT;
 return (error);
}
/*=========================================================

recalibrate the drive.

-----------------------------------------------------*/


//----------------------------------------------------

unsigned check_hd_status()
{
  if(check_status_byte() == error) return(error);
 if((inportb(HD_PORT+7) & 0x01) == 0) return(ok);
 return(check_error_byte());
}

//----------------------------------------------------
// This is a reminder of what is in the tables
//const char status_mask[5]={0x80,0x20,0x40,0x10,0x04};
//const char status_err [5]={0x00,0xcc,0xaa,0x40,0x11};
//const char status_key [5]={0x00,0x00,0x40,0x10,0x00};

unsigned check_status_byte()
```

```
{
 unsigned i;
 unsigned char temp;
 HD_STATUS1 = 0;              // start with a clear
 temp = HD_STATUS = inportb(HD_PORT+7);
 if(watch(HARD))
         { write_string("\n\rHD_STATUS = "); lbyte(temp); write_string("\n\r"); }
 if((temp & 0x80) != 0) return(ok);
 if((temp & 0x20) != 0) { HD_STATUS1 = 0xcc; return(error); }
 if((temp & 0x40) == 0) { HD_STATUS1 = 0xaa; return(error); }
 if((temp & 0x10) == 0) { HD_STATUS1 = 0x40; return(error); }
 if((temp & 0x04) != 0) { HD_STATUS1 = 0x11; }
 return(ok);
}

//-------------------------------------------------
const unsigned char error_table[8] = \
{
BAD_SECTOR,         /* 0x80 - bad block */
BAD_ECC,  /* 0x40 - bad data ecc */
UNDEF_ERR,          /* 0x20 - not used */
BAD_SECTOR,         /* 0x10 - id not found */
UNDEF_ERR,          /* 0x08 - not used */
BAD_CMD,  /* 0x04 - aborted command */
BAD_SEEK, /* 0x02 - trk 0 not found on recalibrate */
BAD_ADDRESS,        /* 0x01 - data address mark not found */
};

unsigned check_error_byte()
{
 unsigned i = 0, temp = 0;
 temp = HD_ERROR = inportb(HD_PORT+1);
 watch_string(HARD,"\n\rError Byte from 1F1 ="); watch_byte(HARD,temp);
 while((temp & 0xff) != 0)
 {
  if((temp & 0x80) != 0)
  {
   HD_STATUS1 = peekb(bios_cs(),&error_table[i]);
   watch_string(HARD,"\n\rError Byte returned ="); watch_byte(HARD,HD_STATUS1);
   return(error);
  }
  temp <<= 1; i++;
 };
 watch_string(HARD,"\n\rNo error detected");
 return (ok);
}

//----------------------------------------------------
// check for transfer overrun
// if ecc mode than use 7f04 as max, else use 8000

unsigned check_overrun(nr_sectors,xfr_off,ecc_char)
{
 if (nr_sectors < 0x7f) return(ok); // always ok

 if((ecc_char & ecc_mode) != 0)
 {
  if ((nr_sectors == 0x7f) && (xfr_off < 4)) return(ok);
 }
 else
 {
  if(nr_sectors == 0x7f) return(ok);
  if((nr_sectors == 0x80) && (xfr_off == 0)) return(ok);
 }
 HD_STATUS1 = BAD_DMA;
 return(error);
}

//-------------------------------------------------

unsigned load_command(hdregs *myparm)
{
 unsigned count = 10000;
// hdregs *myparm;
// myparm = arg;

 // check for time out on drive being ready
 // watch_string(HARD,"Load command entry\n\r");
 do
 {
  // check for controller and drive ready
  if (not_busy() == ok)
  {
   // watch_string(HARD,"not_busy OK\n\r");
   // select drive
   outportb(HD_PORT+6,myparm->hd_frame[6]);
```

```
    if (check_status_byte() == ok)
    {
     //    watch_string(HARD,"status byte was ok\n\r");
     // reset the interrupt flag byte
     HD_INT_FLAG = 0;

     // enable the interrupts
     disable();
     outportb(0xa1,inportb(0xa1) & 0xbf);
     outportb(0x21,inportb(0x21) & 0xfb);
     enable();

     // check for retries
     if((HD_CONTROL & 0xc0) != 0)
     {
      if(((myparm->hd_frame[7] & 0xf0) >= 0x20) && \
      ((myparm->hd_frame[7] & 0xf0) <= 0x40))
      myparm->hd_frame[7] |= 0x01;
     }
     if(watch(HARD))
     {
      // output the commands

      write_string("\n\rPort outs 1f1 = ");
      lbyte(myparm->hd_frame[1]);
      write_string(" 1f2 = "); lbyte(myparm->hd_frame[2]);
      write_string(" 1f3 = "); lbyte(myparm->hd_frame[3]);
      write_string(" 1f4 = "); lbyte(myparm->hd_frame[4]);
      write_string(" 1f5 = "); lbyte(myparm->hd_frame[5]);
      write_string(" 1f6 = "); lbyte(myparm->hd_frame[6]);
      write_string(" 1f7 = "); lbyte(myparm->hd_frame[7]);
      write_string("\n\r");
     }
     outportb(HD_PORT+1,myparm->hd_frame[1]);
     outportb(HD_PORT+2,myparm->hd_frame[2]);
     outportb(HD_PORT+3,myparm->hd_frame[3]);
     outportb(HD_PORT+4,myparm->hd_frame[4]);
     outportb(HD_PORT+5,myparm->hd_frame[5]);
     outportb(HD_PORT+6,myparm->hd_frame[6]);
     outportb(HD_PORT+7,myparm->hd_frame[7]);
     return(ok);
    }
   }
  }
 while(count-- > 0);
 return(error);
}

/*====================================================*/
/*====================================================*/
/*====================================================*/
```

TEN

The Boot File

The Boot.c File is the bootstrap function call routine which is used to load DOS from a floppy disk device. The boot routine attempts to read the boot sector of the specified drive and if successful transfers control to the loaded routine (which in turn loads the remainder of the operating system).

To allow flexibility in the way the system attempts to boot, there is a "boot_list" in the boot program which defines the drives which will be used in the boot process. This list may specify the hard or floppy drives in any order.

If your Boot requirements are non-standard or specific to a dedicated application, you may wish to modify the standard boot routine. All the standard system initialization has been done by the POD when the boot routine is finally called. A specialized boot program may load an operating system or start executing an application directly. If one is not using an operating system to support an application, one must not design and code the application so that it makes reference to non-existent operating system features!

The boot routine should exit if the boot process is not completed. This will allow the POD routine to attempt an alternate path, such as defaulting to SysVue.

```
/*******************************************************************************************************
*
* Copyright (c) FOSCO 1988 - All Rights Reserved
*
* Module Name: AT Bios bootstrap routine
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 12-01-88
*
* Filename: atboot.c
*
* Language: MS C 5.1
*
* Functional Description:
*
* This module does the bootstrap load to get DOS loaded and running.
*
* We look at the boot list to determine what drives should be used for the boot attempts.
* The boot sector is loaded into 0000:7c00 and control is transferred to that address.
* If the boot cannot be performed, then we exit.
*
* Arguments:
*
* Return:
*
* Version History:
*
*******************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

unsigned boot_setup(void);
void interrupt cdecl far boot(void);
void boot_jump(unsigned,unsigned,unsigned);

/* G L O B A L   V A R I A B L E S */
/* G L O B A L   C O N S T A N T S */

// this is the list of possible boot devices, in the order in which they should be tried.
// If all the named drives are tried and fail, then exit
// The list is terminated by a -1

const unsigned boot_list[] ={0x00,0x80,0x01,0x81,-1,};

//This example shows the boot preference order to be:
// Drives A:, C:, B:, and D:

// Programmers Caution:
// Some versions of DOS may not allow booting from B: or D: due to DOS internal programs
// defaulting to A: or C:, regardless of the drive number being passed in the  DL register
// to the DOS boot sector routine.

// Programmers Note:
// This version of boot checks for a particular signature in the last two (2) bytes of the boot
// sector. This seems to be an IBM or Microsoft convention on their boot records. Boot records of
// operating systems other than IBM or MS may not have this signature. If you wish to modify the
// boot module to accept any value for the signature, you may delete the signature comparison logic
// accordingly.


/* L O C A L   D E F I N I T I O N S */

#define boot_retrys 2
#define reset_flag 0x72
#define switch_byte 0x10

/* P R O G R A M */

unsigned boot_setup(void)
{
 link_interrupt(0x19,boot);
 return(ok);
}

//----------- The boot routine ------------------

variables
end_variables bootregs;

void interrupt cdecl far boot(interrupt_registers)
{
 unsigned i,break_flag,boot_drive,boot_index;
 bootregs *myblock;
 myblock = acquire_block(BOOT);
 snapshot_in(BOOT,&es);

 //write_string("----------- Booting ------------\n\r");

 // clear the boot area
 for (i = 0x7c00; i < 0x7e00; pokeb(0x0000,i++,0));

 for(boot_index = 0; peekcs(&boot_list[boot_index]) != -1; boot_index++)
 {
  for (i=0;i < boot_retrys; i++)
  {
   boot_drive = peekcs(&boot_list[boot_index]);

   watch_string(BOOT,"\n\rBooting from Drive # ");
   watch_byte(BOOT,boot_drive);
   watch_string(BOOT,"\n\rAttempt # ");
   watch_byte(BOOT,i);

   myblock -> ax = 0x0000;  // try a reset
   myblock -> dx = boot_drive;
   sys_int(0x13,myblock);
   /* If no carry returned on reset then try booting */
   if((myblock -> flags & carry_bit) == 0)
   {
    myblock -> ax = 0x0201;  /* try to read boot sector */
    myblock -> bx = 0x7c00;
    myblock -> es = 0x0000;
    myblock -> dx = boot_drive;
    myblock -> cx = 0x0001;
    sys_int(0x13,myblock);
    if((myblock -> flags & carry_bit) == 0)
    /* no carry returned on read */
    {
     /* check for boot sector signature */
          /* See programmers note above */
     if (peek(0x0000,0x7dfe) == 0xaa55)
     {
      RESET_FLAG = 0;  /* clear reset flag */
      enable();
      release_block(myblock);
      watch_string(BOOT,"\n\rJumping to boot sector\n\r");
      /* go and don't return */
      boot_jump(0x0000,0x7c00,boot_drive);
     }
     else
     {
      watch_string(BOOT,"\n\rSignature was ");
      watch_word(BOOT,peek(0x0000,0x7dfe));
     }
    }
    else
    {
// check for a timeout on the disk controller
 if((myblock->ax & 0x8000) != 0) i = boot_retrys + 1;
     watch_string(BOOT,"\n\rCarry returned on Read");
    }
   }
   else
   {
    watch_string(BOOT,"\n\rCarry returned on Reset");
   }
  }
 }
 watch_string(BOOT,"\n\rRetrys exhausted");
 snapshot_out(BOOT,&es);
 release_block(myblock);
}

/*======================================================*/
```

ELEVEN

The Cassette Driver

Although an AT Type machine does not have a cassette interface, a driver is provided in the event a program makes a cassette function call. The function call returns a cassette error condition to the caller. The A-Type machines use an extended set of functions codes for AT specific functions for joystick input, virtual mode control, multi-tasking hooks, sys-request functions, etc.

```
/****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios cassette function
*
* Version: 1.01
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: atcass.c
*
* Functional Description:
*  This module returns an error condition in AX in the event a user program calls this function.
*
* Version History:
* 1.01
*  Replaced peeks and pokes with casts
*
****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

void interrupt cdecl far cassette_io(interrupt_registers);

/* G L O B A L S */

extern unsigned char config_table;
extern _cassette_io;

/* L O C A L   D E F I N T I O N S */

#define OPEN_DEVICE 0x80
#define CLOSE_DEVICE 0x81
#define PROGRAM_TERMINATE 0x82
#define EVENT_WAIT 0x83
#define JOYSTICK_SUPPORT 0x84
#define SYS_REQ_KEY 0x85
#define WAIT 0x86
#define MOVE_BLOCK 0x87
#define GET_EXTENDED_MEMORY_SIZE 0x88
#define SWITCH_TO_PROTECTED_MODE 0x89
#define DEVICE_BUSY 0x90
#define INTERRUPT_COMPLETE 0x91
#define RETURN_SYSTEM_CONFIG_PARAMETERS 0xC0

#define rtc_wait_flag 0xa0

/* P R O G R A M */

//-----------------------------------------------------

unsigned cassette_setup()
{
 link_interrupt(0x15,&_cassette_io); return(ok);
}

//-----------------------------------------------------

void interrupt cdecl far cassette_io(interrupt_registers)
{
 enable();
 setds_system_segment();
```

```
snapshot_in(CASSETTE,&es);

switch (ax >> 8)
{
 case OPEN_DEVICE:
 watch_string(CASSETTE,"Open Device");
 break;

 case CLOSE_DEVICE:
 watch_string(CASSETTE,"Close Device");
 break;

 case PROGRAM_TERMINATE:
 watch_string(CASSETTE,"Program Terminate");
 break;

 case EVENT_WAIT:
 watch_string(CASSETTE,"Event Wait");
 if((RTC_WAIT_FLAG & 0x01) != 0)
 {
  flags |= carry_bit;
 }
 else
 {
  outportb(0xa1,inportb(0xa1) & 0xfe);
  USER_FLAG_SEG = es;
  USER_FLAG = bx;
  RTC_LOW = cx;
  RTC_HIGH = dx;
  RTC_WAIT_FLAG = 1;
  // enable PIE in cmos chip
  outcmos(0x0b,(incmos(0x0b) & 0x7f) | 0x40);
 }
 break;

 case JOYSTICK_SUPPORT:
 watch_string(CASSETTE,"Joystick");
 break;

 case SYS_REQ_KEY:
 watch_string(CASSETTE,"Sys Req Key");
 break;

 case WAIT:
 watch_string(CASSETTE,"Wait");
 if((RTC_WAIT_FLAG & 0x01) != 0) flags |= carry_bit;
 else
 {
  disable();
  outportb(0xa1,inportb(0xa1) & 0xfe);
  USER_FLAG_SEG = ds;
  USER_FLAG = rtc_wait_flag;
  RTC_HIGH = cx;
  RTC_LOW = dx;
  RTC_WAIT_FLAG = 1;
  // enable PIE in cmos chip
  outcmos(0x0b,(incmos(0x0b) & 0x7f) | 0x40);
  enable();

  do { } while((RTC_WAIT_FLAG & 0x80) == 0);
  RTC_WAIT_FLAG = 0;
 }
 break;

 case MOVE_BLOCK:
 watch_string(CASSETTE,"Move Block");
 // format = virtual_move(word_count,global_segment,global_offset)
 ax = virtual_move(cx,es,si);
 if (ax != 0) flags |= carry_bit;
 break;

 case GET_EXTENDED_MEMORY_SIZE:
 watch_string(CASSETTE,"Get Ext Mem Size");
 ax = incmos(0x31);
 ax = ax << 8;
 ax |= incmos(0x30);
 break;

 case SWITCH_TO_PROTECTED_MODE:
 watch_string(CASSETTE,"Switch to Virtual");
 // format = virtual_mode(index,global_segment,global_offset);
 ax = virtual_mode(bx,es,si);
 break;

 case DEVICE_BUSY:
    watch_string(CASSETTE,"Device Busy");
    flags &= ~0x0001;
    break;

    case INTERRUPT_COMPLETE:
    watch_string(CASSETTE,"Interrupt Complete");
    break;

    case RETURN_SYSTEM_CONFIG_PARAMETERS:
    watch_string(CASSETTE,"Open Device");
    es = bios_cs();
    bx = &config_table;
    flags &= ~carry_bit;
    ax &= 0x00ff;
    break;

    default:
    watch_string(CASSETTE,"Default");
    ax = (ax & 0x00ff) | 0x8600;
    flags |= carry_bit;
    break;
   }
   snapshot_out(CASSETTE,&es);
  }
```

TWELVE
The Parallel Printer Driver

The Parallel Printer Driver is an Interrupt (0x17) function call that is used to communicate with the printer. A parallel printer is one which has a "Centronics" interface and is referred to as LPT1, 2, or 3. The printer driver provides simple commands to initialize the printer or send a character to the printer.

Although a hardware interrupt line (IRQ7) is normally assigned for the parallel printer port, interrupt operation is not supported by the standard Bios driver. The typical printer adapter has logic to support the use of the interrupt, but it is not clear how the interrupt would be used if there is more than one printer port installed. Some print spooling programs may possibly use the interrupt, but we are not aware of it. There may also be logic in some printer adapters to disable the output data latch, thereby allowing a bi-directional data capability. The control logic to use such a feature is not a standard Bios function.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios LPT driver
*
* Version: 1.02
*
* Author: FOSCO
*
* Date: 2-25-89
*
* Filename: atlpt.c
*
* Language: MS C 5.1
*
* Functional Description:
*
* Int 0x17 - parallel printer driver
*
* Input:
*
* AH = 0 print the character in AL
* on return, ah = 1 if character could not be printed
* (time out).
* Other bits set as on normal status call
*
* AH = 1 initialize the printer port
* returns with AH set with printer status
*
* AH = 2 read the printer status into (ah)
*   7   6   5   4   3   2   1   0
*   |   |   |   |   |   |   |   +-- time out
*   |   |   |   |   |   |   +------ unused
*   |   |   |   |   |   +---------- unused
*   |   |   |   |   +-------------- 1 = i/o error
*   |   |   |   +------------------ 1 = selected
*   |   |   +---------------------- 1 = out of paper
*   |   +-------------------------- 1 = acknowledge
*   +------------------------------ 1 = not busy
*
* DX =  printer to be used (0,1,2)
*
*
* Arguments:
*
* Return:
*
* Version History:
* 1.01
*  Chnaged the time out loop and pre-cleared (AH) to insure returning status was clean.
* 1.02
*  Pre-cleared the port list before scanning to prevent phamtoms LPT's being assumed.
*
*****************************************************************************************************/
```

```
/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

unsigned lpt_setup(void);
void scan_lpt(unsigned);
void interrupt cdecl far printer_io(interrupt_registers);

/* G L O B A L   V A R I A B L E S */

extern unsigned lpt_list[4];
extern unsigned char lpt_timeout_list[4];

/* G L O B A L   C O N S T A N T S */

extern _printer_io;

/* L O C A L   D E F I N I T I O N S */

#define print_the_character 0
#define init_the_port 1
#define read_the_status 2
#define equipment       0x10

/* P R O G R A M */

unsigned lpt_setup()
{
 unsigned i;
 link_interrupt(0x17,&_printer_io);
 for (i=0;i<3;i++) poke40(&lpt_list[i],0); // pre-clear the list
 scan_lpt(0x3bc);
 scan_lpt(0x378);
 scan_lpt(0x278);
 for (i=0;i<3;i++) pokeb40(&lpt_timeout_list[i],0x14);
 return(ok);
}

void scan_lpt(unsigned port)
{
 unsigned i;
 outportb(port,0x55);
 if (inportb(port) == 0x55)
 {
  EQUIP_FLAG += 0x4000;
  for (i = 0; ((i < 3) && (peek40(&lpt_list[i]) != 0)); i++);
  poke40(&lpt_list[i],port);
 }
}

//=====================================================================

void interrupt cdecl far printer_io(interrupt_registers)
{
 unsigned port,i,j;

 setds_system_segment();
 snapshot_in(LPT,&es);

 if (dx <= 2)     /* valid lpt number ? */
 {
  port = peek40(&lpt_list[dx]); /* get port number */
  switch (ax >> 8 ) {  /* test for valid commands */

   case print_the_character:
   ax = ax & 0x00ff;
   watch_string(LPT,"Print Char");
   outportb (port,ax);      /* output character to port */

   // see if the printer is busy

   for (j = (peekb40(&lpt_timeout_list[dx]) & 0xff); j > 0;  j-- )
   {
    for (i= 1000;  i > 0 ;  i-- )
    {
     if ((inportb(port+1) & 0x80) != 0) break; /* if not busy */
    }
    if ((inportb(port+1) & 0x80) != 0) break; /* if not busy */
   }

   if ((inportb(port+1) & 0x80) != 0) /* if not busy */
   {
    outportb (port+2, 0x0d); /* set strobe high */
    outportb (port+2, 0x0c); /* set strobe low */
```

```
     ax = (((inportb(port+1) & 0xf8) ^ 0x48) << 8) | (ax & 0xff);
    }
    else // port was busy, so device not available
    {
     ax = (((ax | 0x0100 ) & 0xf9ff) ^ 0x48);  /* printer timed-out */
    }
    break;

    case init_the_port:
    ax = ax & ~0x00ff;
    watch_string(LPT,"Init Port");

    outportb (port+2, 0x08); /* set init line low */

    /* delay for reset to take place */
    for (i=0;  i <= 4000;  i++ ) ;

    outportb (port+2, 0x0c); /* set init line high */
    goto read_status;

    case read_the_status :
    ax = ax & ~0x00ff;
    watch_string(LPT,"Read Status");

    read_status:
    ax = (((inportb(port+1) & 0xf8) ^ 0x48) << 8) | ax;
   }
  }
  snapshot_out(LPT,&es);
 }
/*==========================================================================================*/
```

THIRTEEN

The Comm Driver

The Comm Driver is an Interrupt (0x14) Function Call that is used to communicate with the serial port adapters.

Note - The most common problem involved with serial devices on the Com ports is the incorrect use of the control signals, CTS, RTS, DTR, and DSR. If you experience problems in communication, try jumpering CTS to RTS, and DSR to DTR.
This may help to localize and correct the problem.

The extended function calls available on some PS/2 machines are not included in this standard driver. These are relatively simple to add if you wish to enhance your Com Driver.

Note - The standard serial port Bios driver does not support interrupt driven operation. Most tele-communication software packages will replace the Bios driver with their own driver.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios com port driver
*
* Version: 1.02
*
* Author: FOSCO
*
* Date: 2-25-89
*
* Filename: atcomm.c
*
* Language: MS C 5.1
*
* Functional Description:
* Interrupt 0x14 -- Serial Port Driver
*
* Provides a function call for initializing, sending data, receiving data, and reading the status
* of an asynchronous adapter.
*
* Upon   Entry:
*
* DX = Logical Async device # (0,1,2,3) for COM1,2,3,4
*
* Initialize the port:
* AH = 00h
* AL = initialization parameters:
*
* Bit     Meaning
* -----   -----------
*  Bits    7,6,5
*          0 0 0   110  Baud
*          0 0 1   150
*          0 1 0   300
*          0 1 1   600
*          1 0 0   1200
*          1 0 1   2400
*          1 1 0   4800
*          1 1 1   9600
*
*  Bits   4,3
*         0 0     No parity
*         0 1     Odd parity
*         1 0     No parity
*         1 1     Even parity
*
* Bit     2
*         0       One stop bit
*         1       Two stop bits
*
* Bits    1,0
*         0 0     5 bits/char
*         0 1     6 bits/char
*         1 0     7 bits/char
*         1 1     8 bits/char
*
* Send    a character:
* AH      = 01h
* AL      = the character to send
*
* Receive a character:
* AH      = 02h
*
* Read    the port status:
* AH      = 03h
*
* Upon   Exit:
*
* For     'Initialize the port' (AH=0):
* AH      = Line status
* Bit     Meaning if = 1
* ---     -------------
* 7       Time out
* 6       Tx Empty (TEMT)
* 5       Tx Holding Register Empty (THRE)
* 4       Break Interrupt (BI)
* 3       Framing Error (FE)
* 2       Parity Error (PE)
* 1       Overrun Error (OE)
* 0       Data Ready (DR)
*
* AL      = Modem status
* Bit     Meaning if = 1
* ---     -------------
* 7       Data Carrier Detect (DCD)
* 6       Ring Indicator (RI)
* 5       Data Set Ready (DSR)
* 4       Clear To Send (CTS)
* 3       Delta Data Carrier Detect (DDCD)
* 2       Trailing Edge Ring Indicator (TERI)
* 1       Delta Data Set Ready (DDSR)
* 0       Delta Clear To Send (DCTS)
*
* For 'Send a character (AH = 1):
* AL = unchanged (character that was sent)
* AH = Completion code
* 1XXX    XXXX - Time out, char not sent
* 0YYY    YYYY - Tx Successful,  Bits 6 - 0 = Line status
*
* For 'Receive a character (AH = 2):
* AL = The received char, if successful
* AH = Completion code
* 1XXX    XXXX - Time out, no char received
* 000Y    YYY0 - Data received, but error detected
* See     Line status Bit Definitions
*
* For 'Get port status' (AH=3):
* AH = Line status (see above)
* AL = Modem status (see above)
*
* Global Variables:
*
* Async_adapter_port_list (word * 4):
*
* Word 0 = adapter address of COM1
* Word 1 = adapter address of COM2
* Word 2 = adapter address of COM3
* Word 3 = adapter address of COM4
*
* These four words contain the hardware port addresses of the async adapters. Four async
* adapters can be supported although the operating system may only recognize COM 1,2.
*
* Async_time_out_count (byte * 4):
*
* Byte 0 = time out count value for COM1
* Byte 1 = time out count value for COM2
* Byte 2 = time out count value for COM3
* Byte 3 = time out count value for COM4
*
* The time outs for no response use the contents of these cells for a count.
*
* Registers Modified:
* AX is modified
```

```
*
* Arguments:
*
* Return:
*
* Version History:
*        1.01
*  Get_divisor corrected by using a peek to reference the baud rate table
*        1.02
*         Pre-cleared comm list before scanning to insure no phantoms present.
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

unsigned comm_setup(void);
void scan_comm(unsigned);
void interrupt cdecl far comm_io(interrupt_registers);

/* G L O B A L   V A R I A B L E S */

extern unsigned comm_list [4];
extern unsigned char comm_timeout_list [4];

/* G L O B A L   C O N S T A N T S */

const unsigned baud_table[8]={1047,768,384,192,96,48,24,12};
extern _comm_io;

/* L O C A L   D E F I N I T I O N S */

/* 8250/16450 register offsets from base address */

#define tx_data_reg 0x00
#define rx_data_reg 0x00
#define div_lsb_reg 0x00
#define div_msb_reg 0x01
#define int_enb_reg 0x01
#define int_id_reg 0x02
#define line_ctrl_reg   0x03
#define modem_ctrl_reg 0x04
#define line_stat_reg   0x05
#define modem_stat_reg  0x06

/* 8250/16450 register bit definitions */

#define txrdy    0x60
#define rxrdy    0x01
#define dsr_bit  0x20
#define dtr_bit  0x01
#define cts_bit  0x10
#define rts_bit  0x02

/* opcode definitions */

#define initialize 0x00
#define send_character 0x01
#define rx_character 0x02
#define read_status     0x03

#define equipment       0x10
#define timeout_bit 0x8000

/* P R O G R A M */

// SCAN Com Ports -
// This routine is called to determine how many serial ports are in the system

unsigned comm_setup(void)
{
 unsigned i;
 link_interrupt(0x14,&_comm_io);
 for (i=0;i<4;i++) pokew40(&comm_list[i],0); // insure list is clear before scanning
 scan_comm(0x3f8);
 scan_comm(0x2f8);
 scan_comm(0x3e8);
 scan_comm(0x2e8);
 for (i=0;i<4;i++) pokeb40(&comm_timeout_list[i],0x01);
 return(ok);
}

void scan_comm(unsigned port)
```

```
{
 unsigned i;
 outportb(port+3,0x55);
 if (inportb(port+3) == 0x55)
 {
  EQUIP_FLAG += 0x0200;
  for (i = 0;((i < 4) && (peek40(&comm_list[i]) != 0));i++);
  poke40(&comm_list[i],port);
 }
}


void interrupt cdecl far comm_io(interrupt_registers)
{
 unsigned port_address,divisor;
 unsigned time_out;
 unsigned long timeout_counter;
 setds_system_segment();

 snapshot_in(COM,&es);

 if (dx > 3) /* check for DX out of range */
 {
  watch_string(COM,"Bad Port");
 }
 else
 {
  /* get the time out parameter */
  time_out = peekb40(&comm_timeout_list[dx]);

  /* scale to use as a major loop counter */
  time_out *= 10; /* figure 1 count = 500 millisec = 10 clock ticks

  /* get the port address for the (DX) specified device */
  port_address = peek40(&comm_list[dx]);

  /* if no serial port for this logical device....*/
  if (port_address == 0)
  {
   /* return with a time-out error */
   ax |= timeout_bit;
  }
  else
  {
   switch (ax >> 8)
   {
    case initialize:       /* INITIALIZE THE PORT */
    ax &= 0x00ff;
    watch_string(COM,"Init Port");
    /* set divisor latch enable */
    outportb(port_address+line_ctrl_reg,0x80);
    /* get the divisor */
    divisor = peekcs(&baud_table[(ax >> 5) & 0x0007]);
    /* load the baud rate divisor into the 8250 */
    outportb(port_address+div_lsb_reg,divisor);
    outportb(port_address+div_msb_reg,(divisor >> 8));
    /* set length,parity,stop bits */
    outportb(port_address+line_ctrl_reg,(ax & 0x1f));
    /* set DTR and RTS lines */
    outportb(port_address+modem_ctrl_reg,(0x08 | dtr_bit | rts_bit));
    ax = inportb(port_address+line_stat_reg);
    ax = (ax << 8) | inportb(port_address+modem_stat_reg);
    break;

    case send_character:  /* SEND A CHARACTER */
    watch_string(COM,"Send Char");

    /* start with clean return status */
    ax &= 0x00ff;

    /* set dtr and rts */
    outportb(port_address+modem_ctrl_reg, inportb(port_address+modem_ctrl_reg) | dtr_bit | rts_bit);
    /* check dsr and cts */

    timeout_counter = set_timeout_count(time_out);

    while(((inportb(port_address+modem_stat_reg) & (dsr_bit | cts_bit)) != (dsr_bit | cts_bit)))
    {
     if (TIMER_LONG >= timeout_counter) {ax |= timeout_bit; break;}
    }
    if ((ax & 0xff00) == 0) /* OK - no timeout */
    {
     timeout_counter = set_timeout_count(time_out);

     while((inportb(port_address+line_stat_reg) & txrdy) != txrdy)
     {
        if (TIMER_LONG >= timeout_counter) {ax |= timeout_bit; break;}
       }
       if ((ax & 0xff00) == 0) /* OK - no timeout */
       {
        /* send the character */
        outportb(port_address+tx_data_reg,ax);
        /* return the status in AH */
        ax = (ax & 0x00ff) | (inportb(port_address+line_stat_reg) << 8);
       }
      }
      break;


      case    rx_character:       /* RECEIVE A CHARACTER */
      watch_string(COM,"Rx Char");

      /* start with clean return status */
      ax &= 0x00ff;

      /* set DTR out */
      outportb(port_address+modem_ctrl_reg, inportb(port_address+modem_ctrl_reg) | dtr_bit);

      /* check DSR */
      timeout_counter = set_timeout_count(time_out);
      while((inportb(port_address+modem_stat_reg) & dsr_bit) != dsr_bit))
      {
       if (TIMER_LONG >= timeout_counter) {ax |= timeout_bit; break;}
      }
      if ((ax & 0xff00) == 0) /* OK - no timeout */
      {

       /* check for data ready */
       timeout_counter = set_timeout_count(time_out);
       while((inportb(port_address+line_stat_reg) & rxrdy) != rxrdy))
       {
        if (TIMER_LONG >= timeout_counter) {ax |= timeout_bit; break;}
       }
       if ((ax & 0xff00) == 0) /* OK - no timeout */
       {
        ax = ((inportb(port_address+line_stat_reg) & 0x1e) << 8) | inportb(port_address+rx_data_reg);
       }
      }
      break;

      case read_status:         /* READ THE STATUS */
      watch_string(COM,"Read Status");

      ax = inportb(port_address+line_stat_reg);
      ax = (ax << 8) | inportb(port_address+modem_stat_reg);
      break;

      default:
      watch_string(COM,"Bad Command");
      ax |= timeout_bit;

     }
    }
   }
   snapshot_out(COM,&es);
  }

  /*=====================================================*/
```

FOURTEEN

The Timer Interrupt

The Timer Interrupt is called by the hardware real-time clock every 53 milliseconds to increment the time count. This count is used by DOS to calculate the time of day.

Note - A user supplied routine may be linked to the timer interrupt routine, using interrupt 1C. This user routine should be of minimal execution time, since other interrupts are disabled while it is running. If it retains control too long, other interrupts may be missed.

```
/****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios real time clock service routine
*
* Version: 1.01
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: attmr.c
*
* Language: MS C 5.1
*
* Functional Description:
*
* Interrupt 0x08
* This routine services the real time clock interrupt.
*
* Arguments:
*
* Return:
*
* Version History:
* 1.01
*  Replaced peeks and pokes with casts
*
****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

void interrupt cdecl far timer_int(interrupt_registers);

/* G L O B A L S */

extern _timer_int;

/* C O N S T A N T S */

/* D E F I N E S */

/* P R O G R A M */

unsigned timer_setup()
{
 link_interrupt(0x08,&_timer_int); return(ok);
}

//=============================================

void interrupt cdecl far timer_int(interrupt_registers)
{
 /* increment timer count check for 24 hour value */

 if(++TIMER_LONG >= TIMER_LONG_MAX) { TIMER_LONG = 0; TIMER_OFL = 1; }

 if(--MOTOR_COUNT == 0)
 {
  MOTOR_STATUS &= 0xf0;  /* turn off motor running bits */
  outportb(0x3f2,0x0c); /* turn off any FD motors */
 }
 timer_chain();
 eoi_sequence();
}
```

FIFTEEN

The Time of Day Function

The Time of Day Function is used to read and set the timer counter, which is incremented by the timer interrupt.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios time of day function
*
* Version: 1.03
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: attod.c
*
* Language: MS C 5.1
*
* Functional Description:
*
* Interrupt 0x1A - Time of Day function call -
*       read/set the clock
*
*
* Interrupt 1AH - Time of Day function call
*  read/set the internal clock or the 6818 clock chip
*
* AH = 0 - read the clock.
*  returns:
*   CX = high word of count
*   DX = low word of count
*   AL = 0 if no 24 hour rollover since last read
*
* AH = 1 - set the clock
*  CX = high word of count
*  DX = low word of count
*
* AH = 2 - read the time from the 6818 clock
*  returns:
*   CH = BCD hours
*   CL = BCD minutes
*   DH = BCD seconds
*   CY flag if 6818 not running
*
* AH = 3 - set the time to the 6818 clock
*  CH = BCD hours
*  CL = BCD minutes
*  DH = BCD seconds
*  DL = 0/1 = normal/daylight time
*
* AH = 4 - read the date from the 6818 clock
*  returns:
*   CH = BCD century
*   CL = BCD year
*   DH = BCD month
*   DL = BCD day
*   CY flag if 6818 not running
*
* AH = 5 - set the date to the 6818 clock
*  CH = BCD century
*  CL = BCD year
*  DH = BCD month
*  DL = BCD day
*
* AH = 6 - set the 6818 alarm
*  CH = BCD hours from set time
*  CL = BCD minutes from set time
*  DH = BCD seconds from set time
*  returns:
*   CY flag if alarm already set
*  NOTE: function 6 is hooked through Int. 4ah, user replaceable.
*
*
* AH = 7 - reset the 6818 alarm off
```

```
*
* Version History
* 1.01
*  Added enable(); coming into ISR. Fixed Procomm Exiting hangup problem
* 1.02
*  Changed outcmos(32... to outcmos(0x32... for setting century byte
* 1.03
*  Replaced peeks and pokes with casts
*
**************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

unsigned time_of_day_setup();
void interrupt cdecl far time_of_day(interrupt_registers);
unsigned update_in_progress(void);
unsigned initialize_status(void);

/* G L O B A L S */

extern _time_of_day;

/* L O C A L   D E F I N I T I O N S */

#define READ_CURRENT_CLOCK 0x00
#define SET_CURRENT_CLOCK 0x01
#define READ_RTC 0x02
#define SET_RTC 0x03
#define READ_DATE 0x04
#define SET_DATE 0x05
#define SET_ALARM 0x06
#define RESET_ALARM 0x07

/* P R O G R A M */

unsigned time_of_day_setup()
{
 link_interrupt(0x1a,&_time_of_day);
 return(ok);
}

void interrupt cdecl far time_of_day(interrupt_registers)
{
 enable();
 setds_system_segment();
 snapshot_in(TOD,&es);
 switch(ax >> 8)
 {
  case READ_CURRENT_CLOCK:
  watch_string(TOD,"Read Current Clock");
  ax = (ax & 0xff00) | TIMER_OFL;
  dx = TIMER_LOW;
  cx = TIMER_HIGH;
  TIMER_OFL = 0;
  break;

  case SET_CURRENT_CLOCK:
  watch_string(TOD,"Set Current Clock");
  TIMER_LOW = dx;
  TIMER_HIGH = cx;
  TIMER_OFL = 0;
  break;

  case READ_RTC:
  watch_string(TOD,"Read Time");
  flags &= ~carry_bit;
  if (update_in_progress() == error)
  {
   flags |= carry_bit;
  }
  else
  {
   dx = incmos(0);
   dx = dx << 8;
   cx = incmos(4);
   cx = (cx << 8) | incmos(2);
  }
  break;

  case SET_RTC:
  watch_string(TOD,"Set Time");
  if (update_in_progress() == error)
  {
   initialize_status();
  }
  outcmos(0x00,dx >> 8);
  outcmos(0x02,cx);
  outcmos(0x04,cx >> 8);
  outcmos(0x0b,(incmos(0x0b) & 0x30) | (dx & 0x0001) | 2);
  break;

  case READ_DATE:
  watch_string(TOD,"Read Date");
  flags &= ~carry_bit;
  if (update_in_progress() == error)
  {
   flags |= carry_bit;
  }
  else
  {
   cx = incmos(0x32);
   cx = (cx << 8) | incmos(9);
   dx = incmos(8);
   dx = (dx << 8) | incmos(7);
  }
  break;

  case SET_DATE:
  watch_string(TOD,"Set Date");
  if (update_in_progress() == error)
  {
   initialize_status();
  }
  outcmos(6,0);
  outcmos(7,dx);
  outcmos(8,dx >> 8);
  outcmos(9,cx);
  outcmos(0x32,cx >> 8);
  outcmos(0x0b,(incmos(0x0b) & 0x7f));
  break;

  case SET_ALARM:
  watch_string(TOD,"Set Alarm");
  flags &= ~carry_bit;
  if((incmos(0x0b) & 0x20) != 0)
  {
   ax = 0;
   flags |= carry_bit;
  }
  else
  {
   if(update_in_progress() == error)
   {
    initialize_status();
   }
   outcmos(1,dx >> 8);
   outcmos(3,cx);
   outcmos(5,cx >> 8);
   outportb(0xa1,inportb(0xa1) | 0xfe);
   outcmos(0x0b,(incmos(0x0b) & 0x7f) | 0x20);
  }
  break;

  case RESET_ALARM:
  watch_string(TOD,"Reset Alarm");
  outcmos(0x0b,incmos(0x0b) & 0x57);
  break;

  default:
  watch_string(TOD,"Bad Command");
  flags |= carry_bit;
  break;

 }
 snapshot_out(TOD,&es);
}


//-----------------------------------------------

unsigned update_in_progress(void)
{
 unsigned j;
 for(j = 0; j < 600; j++)
 {
  if ((incmos(0x0a) & 0x80) == 0) return(ok);
 }
 return(error);
```

```
}

unsigned initialize_status(void)
{
 outcmos(0x0a,26);
 outcmos(0x0b,82);
 incmos(0x0c);
 incmos(0x0d);
}
```

SIXTEEN

The Print Screen Function

The Print Screen Function is used to print the screen contents to LPT1.

Some additions and enhancements could be:
* Creating an "active_printer" flag (in system ram) which could be used to direct screen prints to LPT2 or LPT3.
* Providing a screen print function to output on a serial channel.
* Using this module as a guideline for creating a print-screen to disk-file applications program.

There are various public-domain print-screen functions which support video adapters other than MDA and CGA. An EGA print-screen routine for instance, has been circulated on various Bulletin Board Systems. The logic of one of these could be adapted to replace the standard print screen function , if desired.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: AT Bios print screen function
*
* Version: 1.01
*
* Author: FOSCO
*
* Date: 10-20-89
*
* Filename: atprsc.c
*
* Language: MSC 5.1
*
* Functional Description:
*
* Interrupt 0x05
* This module copies the contents of the screen buffer to the printer.
*
* Arguments:
* None
*
* Return:
* None
*
* Version History:
* 1.01
*  replaced peeks and pokes with casts
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

void interrupt cdecl far print_screen(interrupt_registers);
void print_screen_service(struct prscregs *);

/* G L O B A L S */

extern _print_screen;

/* P R O G R A M */

void print_screen_setup()
{
 link_interrupt(0x05,&_print_screen);
}

variables
```

```
unsigned columns_per_line,old_cursor,current_cursor;
unsigned char row,col,current_char;
end_variables prscregs;

void interrupt cdecl far print_screen(interrupt_registers)
{
 prscregs *myblock;
 unsigned print_error = false;
 myblock = acquire_block(PRSC);
 if (PRSC_BUSY != T) /* check status byte */
 {
  PRSC_BUSY = 1;  /* set status = busy */
  myblock->ax = 0x0f00;
  sys_int(0x10,myblock);
  /* get current screen mode */
  myblock->columns_per_line = myblock->ax >> 8;

  myblock->ax = 0x000d;
  myblock->dx = 0;
  sys_int(0x17,myblock);
  myblock->ax = 0x000a;
  myblock->dx = 0;
  sys_int(0x17,myblock);

  myblock->ax = 0x0300;
  sys_int(0x10,myblock);
  /* read and save old cursor position */
  myblock->old_cursor = myblock->dx;
  myblock->row = myblock->col = 0;

  for (myblock->row = 0;
        (myblock->row < 25) & !print_error;
                  myblock->row++)
  {
   for (myblock->col = 0;
        (myblock->col < myblock->columns_per_line) & !print_error;
                  myblock->col++)
   {
        // set cursor position
    myblock->ax = 0x0200;
    myblock->dx = (myblock->row <<8) | myblock->col;
    sys_int(0x10,myblock);

        // read character at cursor
    myblock->ax = 0x0800;
    myblock->bx = 0;
    sys_int(0x10,myblock);

        // print character
    myblock->current_char = myblock->ax;
    if (myblock->current_char==0) myblock->current_char = 0x20;
    myblock->dx = 0;
    myblock->ax = myblock->current_char & 0x00ff;
    sys_int(0x17,myblock);
    if((myblock->ax & 0x0100) != 0) print_error = true;
   }
   // do the cr-lf at end of columns
   myblock->ax = 0x000d;
   myblock->dx = 0;
   sys_int(0x17,myblock);
   myblock->ax = 0x000a;
   myblock->dx = 0;
   sys_int(0x17,myblock);
   if((myblock->ax & 0x0100) != 0) print_error = true;
  }
  myblock->ax = 0x0200;  // reset cursor to original position
  myblock->dx = myblock->old_cursor;
  sys_int(0x10,myblock);
  PRSC_BUSY = 0; /* clear status = not busy */
 }
}
```

SEVENTEEN

The Vue File

The Vue.c file is the source code for SysVue, the resident monitor/debugger program. SysVue allows the user to gain access to the system features from any program. If the bootstrap operation is unsuccessful, program control is automatically transferred to SysVue.

SysVue has some Watch features built-in to assist in development/debugging. The "select" command is used to set and clear specific bits in the watch_selection variable. These bits are checked when "watch" is "on" and display commands imbedded in the various function routines are enabled. These display commands are:

```
"watch_string(device,"string");
"watch_word(device,word_value);
"watch_byte(device,byte_value);
```

These watch commands check to see if the corresponding "device" bit is set in the watch_flag variable. If so, the command is executed. When developing and debugging code, this feature is extremely handy. For a final version, one may wish to delete the "watch_xxxx()" calls to save space and increase speed.

The SysVue "SI" command is used to display pertinent data about the Bios and the system. It is especially handy for tech support communications, allowing the user to use the "SI" command to display and convey commonly requested information to the technically oriented person. Since it also shows configuration information, it permits a quick peek at the current system configuration.

The system Setup is accomplished by the use of the:
"Time" and "Date" commands to set the clock\calendar
"Drive" to set the number and type of floppy and hard drives
"Video" to set the type of video adapter
"Mem" to set the size of system and extended memory
"Npx" to declare the presence of the numeric processor
"Types" to display the hard disk types in the resident type table
"Cmos" to display the checksum value of the CMOS RAM

The redirection command ">",">con",">lpt" are used to copy the screen output to a printing device. This allows generating audit trails when involved in debugging, and with "watch" enabled, can provide a wealth of data concerning internal Bios operation. Since one can re-compile the Bios with additional "watch_xxx()" commands, debugging aids can easily be placed where desired.

The "Trace" and "e" (for execute) commands allow a detailed single-step trail to be generated when required. Once a problem area is located with "watch" features, this single stepping can provide lowest level tracing of system action. Single stepping does NOT sense when the stack-segment or stack-pointers are changed, so unpredictable results may occur when these registers are changed.

The "Int" command allows the execution of an interrupt, using the register values set with the "Rxx" commands. This permits specialized execution of bios functions from within sysvue. With the watch function enabled, this is a handy debugging aid.

SysVue may also be expanded with new commands as desired.

An easy way for you to generate documentation of your version of SysVue is to use the ">LPT" command to provide hard copy of the various screens and menus. These printouts may then be integrated into the user documentation you create.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988 - All Rights Reserved
*
* Module Name: AT Bios SysVue
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 12-01-88
*
* Filename: atvue.c
*
* Functional Description:
*
*  This is the SysVue module which is a Bios-resident debug/monitor program.
*
*  NOTE:
*   Entry to SysVue is by CTRL-ALT-BREAK.
*
* Arguments:
*
* Return:
*
* Version History:
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include "atkit.h"

/* F U N C T I O N   P R O T O T Y P E S */

unsigned sysvue_setup();
void interrupt cdecl far sysvue(void);
void interrupt cdecl far sysbreak(void);
char ci(void);
void co(char);
unsigned ishex(char);
unsigned islower(char);
unsigned isalpha(char);
unsigned isdelim(char);
void lword(unsigned);
void pword(unsigned);
void lbyte(char);
void pbyte(char);
char get_byte(void);
unsigned get_word(void);
void poked(unsigned,unsigned long int);
void set_system_segment(void);
void syntax_error(void);
void write_string(unsigned);
unsigned get_token(void);
unsigned get_token_number(unsigned *);
void find_next_list_item(void);
void get_line_input(char); /* arg is line terminator char */
unsigned get_token_value();
unsigned long get_limits();
void set_vector(char,unsigned,unsigned);
unsigned long get_vector(char);
unsigned console_break(void);
void far_call(unsigned,unsigned);
unsigned value_present(char);
void display_regs(unsigned);
void display_flags(unsigned);

/* G L O B A L   V A R I A B L E S */
```

```
/*------ external data (variables) declarations ---------*/

extern reset;
extern unsigned comm_list[];
extern unsigned lpt_list[];

/* variables are in system scratch segment of RAM */

extern char sysvue_busy; /* 0/1 = not busy/busy */
extern unsigned long break_chain;
extern unsigned long trace_chain;
extern unsigned long trap_chain;
extern char mode_type,mode_size,sum,char_count;
extern char exit_flag;
extern char buffer[80];
extern unsigned buffer_index,token_index;
extern list_index,token_number;
extern unsigned command_index;
extern unsigned token_value[8];
extern unsigned cc;
extern char present;
extern unsigned last_dump_start_seg;
extern unsigned last_dump_start_off;
extern unsigned enter_seg,enter_off;
extern char inchar;
extern unsigned int_seg,int_off;
extern unsigned reg_index;
extern unsigned reg_saves[15];
extern unsigned inreg_saves[15];
extern unsigned outreg_saves[15];
extern unsigned trace_count;
extern char break_flag;
extern char token_buffer[];
extern char last_delim;
extern char record_type;
extern unsigned watch_flag;
extern unsigned watch_selection;
extern unsigned u_found;
extern unsigned jj;
extern unsigned redirect_flag;
extern char dec_array[6];

/* G L O B A L   C O N S T A N T S */

/*------ external data (constants) declarations ---------*/
/* constants are in Bios ROM */

extern bios_id;
extern ver_id;
extern bios_name;
extern owner_id;
extern date_stamp;

/* L O C A L   D E F I N I T I O N S */

#define byte 1
#define word 2
#define dword 4

#define ascii 1
#define hex 2

#define data_record 0
#define end_of_file_record 1
#define extended_address_record 2
#define start_address_record 3

/* this are the delimiters for biosvue command strings */

#define space 32
#define comma 44
#define colon 58
#define semicolon 59
#define leftparen 40
#define rightparen 41
#define period 46
#define ques 63
#define tab 9
#define quote 34
#define slash 47

/*------------ enumeration lists ------------*/

/* This list MUST be in the same order as the command token list.
** The enumerated values for the commands are assigned by this list.
** These values are used by the case operator in the main body of SysVue.
** For each entry in this list, there must be a corresponding entry in the Command Token List.
** The token list is searched and the number of the find is used as the command switch value.
*/

enum command
{
 command_NUL,  /* Nul value = no find */
 command_C,  /* compare */
 command_CA,  /* compare ascii */
 command_CB,  /* compare byte */
 command_CW,  /* compare word */
 command_CD,  /* compare double */
 command_CHK,  /* checksum */
 command_CLS,  /* clear screen */
 command_COLD_BOOT, /* cold boot */
 command_CMOS, /* display cmos contents and checksum */
 command_DATE, /* set date */
 command_D,  /* dump */
 command_DA,  /* dump ascii */
 command_DB,  /* dump byte */
 command_DW,  /* dump word */
 command_DD,  /* dump double word */
 command_DRIVE, /* set number and type of drives */
 command_E,  /* enter or execute */
 command_EA,  /* enter ascii */
 command_EB,  /* enter byte */
 command_EW,  /* enter word */
 command_ED,  /* enter double */
 command_EXIT,  /* exit from SysVue */
 command_F,  /* fill */
 command_FA,  /* fill ascii */
 command_FB,  /* fill byte */
 command_FW,  /* fill word */
 command_FD,  /* fill double */
 command_H,  /* hex add and subtract */
 command_HARD_BOOT, /* hard boot */
 command_HELP,
 command_I,  /* input port */
 command_IB,  /* input byte */
 command_IW,  /* input word */
 command_INT,  /* interrupt */
 command_M, /* move */
 command_MEM, /* set/display mem sizes */
 command_NPX, /* select NPX present or not */
 command_O,  /* output port */
 command_OB,  /* output byte */
 command_OW,  /* output word */
 command_QUIT,  /* exit from SysVue */
 command_R,  /* display registers */
 command_RAX,  /* register ax */
 command_RBX,  /* register bx */
 command_RCX,  /* register cx */
 command_RDX,  /* register dx */
 command_RSP,  /* register sp */
 command_RBP,  /* register bp */
 command_RSI,  /* register si */
 command_RDI,  /* register di */
 command_RDS,  /* register ds */
 command_RES,  /* register es */
 command_RSS,  /* register ss */
 command_RCS,  /* register cs */
 command_RIP,  /* register ip */
 command_RF,  /* register flags */
 command_RHEX,  /* read Intel hex */
 command_SI,  /* display info */
 command_SELECT, /* select watch parameters */
 command_S,  /* search */
 command_SA,  /* search ascii */
 command_SB,  /* search byte */
 command_SW,  /* search word */
 command_SD,  /* search double */
 command_TIME, /* set time */
 command_T, /* single step */
 command_TRACE, /* single step */
 command_TYPES, /* list hard drive types on table */
 command_VIDEO, /* set/display video adapters */
 command_WATCH, /* watch bios */
 command_WARM_BOOT, /* warm boot */
 command_WHEX,  /* write Intel hex */
 command_TO_LPT, /* direct output to lpt */
 command_TO_CON,  /* direct output to console */
 command_BACK_TO_CON,  /* default back to console */
};

/*  this enumeration list is used for the register
** display commands.
```

```
*/

enum order
{
 nulreg,
 axsave,bxsave,cxsave,dxsave,spsave,
 bpsave,sisave,disave,dssave,essave,
 sssave,cssave,ipsave,pswsave,
};

/* L O C A L   C O N S T A N T S */

// this is a list of the command tokens for biosvue, it MUST agree with the enumeration list

const command_token_list[]=
{
 "C",  /* compare */
 "CA",  /* compare ascii */
 "CB",  /* compare byte */
 "CW",  /* compare word */
 "CD",  /* compare double */
 "CHK",  /* checksum */
 "CLS", /* clear screen */
 "COLD", /* cold boot */
 "CMOS", /* display cmos contnets and checksum */
 "DATE", /* set date */
 "D",  /* dump */
 "DA",  /* dump ascii */
 "DB",  /* dump byte */
 "DW",  /* dump word */
 "DD",  /* dump double */
 "DRIVE", /* display or set drives */
 "E",  /* enter */
 "EA",  /* enter ascii */
 "EB",  /* enter byte */
 "EW",  /* enter word */
 "ED",  /* enter double */
 "EXIT", /* exit from SysVue */
 "F",  /* fill */
 "FA",  /* fill ascii */
 "FB",  /* fill byte */
 "FW",  /* fill word */
 "FD",  /* fill double */
 "H", /* hex add and subtract */
 "HARD", /* hard boot */
 "HELP", /* display help */
 "I",  /* input port */
 "IB",  /* input byte */
 "IW",  /* input word */
 "INT",  /* interrupt */
 "M",  /* move */
 "MEM", /* set/display meory sizes */
 "NPX", /* select NPX present or not */
 "O",  /* output port */
 "OB",  /* output byte */
 "OW",  /* output word */
 "QUIT", /* exit form SysVue */
 "R",  /* display registers */
 "RAX",  /* register ax */
 "RBX",  /* register bx */
 "RCX",  /* register cx */
 "RDX",  /* register dx */
 "RSP",  /* register sp */
 "RBP",  /* register bp */
 "RSI",  /* register si */
 "RDI",  /* register di */
 "RDS",  /* register ds */
 "RES",  /* register es */
 "RSS",  /* register ss */
 "RCS",  /* register cs */
 "RIP",  /* register ip */
 "RF",  /* register flags */
 "RHEX", /* read Intel hex */
 "SI",  /* display id */
 "SELECT", /* select watch parameters */
 "S",  /* search */
 "SA",  /* search ascii */
 "SB",  /* search byte */
 "SW",  /* search word */
 "SD",  /* search double */
 "TIME", /* set time */
 "T",
 "TRACE",
 "TYPES", /* list hard drive types on table */
 "VIDEO", /* set/display video adpaters */
 "WATCH", /* watch bios */
 "WARM", /* warm boot */
 "WHEX", /* write Intel hex */
 ">LPT",
 ">CON",
 ">",
 8-1,  /* list terminator */
};

// the register token list is searched to determine the switch value for the register commands.

const register_token_list[]=
{
 "AX","BX","CX","DX","SP","BP","SI","DI","DS","ES","SS","CS","IP",-1,
};

// the flag token list is used for the flag register command

const flag_token_list[]=
{
 "NC","CY","--","--","PO","PE","--","--","NA","AC","--","--","NZ","ZR","--","--",
 "PL","NG","DI","EI","UP","DN","NV","OV","--","--","--","--","--","--","--","--",-1,
};

// the flag display token list is used for the flag register command

const flag_display_token_list[16][2]=
{
 "NC","CY","--","--","PO","PE","--","--","NA","AC","--","--","NZ","ZR","--","--",
 "PL","NG","DI","EI","UP","DN","NV","OV","--","--","--","--","--","--","--","--",
};

// This is a list of the delimiter characters to be accepted when parsing command lines.
// A zero value marks the  end of the list.

const char delim_list[]=
{
 space,comma,colon,semicolon,leftparen,rightparen,period,ques,tab,quote,slash,0
};


/* P R O G R A M */

/*========= SysVue Setup Routine ==========*/

// This routine is called by the POD to effect the installation of SysVue by patching Interrupt 18.

unsigned sysvue_setup()
{
 link_interrupt(0x18,sysvue);
 return(ok);
}

/*=============================================*/
/*=============================================*/
/*======= SysVue Main Routine =================*/
/*=============================================*/
/*=============================================*/

variables
end_variables vueregs;

// these are arbitrary identifiers so the main routine will know how it was entered

#define trap_origin 5
#define trace_origin 6
#define boot_origin 7
#define divide_origin 8
#define syserr_origin 9

void sysmain();

void interrupt cdecl far Divide(interrupt_registers)
{
 sysmain(divide_origin);
}

void interrupt cdecl far SysError(interrupt_registers)
{
 sysmain(syserr_origin);
}


void interrupt cdecl far Sysvue(interrupt_registers)
{
 sysmain(boot_origin);
}
```

```
void interrupt cdecl far Systrap(interrupt_registers)
{
 sysmain(trap_origin);
}

void interrupt cdecl far Systrace(interrupt_registers)
{
 sysmain(trace_origin);
}


void sysmain(unsigned origin,interrupt_registers)
{
 vueregs *myblock;
 enable();
 myblock = acquire_block(VUE);
 /* variables will be referenced in sys seg */

 exit_flag = 0;

 if((origin == boot_origin) && (sysvue_busy != 0))
 {
  write_string("\n\rSysVue Busy");
 }
 else
 {
  sysvue_busy=1;


  // chain the break vector through sysvue  logic: if the break flag is set by the sysbreak isr,
  // return a true from the csts routine to abort the op in progress. when exiting sysvue:
  // de-chain the vector back to the original state for normal operation.

  break_chain = get_vector(0x1b);
  link_interrupt(0x1b,sysbreak);

  // move the stacked registers into the saves so we know where we came from
  reg_saves[axsave] = ax;
  reg_saves[bxsave] = bx;
  reg_saves[cxsave] = cx;
  reg_saves[dxsave] = dx;
  reg_saves[sisave] = si;
  reg_saves[disave] = di;
  reg_saves[essave] = es;
  reg_saves[dssave] = ds;
  reg_saves[spsave] = sp;
  reg_saves[sssave] = get_ss();
  reg_saves[pswsave] = flags;
  reg_saves[ipsave] = ip;
  reg_saves[cssave] = cs;


  if(origin == trace_origin)  // de-chain trace vector
  {
   set_vector(0x1,trace_chain >> 16,trace_chain);

   // if(peek(0x26a,0x316)  == 0x0beb)
   //{
    // place for option of silent trace until condition
    // trace_count = 1;

    display_regs(&reg_saves);
    write_string("\n\r          ");
    lword(cs);write_string(":");lword(ip);write_string(" ");
    lbyte(peekb(cs,ip));
    write_string(" ");
    lword(peek(cs,ip+1));

   //}
   if(--trace_count != 0)
   {
    if(console_break())
    {
     exit_flag=0;
     trace_count=1;
    }
    else
    {
     // check for any key
     if (kbhit())
     {
      exit_flag=0;
      trace_count=1;
     }
     else
    {
     exit_flag=1;
     reg_saves[pswsave] |= 0x0100; // set trace bit
     // chain-in the trace vector
     trace_chain = get_vector(0x1);
     link_interrupt(0x1,Systrace);
    }
   }
  }
}

if(origin == trap_origin)  // de-chain trap vector
{
 // restore the trapped locations and decrement the pc value
 // clear the trap saves and stuff
}

if(origin == boot_origin)
{
 trace_count = 1;
 // restore the trapped locations and decrement the
 // pc value
 // clear the trap saves and stuff
}

mode_size = byte; /* default to byte presentation mode */
mode_type = hex; /* default to hex (not ascii) */
if(origin == boot_origin)
{
 write_string("\n\rSysVue Copyright (c) FOSCO 1988, 1989 - All Rights Reserved.");
}

if(origin == divide_origin)
{
 write_string("\n\rDivide Overflow at location - ");
 lword(cs); write_string(":");lword(ip);
}

if(origin == syserr_origin)
{
 write_string("\n\rSystem Error, Type - "); lword(ax);
}


while (exit_flag ==0)
{

 write_string("\n\rSysVue>");
 get_line_input(cr);

 /* move past leading spaces */
 while(buffer[buffer_index] == ' ') buffer_index++;

 if(get_token() == true) /* if there is a command */
 {
  /* get the token number for the
  ** command from the token list
  */
  command_index = get_token_number(&command_token_list);

  /* get the rest of the arguments on the command line */
  get_values();

  /* execute the command */
  switch (command_index)
  {
   default: /* command not found */
   syntax_error();
   break;

   /*============ COMPARE COMMAND =======*/

   case command_CA:
   mode_type = ascii; mode_size = byte; goto compare_case;

   case command_CB:
   mode_type = hex; mode_size = byte; goto compare_case;

   case command_CW:
   mode_type = hex; mode_size = word; goto compare_case;

   case command_CD:
   mode_type = hex; mode_size = dword; goto compare_case;

   case command_C:
```

```
compare_case:
do
{
 if (peekb(token_value[0],token_value[1]) != peekb(token_value[4],token_value[5]))
 {
  co(13); co(10);
  lword(token_value[0]); co(':');
  lword(token_value[1]);
  co(' '); co('>'); co(' ');
  lbyte(peekb(token_value[0],token_value[1]));
  co(' '); co(' ');
  lbyte(peekb(token_value[4],token_value[5]));
  co(' '); co('<'); co(' ');
  lword(token_value[4]); co(':');
  lword(token_value[5]);
 }
 if(console_break()) break;
 token_value[1]++; token_value[5]++;
}
while ((token_value[1] < token_value[3]) &&
(token_value[1] != 0));

break;

/*======= Clear Screen Command ===========*/

case command_CLS:
// read current video mode
myblock->ax = 0x0f00;
sys_int(0x10,myblock);
// reset video mode (clears screen)
myblock->ax = myblock->ax & 0x00ff;
sys_int(0x10,myblock);
break;

/*=========== CMOS Command ================*/

case command_CMOS:

// this would be a great place to have a dump of the CMOS contents.

write_string("Calculated CMOS checksum: ");
token_value[1]=0;
for(token_value[0] = 0x10; token_value[0] < 0x2e;
token_value[0]++)
{
 token_value[1] += incmos(token_value[0]);
}
lword(token_value[1]);

write_string("\n\r    Actual CMOS checksum: ");
lword((incmos(0x2e)<< 8) | incmos(0x2f));

break;

/*============ D Commands =================*/

case command_DA:
mode_size = byte; mode_type = ascii; goto dump_case;

case command_DB:
mode_size = byte; mode_type = hex; goto dump_case;

case command_DW:
mode_size = word; mode_type = hex; goto dump_case;

case command_DD:
mode_size = dword; mode_type = hex; goto dump_case;

case command_D:
dump_case:

cc = 256; /* default dump length in characters */
if (mode_type == ascii) cc = 1024;

if ((present & 0x80) == 0)
{
 token_value[1] = last_dump_start_off + cc;
 token_value[0] = last_dump_start_seg;
}

if (token_value[3] == 0)
{
 token_value[3] = token_value[1] + cc;
}
last_dump_start_seg = token_value[0];
```

```
last_dump_start_off = token_value[1];

do
{
 co(cr); co(lf); lword(token_value[0]);
 co(':');lword(token_value[1]); co(' ');

 if (mode_type == ascii)
 {
  do
  {
   cc = peekb(token_value[0],token_value[1]++) & 0x7f;
   if ((cc < 0x20) || (cc > 0x7e)) cc = '.';
   co(cc);
  }
  while ((token_value[1] % 64) != 0);
 }

 if ((mode_type == hex) || (mode_type == 0))
 {
  if ((mode_size == byte) || (mode_size == 0))
  {
   do
   {
    co(' ');
    lbyte(peekb(token_value[0],token_value[1]++));
    /* print extra space every four characters */
    if ((token_value[1] % 4) == 0) co(' ');
   }
   while ((token_value[1] % 16) != 0);
  }

  if (mode_size == word)
  {
   do
   {
    co(' ');
    lword(peek(token_value[0],token_value[1]));
    token_value[1] += 2;
    if (((token_value[1] & 0xfffe) % 4) == 0) co(' ');
   }
   while (((token_value[1] & 0xfffe) % 16) != 0);
  }

  if (mode_size == dword)
  {
   do
   {
    co(' ');
    lword(peek(token_value[0],token_value[1]+2));
    co(':');
    lword(peek(token_value[0],token_value[1]));
    token_value[1] += 4;
    if (((token_value[1] & 0xfffc) % 4) == 0) co(' ');
   }
   while (((token_value[1] & 0xfffc) % 16) != 0);
  }

  /* print the ascii char for the hex dump */
  co(' ');
  token_value[1] -= 16;
  do
  {
   cc = peekb(token_value[0],token_value[1]++) & 0x7f;
   if ((cc < 0x20) || (cc > 0x7e)) cc = '.';
   co(cc);
  }
  while ((token_value[1] % 16) != 0);
 }
 if (console_break()) break;
}
while ((token_value[1] < token_value[3]) && (token_value[1] != 0));
break;

/*=========== Exit Commands ============*/

case command_EXIT:
case command_QUIT:
exit_flag=1;
break;

/*=========== Enter Commands ============*/

case command_EA:
mode_size = byte; mode_type = ascii; goto enter_case;
```

```
case command_EB:
mode_size = byte; mode_type = hex; goto enter_case;

case command_EW:
mode_size = word; mode_type = hex; goto enter_case;

case command_ED:
mode_size = dword; mode_type = hex; goto enter_case;

case command_E:

// if e<enter> then it is execute else it is enter


if (value_present(1) == 0)  // e<enter> is execute
{
 exit_flag=1;
 trace_count = -1;
 reg_saves[pswsave] |= 0x0100; // set trace bit
 // chain-in the trace vector
 trace_chain = get_vector(0x1);
 link_interrupt(0x1,Systrace);
 break;

}

enter_case:

/* if there is a command line input, just store it */
if(value_present(3) == true)
{
 if(mode_size == byte) pokeb(token_value[0],token_value[1],token_value[3]);
 if(mode_size == word) poke(token_value[0],token_value[1],token_value[3]);
 if(mode_size == dword)
{
  poke(token_value[0],token_value[1],token_value[3]);
  poke(token_value[0],token_value[1]+2,token_value[2]);
 }
}
else
{
 lword(token_value[0]); co(':'); lword(token_value[1]);

 /* solicit input */

 enter_seg = token_value[0];
 enter_off = token_value[1];

 inchar = ' ';
 while (inchar == ' ')
 {
  co(' ');
  if(mode_size==byte) lbyte(peekb(enter_seg,enter_off));
  if(mode_size==word) lword(peek(enter_seg,enter_off));
  if(mode_size==dword)
  {
   lword(peek(enter_seg,enter_off+2));
   co(':');
   lword(peek(enter_seg,enter_off));
  }
  co('-');
  get_line_input(' ');
  /* terminate this on a space or a cr */
  get_values();
  if(value_present(1))
  {
   if(mode_size==byte) pokeb(enter_seg,enter_off,token_value[1]);
   if(mode_size==word) poke(enter_seg,enter_off,token_value[1]);
   if(mode_size==dword)
   {
    poke(enter_seg,enter_off,token_value[1]);
    poke(enter_seg,enter_off+2,token_value[0]);
   }
  }
  enter_off = enter_off + mode_size;
  if ((enter_off % 8) == 0)
  {
   co(cr); co(lf);
   lword(enter_seg); co(':'); lword(enter_off);
  }
 }
}
break;

/*========== Fill Commands ============*/
```

```
case command_FB:
mode_size = byte; mode_type = hex; goto fill_case;

case command_FW:
mode_size = word; mode_type = hex; goto fill_case;

case command_FD:
mode_size = dword; mode_type = hex; goto fill_case;

case command_F:
fill_case:
do
{
 if(mode_size == byte) pokeb(token_value[0],token_value[1],token_value[5]);
 if(mode_size == word) poke(token_value[0],token_value[1],token_value[5]);
 if(mode_size == dword)
 {
  poke(token_value[0],token_value[1],token_value[5]);
  poke(token_value[0],token_value[1]+2,token_value[4]);
 }
 if(console_break()) break;
 token_value[1] += mode_size;
}
while ((token_value[1] < token_value[3]) &&
(token_value[T] != 0));
break;

/*======= Int Command =================*/

// This command invokes the interrupt selected. It uses the sys_int_registers and the
// sys_int command. The user has to confirm the command with an X to eXecute it.

// this command should use a separate set of registers ????

case command_INT:

/* get the interrupt number from the command line */

/* print the vector value */

if (token_value[1] < 256)
{
 write_string("Interrupt is at ");

 int_off = peek(0x00,token_value[1] * 4);
 int_seg = peek(0x00,(token_value[1] * 4) +2);

 lword(int_seg); co(':'); lword(int_off); co(' ');

 write_string("Press X to confirm eXecution");

 /* wait for confirmation */
 cc = ci();

 if ((cc == 'X') || (cc == 'x'))
 {
  write_string("\n\r"); // do a cr lf for a clean line
  myblock ->ax = reg_saves[axsave];
  myblock ->bx = reg_saves[bxsave];
  myblock ->cx = reg_saves[cxsave];
  myblock ->dx = reg_saves[dxsave];
  myblock ->si = reg_saves[sisave];
  myblock ->di = reg_saves[disave];
  myblock ->ds = reg_saves[dssave];
  myblock ->es = reg_saves[essave];

  sys_int(token_value[1],myblock);

  outreg_saves[axsave] = myblock ->ax;
  outreg_saves[bxsave] = myblock ->bx;
  outreg_saves[cxsave] = myblock ->cx;
  outreg_saves[dxsave] = myblock ->dx;
  outreg_saves[sisave] = myblock ->si;
  outreg_saves[disave] = myblock ->di;
  outreg_saves[essave] = myblock ->es;
  outreg_saves[dssave] = myblock ->ds;
  outreg_saves[pswsave] = myblock ->flags;

  /* display the register upon return */
  display_regs(&outreg_saves);
 }
}
else write_string("Value Out of Range ");

break;
```

```
/*============= HELP =============*/

case command_HELP:
write_string("C <range> <address> - compare\n\r");
write_string("CLS - Clear Screen\n\r");
write_string("CHK <range> - checksum\n\r");
write_string("COLD - cold boot to cuurent Bios\n\r");
write_string("CMOS - display the CMOS checksum\n\r");
write_string("D[type][<range>] - dump memory\n\r");
write_string("DRIVE [0-3 | A:-F:][type] - display/set drives\n\r");
write_string("DATE [xx/xx/xxxx] - Display or Set Date\n\r");
write_string("E - execute single step until any key hit\n\r");
write_string("E[type] <address> [<list>] - enter\n\r");
write_string("EXIT - exit from SysVue\n\r");
write_string("F[type] <range> <list> - fill\n\r");
write_string("H <value> <value> - hex add/sub\n\r");
write_string("HARD - re-boot to primary Bios\n\r");
write_string("HELP - help screen\n\r");
write_string("I[type] <address> - In port\n\r");
write_string("INT <value> - Interrupt\n\r");
write_string("M <range> <address> - move\n\r");
write_string("MEM [sys,ext] - display/set memory sizes\n\r");
pause();
write_string("NPX [0|1] - No NPX/Yes NPX\n\r");
write_string("O[type] <address> <value> - Out port\n\r");
write_string("QUIT - exit from SysVue\n\r");
write_string("R[<reg>] <value> - register\n\r");
write_string("RHEX <range> - read Intel hex\n\r");
write_string("S[type] <range> <key> <mask> - search\n\r");
write_string("SELECT - Select Watch Parameters\n\r");
write_string("SI - Display System Information\n\r");
write_string("TIME [xx:xx:xx] - Display or Set Time\n\r");
write_string("TRACE [count] - Trace count steps or until anykey hit\n\r");
write_string("TYPES - Display hard disk type parameters\n\r");
write_string("VIDEO [type] - display/set video type\n\r");
write_string("WARM - warm boot to current Bios\n\r");
write_string("WATCH - Toggle watch on/off\n\r");
write_string("WHEX <range> - write Intel hex\n\r");
write_string(">[CON | LPT] - redirect console output\n\r");

break;

/*========== Calculator Command =========*/

case command_H:
write_string("Sum is: "); lword(token_value[1]+token_value[3]);
write_string("  Difference is: "); lword(token_value[1]-token_value[3]);
break;

/*===== Sys Info Command ===========*/

case command_SI:

write_string("\n\r  Current Time: "); display_time();
write_string("\n\r  Current Date: ");display_date();

write_string("\n\r     Bios Name: "); write_string(&bios_id);

write_string("\n\r  Bios Version: "); write_string(&ver_id);

write_string("\n\r     Bios Date: "); write_string(&date_stamp);

write_string("\n\r     Copyright: "); write_string(&owner_id);

write_string("\n\r     Bios Size: 64k At Segment: "); lword(bios_cs());

write_string("\n\r           CPU: ");
// a cpu type sensing routine could be added here
write_string("80286");
write_string("\n\r           NPX: "); display_npx();

write_string("\n\r     Com Ports: ");
for (cc = 0;
((cc < 4) && (peek40(&comm_list[cc]) != 0));cc++)
{
 lword(peek40(&comm_list[cc])); co(' ');
}

write_string("\n\r     LPT Ports: ");
for (cc = 0;
((cc < 3) && (peek40(&lpt_list[cc]) != 0));cc++)
{
 lword(peek40(&lpt_list[cc])); co(' ');
}

// display the disk drives in the system
```

```
display_drives();

display_video();


write_string("\n\rMemory Size       System        Extended");
write_string("\n\r      Expected:    ");
display_sys_mem();
write_string("        ");
display_ext_mem();
write_string("\n\r        Found:    ");
bin2dec(peek40(0x13) + (sys_seg_size/1024),&dec_array);
co(dec_array[0]);
co(dec_array[1]);
co(dec_array[2]);
co(dec_array[3]);
co(dec_array[4]);

write_string("        ");
bin2dec(cmos_ext_found(),&dec_array);
co(dec_array[0]);
co(dec_array[1]);
co(dec_array[2]);
co(dec_array[3]);
co(dec_array[4]);

#if(0==1)
write_string("\n\rCMOS Values are:");
write_string("\n\rDiagnostic   ");lbyte(incmos(0x0e));
write_string("\n\rRestart      ");lbyte(incmos(0x0f));
write_string("\n\rFloppy Disks ");lbyte(incmos(0x10));
write_string("\n\rHard Disks   ");lbyte(incmos(0x12));
write_string("\n\rEquipment    ");lbyte(incmos(0x14));
#endif

break;

/*------- set time -------------*/

case command_TIME:
if(value_present(0)) // if anything entered
{
 myblock->cx = token_value[0];
 myblock->cx = (myblock->cx << 8) | token_value[1];
 myblock->dx = token_value[3];
 myblock->dx = myblock->dx << 8;
 myblock->ax = 0x0300;
 sys_int(0x1a,myblock);
 if((myblock->flags & carry_bit) != 0)
 write_string("Error setting time");
}
write_string("Current Time is - ");display_time();
break;

/*------- set date -------------*/

case command_DATE:
if(value_present(0)) // if anything entered
{
 myblock->dx = token_value[0];
 myblock->dx = (myblock->dx << 8) | token_value[1];
 myblock->cx = token_value[3];
 myblock->ax = 0x0500;
 sys_int(0x1a,myblock);
 if((myblock->flags & carry_bit) != 0)
 write_string("Error setting date");
}
write_string("Current Date is - ");display_date();
break;

/*------- set drives -----------*/

case command_DRIVE:

// if value - a-f, then set type of drive in system
if((token_value[0] >= 0x0a) && (token_value[0] <= 0x0f))
{
 // a and b are easy, c,d,e,f have to count drives

 // A: is floppy
 if(number_of_floppies() > 0)
 {
  if(token_value[0] == 0x0a)
  {
   outcmos(0x10,(incmos(0x10) & 0x0f) | ((token_value[1] & 0x0007) << 4));
  }
```

```
}
// B: is floppy
if(number_of_floppies() > 1)
{
 if(token_value[0] == 0x0b)
 {
  outcmos(0x10,(incmos(0x10) & 0xf0) | (token_value[1] & 0x0007));
 }
}
// C: is floppy
if(number_of_floppies() > 2)
{
 if(token_value[0] == 0x0c)
 {
  outcmos(0x11,(incmos(0x11) & 0x0f) | ((token_value[1] & 0x0007) << 4));
 }
}
// D: is floppy
if(number_of_floppies() > 3)
{
 if(token_value[0] == 0x0d)
 {
  outcmos(0x11,(incmos(0x11) & 0xf0) | (token_value[1] & 0x0007));
 }
}
//------- these are for hard disks ---------
if(number_of_floppies() <= 2)
// C: and D: are hard drives
{
 if(token_value[0] == 0x0c)
 {
  if (token_value[1] < 14) // use primary cell
  {
   outcmos(0x12,(incmos(0x12) & 0x0f) | ((token_value[1] & 0x000f) << 4));
  }
  else  // use secondary cell
  {
   outcmos(0x12,incmos(0x12) | 0xf0);
   outcmos(0x19,token_value[1]);
  }
 }
 if(token_value[0] == 0x0d)
 {
  if (token_value[1] < 14) // use primary cell
  {
   outcmos(0x12,(incmos(0x12) & 0xf0) | (token_value[1] & 0x000f));
  }
  else  // use secondary cell
  {
   outcmos(0x12,incmos(0x12) | 0x0f);
   outcmos(0x1a,token_value[1]);
  }
 }
}
if(number_of_floppies() == 3)
// D: and E: are hard drives
{
 if(token_value[0] == 0x0d)
 {
  if (token_value[1] < 14) // use primary cell
  {
   outcmos(0x12,(incmos(0x12) & 0x0f) | ((token_value[1] & 0x000f) << 4));
  }
  else  // use secondary cell
  {
   outcmos(0x12,incmos(0x12) | 0xf0);
   outcmos(0x19,token_value[1]);
  }
 }
 if(token_value[0] == 0x0e)
 {
  if (token_value[1] < 14) // use primary cell
  {
   outcmos(0x12,(incmos(0x12) & 0xf0) | (token_value[1] & 0x000f));
  }
  else  // use secondary cell
  {
   outcmos(0x12,incmos(0x12) | 0x0f);
   outcmos(0x1a,token_value[1]);
  }
 }
}
if(number_of_floppies() == 4)
// E: and F: are hard drives
{
 if(token_value[0] == 0x0e)
```

```
   {
    if (token_value[1] < 14) // use primary cell
    {
     outcmos(0x12,(incmos(0x12) & 0x0f) | ((token_value[1] & 0x000f) << 4));
    }
    else  // use secondary cell
    {
     outcmos(0x12,incmos(0x12) | 0xf0);
     outcmos(0x19,token_value[1]);
    }
   }
   if(token_value[0] == 0x0f)
   {
    if (token_value[1] < 14) // use primary cell
    {
     outcmos(0x12,(incmos(0x12) & 0xf0) | (token_value[1] & 0x000f));
    }
    else  // use secondary cell
    {
     outcmos(0x12,incmos(0x12) | 0x0f);
     outcmos(0x1a,token_value[1]);
    }
   }
  }
 calc_cmos(); // recalc checksum
}
else
{
 if(value_present(1)) // if anything entered
 {
  // if value = 0-4, then # of drives in system
  if(token_value[1] <= 4)
  {
   if(token_value[1] == 0) // set no-drives in system
   {
    outcmos(0x14,incmos(0x14) & ~0x01);
    calc_cmos();
   }
   else // there are drives in system
   {
    outcmos(0x14,incmos(0x14) | 0x01);
    outcmos(0x14,(incmos(0x14) & 0x3f) | \
    ((token_value[1]-1) << 6));
    calc_cmos();
   }
  }
 }
}
write_string("Options are\n\r");
write_string("0-4 = # of floppy drives\n\r");
write_string("A:n - F:n = set drive type\n\r");
write_string("For Floppy drives:\n\r");
write_string("  0 =  no drive\n\r");
write_string("  1 =  360k drive\n\r");
write_string("  2 =  1.2M drive\n\r");
write_string("  3 =  720k drive\n\r");
write_string("  4 = 1.44M drive\n\r");
write_string("For Hard drives:");
write_string("  n = drive type\n\r");
write_string("For list of hard drive types, enter ""Types""");
display_drives();
break;

/*------ list drive types from hdisk table ----*/

case command_TYPES:
display_hdisk_types();
break;

/*------- set memory sizes ----------*/

case command_MEM:

if(value_present(1)) // if anything entered
{
 set_sys_mem(dec2bin(token_value[1]));
}
if(value_present(3)) // if anything entered
{
 set_ext_mem(dec2bin(token_value[3]));
}
write_string("  System Memory Size is - ");display_sys_mem();
write_string("\n\rExtended Memory Size is - ");display_ext_mem();

break;
```

```
/*--- select NPX present/not present ----*/

case command_NPX:
if(value_present(1))
{
 if((token_value[1] & 0x01) == 0)
 {
  outcmos(0x14,incmos(0x14) & ~0x02);
 }
 else
 {
  outcmos(0x14,incmos(0x14) | 0x02);
 }
 calc_cmos();
}
write_string("Options are: 0 = No, 1 = Yes\n\r");
write_string("NPX ");
display_npx();
break;


/*------- set video type ----------*/

case command_VIDEO:
write_string("Options are\n\r0=EGA\n\r1=CGA 40x25\n\r2=CGA 80x25\n\r3=Monochrome");
if(value_present(1)) // if anything entered
{
 // chekc for argument range validity
 if((token_value[1] >= 0) && (token_value[1] <= 3))
 {
  // this is where we set the video bits
  outcmos(0x14,(incmos(0x14) & 0xcf) | ((token_value[1] & 0x0003) << 4));
  calc_cmos();
 }
 else
 {
  write_string("\n\rVideo type argument must be between 0-3");
     beep();
 }
}
else
{
 write_string("\n\rTo change Expected Video, type ""VIDEO n <enter>""\n\r");
}
display_video();
break;


/*======== Input from port Command ======*/

case command_IB: mode_size = byte; lbyte(inportb(token_value[1])); break;

case command_IW: mode_size = word; lword(inport(token_value[1])); break;

case command_I:
switch (mode_size)
{
 case byte: lbyte(inportb(token_value[1])); break;
 case word: lword(inport(token_value[1])); break;
}
break;

/*========= Output from Port Commands =========*/

case command_OB: mode_size = byte; outportb(token_value[1],token_value[3]); break;

case command_OW: mode_size = word; outport(token_value[1],token_value[3]); break;

case command_O:
switch (mode_size)
{
 case byte: outportb(token_value[1],token_value[3]); break;
 case word:  outport(token_value[1],token_value[3]); break;
}
break;

/*=== Move Command ===*/

case command_M:

do
{
 pokeb(token_value[4],token_value[5]++,
 peekb(token_value[0],token_value[1]++));
 if(console_break()) break;
}
```

```
while ((token_value[1] < token_value[3]) &&
(token_value[1] != 0));
break;

/*=== Register Display Command ===*/

case command_R:    /* display all the registers */
display_regs(&reg_saves);
break;

/*=== Register Display and Change Commands ===*/

case command_RAX: reg_index = axsave; goto r_display;

case command_RBX: reg_index = bxsave; goto r_display;

case command_RCX: reg_index = cxsave; goto r_display;

case command_RDX: reg_index = dxsave; goto r_display;

case command_RSP: reg_index = spsave; goto r_display;

case command_RBP: reg_index = bpsave; goto r_display;

case command_RSI: reg_index = sisave; goto r_display;

case command_RDI: reg_index = disave; goto r_display;

case command_RDS: reg_index = dssave; goto r_display;

case command_RES: reg_index = essave; goto r_display;

case command_RSS: reg_index = sssave; goto r_display;

case command_RCS: reg_index = cssave; goto r_display;

case command_RIP: reg_index = ipsave; goto r_display;

r_display:

/* if there is a command line input, just store it */

if(value_present(1) == true)
{
 reg_saves[reg_index] = token_value[1];
}
else
{
 write_string(peek(bios_cs(),&register_token_list[reg_index-1]));
 co(' ');
 lword(reg_saves[reg_index]);
 co(cr); co(lf); co(':');

 /* solicit input for register */

 get_line_input(cr);
 get_values();
 if(value_present(1))
 {
  reg_saves[reg_index] = token_value[1];
 }
}
break;

/*-------- Flag Register Command ---------*/

case command_RF:

/* we have to look at command line for flag alpha's */
/* if there is a command line input, just store it */

command_index = get_token_number(&flag_token_list);

if(command_index == 0)  // solicit input
{
 display_flags();
 write_string(" - ");
 get_line_input(cr);
 get_token();
 command_index = get_token_number(&flag_token_list);
}

if (command_index != 0)
{
 command_index--; // rebase to zero
 lword(command_index);
 if ((command_index & 1) == 0)
 {
  /* even number - clear bit */
  reg_saves[pswsave] &= ~(1 << (command_index/2));
 }
 else
 {
  /* odd number - set bit */
  reg_saves[pswsave] |= (1 << (command_index/2));
 }
}
break;

/*========= Read Intel Hex =================*/

case command_RHEX:

/* start looking for record marker */

do
{
 while(inchar = ci() != ':') {}
 sum = 0;
 char_count = get_byte();
 token_value[1] = get_word();
 record_type = get_byte();

 switch (record_type)
 {
  case data_record:
  if (char_count == 0)
  {
   record_type = end_of_file_record;
   break;
  }
  while (char_count > 0)
  {
   pokeb(token_value[0],token_value[1],get_byte());
  }
  get_byte();
  if (sum != 0) record_type = end_of_file_record;
  break;

  case end_of_file_record:
  /* return to sysvue main */
  break;

  case extended_address_record:
  /* set load address segment */
  token_value[0] = get_word();
  get_byte();
  if (sum != 0) record_type = end_of_file_record;
  break;

  case start_address_record:
  /* set cs:ip of reg set */
  reg_saves[cssave] = get_word();
  reg_saves[ipsave] = get_word();
  get_byte();
  if (sum != 0) record_type = end_of_file_record;
  break;
 }
}
while (record_type != end_of_file_record);
break;

/*========= Write Intel Hex =================*/

case command_WHEX:

/* disregard if range = nul */
if ((token_value[1] || token_value[3]) != 0)
{
 /* write extended address record for segment */
 write_string("\r\n:");
 sum = 0;
 pbyte(2);pword(0);pbyte(2);
 pword(token_value[0]);
 pbyte(sum);
 do
 {
  write_string("\r\n:");
  if ((token_value[3] - token_value[1]) > 16)
  {
   char_count = 16;
  }
  else
  {
   char_count = token_value[3] - token_value[1];
  }
  sum = 0;
  pbyte(char_count); /* record count */
  pword(token_value[1]); /* address */
  pbyte(0); /* data mode_type record */
  while (char_count > 0)
  {
   pbyte(peekb(token_value[0],token_value[1]));
   token_value[1]++;
   char_count--;
  };
  pbyte(sum);
  if (console_break()) break;
 }
 while ((token_value[1] < token_value[3]) &&
 (token_value[1] != 0));

 /* write start address record */
 if ((token_value[4] || token_value[5]) != 0)
 {
  write_string("\r\n:");
  sum = 0;
  pbyte(4);pword(0);pbyte(4);
  pword(token_value[4]);
  pword(token_value[5]);
  pbyte(sum);
 }
 write_string("\r\n:00000001FF");
}
break;

/*========= Search Commands ================*/

case command_SA: mode_size = byte; mode_type = ascii; goto search_case;

case command_SB: mode_size = byte; mode_type = hex; goto search_case;

case command_SW: mode_size = word; mode_type = hex; goto search_case;

case command_SD: mode_size = dword; mode_type = hex; goto search_case;

case command_S:

search_case:

/* if no mask entered, then all bits significant */
if((token_value[6] | token_value[7] ) == 0)
{
 token_value[6] = ~token_value[6];
 token_value[7] = ~token_value[7];
}

if (mode_size == byte)
/* clear un-needed bits from mask */
{
 token_value[6] = 0;
 token_value[7] &= 0x00ff;
}

if (mode_size == word)
/* clear un-needed bits from mask */
{
 token_value[6] = 0;
}

do
{
 if((((peek(token_value[0],token_value[1]) & token_value[7]) ==
               (token_value[5] & token_value[7]))) &&
 (((peek(token_value[0],token_value[1]+2) & token_value[6]) ==
                         (token_value[4] & token_value[6]))))
 {
  co(13); co(10); /* cr-lf */
  lword(token_value[0]);
  co(':');
  lword(token_value[1]);
  co(' ');

  if (mode_size == byte)
  {
   lbyte(peekb(token_value[0],token_value[1]));
  }
```

```
   if (mode_size == word)
   {
    lword(peek(token_value[0],token_value[1]));
   }

   if (mode_size == dword)
   {
    lword(peek(token_value[0],token_value[1]+2));
    co('-');
    lword(peek(token_value[0],token_value[1]));
   }
  }
 if(console_break()) break;
 token_value[1] ++;
}
while ((token_value[1] < token_value[3]) &&
(token_value[1] != 0));
break;

/*========= Checksum Command ===========*/

case command_CHK:
sum = 0;
do
{
 sum += peekb(token_value[0],token_value[1]++);
 if (console_break()) break;
}
while ((token_value[1] < token_value[3]) &&
(token_value[1] != 0));
/* add in the last byte */
sum += peekb(token_value[0],token_value[1]);
write_string("Checksum = ");
lbyte(sum);
break;

/*========== Re-Boot Commands ===========*/

case command_COLD_BOOT:
RESET_FLAG = 0; /* reset the warm boot flag */
disable();     incmos(0x00);                   // set NMI mask
far_call(bios_cs(),&reset);
break;

case command_WARM_BOOT:
/* reset the warm boot flag */
RESET_FLAG = 0x1234;
disable();     incmos(0x00);                   // set NMI mask
far_call(bios_cs(),&reset);
break;

case command_HARD_BOOT: /* forces cs: to f000 */
RESET_FLAG = 0; /* reset the warm boot flag */
disable();     incmos(0x00);                   // set NMI mask
far_call(0xf000,0xfff0);
break;

/*========== Trace  Commands ===========*/

case command_T:
case command_TRACE:
trace_count = token_value[1];
if(trace_count == 0) trace_count = 1;
exit_flag=1;
reg_saves[pswsave] |= 0x0100; // set trace bit
// chain-in the trace vector
trace_chain = get_vector(0x1);
link_interrupt(0x1,Systrace);
break;

/*======= Select Watch parms command =======*/

case command_SELECT:
if (value_present(1))
{
 // modify parms
 watch_selection = token_value[1];
}
write_string("Options are (the OR) as follows:\n\r");
lword(VIDEO);  write_string(" = Video\n\r");
lword(VUE);    write_string(" = SysVue\n\r");
lword(CASSETTE); write_string(" = Cassette\n\r");
lword(EQUIPMENT); write_string(" = Equip\n\r");
lword(MEMORY);  write_string(" = Mem\n\r");
lword(LPT);  write_string(" = Lpt\n\r");
lword(COM);  write_string(" = Com\n\r");
```

```
    lword(FLOPPY);  write_string(" = Floppy\n\r");
    lword(HARD);  write_string(" = Hard\n\r");
    lword(TOD);  write_string(" = Tod\n\r");
    lword(BOOT);  write_string(" = Boot\n\r");

    write_string("Current Watch Selections are: ");
     watch_selection &= WATCH_MASK;
    if(watch_selection == 0) write_string("None");
    if((watch_selection & VIDEO) != 0) write_string("Video ");
    if((watch_selection & VUE) != 0) write_string("SysVue ");
    if((watch_selection & CASSETTE) != 0) write_string("Cass ");
    if((watch_selection & EQUIPMENT)!= 0) write_string("Equip ");
    if((watch_selection & MEMORY) != 0) write_string("Mem ");
    if((watch_selection & LPT) != 0) write_string("Lpt ");
    if((watch_selection & COM) != 0) write_string("Com ");
    if((watch_selection & FLOPPY) != 0) write_string("Floppy ");
    if((watch_selection & HARD) != 0) write_string("Hard ");
    if((watch_selection & TOD) != 0) write_string("Tod ");
    if((watch_selection & BOOT) != 0) write_string("Boot ");
    if((watch_selection & TIMER) != 0) write_string("Tmr ");
    if((watch_selection & KEYBOARD) != 0) write_string("Kb ");

    write_string("\n\rWatch is currently ");
    if(watch_flag == 0)
    {
     write_string("Off");
    }
    else
    {
     write_string("On");
    }
    break;

    /*=========== Watch command ==============*/

    case command_WATCH:
    // toggle watch flag
    if (watch_flag != 0)
    {
     watch_flag = 0;
     Write_String(" Off");
    }
    else
    {
     Write_String(" On");
     watch_flag = watch_selection;
     if(redirect_flag != 0) watch_flag &= ~LPT;
    }
    break;


    /*=========== Re-direction  commands ==============*/

    case command_TO_CON:
    case command_BACK_TO_CON:
    redirect_flag = 0;
    break;

    case command_TO_LPT:
    redirect_flag = 1;
    // If you are watching the printer driver, it must not be copying screen output to the printer.
    // An insidious recursion results, blowing the system away !
    if((watch_flag & LPT) != 0) watch_flag &= ~LPT;
    break;

  }
 }
};

ax = reg_saves[axsave];
bx = reg_saves[bxsave];
cx = reg_saves[cxsave];
dx = reg_saves[dxsave];
si = reg_saves[sisave];
di = reg_saves[disave];
ds = reg_saves[dssave];
es = reg_saves[essave];
flags = reg_saves[pswsave];
cs = reg_saves[cssave];
ip = reg_saves[ipsave];


// If tracing, we should link the trace vector here
// If trapping (breakpointing), we should link the trap vector here.

set_vector(0x1b,break_chain >> 16,break_chain);
```

```
  sysvue_busy = 0; /* release Sysvue */
 }
 release_block(myblock);
}

/********************************************************/

/*=========== Console Status Routines ============*/

unsigned console_break()
/* returns true for abort operation */
{
 if (break_flag == 0) return(false);
 break_flag = 0;
 return(true);
}

/*-----------------------------------------------*/

/* dump byte and word functions */

void pword(w) /* print word, accumulate checksum */
{
 pbyte(w >> 8); pbyte(w);
}

void pbyte(outchar) /* print byte, accumulate checksum */
{
 sum -= outchar; /* build checksum for whex command */
 lbyte(outchar);
}

/* dummy nmi interrupt routine */

void nmi_int()
{
 ;
}

/*--------- supporting functions -------------*/

// This function displays the register set. It is used by the 'R' Command and the 'INT' Command.

void display_regs(unsigned *reg_saves)
{
 co(cr); co(lf);
 for (token_index = 0;token_index < 13;token_index++)
 {
  write_string(peek(bios_cs(),&register_token_list[token_index]));
  co('=');
  lword(reg_saves[token_index+1]);
  co(' '); co(' ');
  if (token_index == 7) { co(cr); co(lf); }
 }
 co(' '); co(' ');
 display_flags(reg_saves);
}

void display_flags(unsigned *reg_saves)
{
 unsigned even_odd;
 int scan_index;
 /* display the flag conditions */

 for (scan_index = 15; scan_index > -1; scan_index--)
 {
  even_odd = (reg_saves[pswsave] >> scan_index) & 1;

  if (peekb(bios_cs(),peek(bios_cs(),&flag_display_token_list[scan_index][even_odd])) != '-')
  {
   write_string(peek(bios_cs(),&flag_display_token_list[scan_index][even_odd]));
   co(' ');
  }
 }
}

/*------- get a byte for rhex ---------*/

char get_byte()
{
 char cc,d;
 cc = ci() - '0';
 if (cc > 9) cc -= 7;
 d = ci() - '0';
 if (d > 9) d -= 7;
 cc = cc << 4 | d;
 sum += cc;
 return (d);
}

/*------- get a word for rhex ---------*/

unsigned get_word()
{
 return ((get_byte() << 8) | get_byte());
}

unsigned isdelim(char c) /* returns true or false */
{
 int q=0;
 char cc;
 /*  write_string("\n\rIsdelim "); lbyte(c);co(' ');*/
 for (;(cc = peekb(bios_cs(),&delim_list[q++])) != 0;)
 {
  /*    lbyte(cc);co(' ');*/
  if (cc == (c & 0xff)) return(true);
 };
 return(false);
}

unsigned get_token()
{
 // this gets the next token from the line input buffer and puts it into the token buffer

 // It uses the buffer_index and advances until end of data.
 // It returns true if a token was found and false if no token found

 if ((buffer[buffer_index] == 0) || (buffer[buffer_index] == cr)) return(false);
 /* move from buffer to token buffer */
 for (token_index=0; (isdelim(buffer[buffer_index]) == false) && (buffer[buffer_index] != 0) ;
 token_buffer[token_index++]=buffer[buffer_index++]);
 token_buffer[token_index] = 0;
 /* put in end of string marker */
 last_delim = buffer[buffer_index];
 buffer_index++; /* advance buffer index over delimiter */
 return (true);
}

/*...........................................*/


void syntax_error(void)
{
 write_string("\n\rSyntax Error");
}

// Get Line Input is the normal input function. It always terminates on a cr OR on the char
// specified on the callers arg. If the caller wants only a CR termimation, then he calls with CR
// as the arg.

void get_line_input(term)  /*---- get the input line ----*/
{
 buffer_index = 0;
 do
 {
  if((inchar = ci()) == bspace)
  {
   if (buffer_index != 0)
   {
    buffer_index--;
    co(bspace);
    co(' ');
    co(bspace);
    buffer[buffer_index] = 0;
   }
  }
  else    /* not a backspace */
  {
   /* lower to upper case */
   if ((inchar >= 'a') && (inchar <= 'z')) inchar ^= 0x20;
   co(inchar);  /* echo character */
   if((inchar != term) && (inchar != cr))
   /* if a regular character */
   {
    buffer[buffer_index] = inchar;
    buffer_index++;
   }
   else  /* a cr or a term char */
   {
    buffer[buffer_index] = 0;
    buffer[buffer_index+1] = 0;
    if (inchar == cr) co(lf);
   }
  }
 }
 while ((inchar != term) && (inchar != cr));
 buffer_index = 0;
}

/*--- get the value specified by this token ---*/

unsigned get_token_value()
{
 char cc;
 unsigned token_value;
 token_value = 0; /* pre-clear token value cell */
 token_index = 0;
 token_number = get_token_number(&register_token_list);
 if (token_number == 0) /* it is a value, not a token */
 {  /* determine value */
  token_index = 0;
  while (token_buffer[token_index] != 0)
  {
   cc = token_buffer[token_index];
   if (ishex(cc) == true)
   {
    if (isalpha(cc) == true) cc += 9;
    cc &= 0x0f; /* now it is a nibble */
    /* merge it into token */
    token_value = (token_value << 4) | cc;
   }
   token_index++;
  }
  return(token_value);
 }
 else

 // if there is a token #, then get the value of that token

 {
  return(reg_saves[token_number-1]);
 }
}

/*--------------------------------------------------*/
// search the specified token list against the token buffer and return the token number if a find -
// no find - return token_number = 0

unsigned get_token_number(unsigned *token_list_ptr)
{
 unsigned token_index,list_index=0,token_number=1;
 /* points to current string being checked */
 unsigned list_item;
 unsigned item_index=0;

 while
 ((list_item = peek(bios_cs(), &token_list_ptr[list_index++])) != -1)
 {
  for (token_index = 0,item_index = 0;
  ((token_buffer[token_index] != 0) && (peekb(bios_cs(),list_item+item_index) != 0) &&
  (token_buffer[token_index] == peekb(bios_cs(), list_item+item_index)));
  /* co(peekb(bios_cs(),list_item+list_index)),*/
  token_index++,item_index++);

  // If they are both at the end of string, then they matched.

  if ((token_buffer[token_index] == 0) && (peekb(bios_cs(),list_item+item_index) == 0))
  {
   /*
   **    write_string("\n\rToken is "); lword(token_number);
   */
   return(token_number);
  }
  /* count the tokens */
  token_number++;
  /*
  **  write_string("\n\rToken # ");
  **    lword(token_number);
  **    write_string("\n\rList index ");
  **    lword(list_index);
  **    write_string("\n\rList pointer ");
  **    lword(list_item);
  */

 };
 /*  co(peekb(bios_cs(),list_item+list_index));
 */
 return(0);
```

```
}
/*----------------------------------------------*/

/* The sysbreak routine is invoked by the CTRL-BREAK or CTRL-C keys and sets the break flag.
** This is checked by the CSTS routine to see if an operation should be aborted. It is linked
** upon entry to SysVue and de-linked upon exit.
*/

void interrupt cdecl far sysbreak(interrupt_registers)
{
 cstods(); // do this because it is assumed that cs = ds
 setds_system_segment();
 /* all variables will be referenced in sys seg */
 break_flag = 1;
}

/* This function evaluates the arguments on the command line and gets the values associated with
** the args and places their values into the token_value array.
** The values are arranged:
** seg:off - seg:off - seg:off - seg:off
** The corresponding present_bit is set in the 'present' char to indicate if a an arg was
** present for that value.
*/

get_values()
{
 unsigned i; /* i is a local variable used for the index */
 /* clear the token value array */
 for (i=0; i< 8;token_value[i++] = 0); present = 0;  i = 0;
 /* write_string("\n\rEntering get_values ");
 */
 do
 {
  if(get_token() == true) /* there is a token - */
  {
   token_value[i+1] = get_token_value();
   present |= (0x80 >> (i+1));
   if ((last_delim == ':') || (last_delim == '/'))
   /* the token is a register token */
   {
    token_value[i] = token_value[i+1];
    present |= (0x80 >> i);
    present &= ~(0x80 >> (i+1));;
    token_value[i+1] = 0;
    if(get_token() == true) /* there is a token - */
    {
     token_value[i+1] = get_token_value();
     present |= (0x80 >> (i+1));
    }
   }
  }
  i += 2;
 }
 while (i < 8);
 /*
 **write_string("\n\rExiting get_values");
 */
}

/* Value present looks at the present bit for token_values 0-7 and returns a true if the bit is
** set, indicating a value was entered
*/

unsigned value_present(index)
{
 if (((present << index) & 0x80) != 0) return(true);
 return(false);
}
```

SECTION F

Odds and Ends

This section includes tips on loading a Test Bios into Static Ram by using Debug Batch files, and other assorted subjects.

ONE

Loading Bios Test Ram

The following example shows how a Static Ram Adapter card installed in a target system can be loaded with the Bios image to test the Bios without programming Eproms. This example assumes the target machine has 64 Kbytes of static Ram at segment D000. Note that the Adapter uses static rather than dynamic Ram and that the Ram is located in the high memory map area, rather than in the System Ram area (0 - 640 K). The low Ram area is cleared when a Bios starts, so the Bios image must be outside the 0-640 K range. Dynamic Ram Refresh is re-initialized during the Bios Power-up, so Static Ram is required for the Test-Bios to remain intact.

Create a file named **loadat.inp** file (as shown below) to provide input to the debug program. It will be used to automate the copying of the Bios image from the DOS transient area to the static Ram location. Note that the transfer is done in 4000 byte portions, as the debug program treats the length argument (*4000*) as a signed integer. A value of 8000 would be unreliable. Following the copying commands, a Go command starts executing the Bios at its *reset* label. By inspecting the Bios.map file, the offset for the label *reset* may be verified.

```
nbios.bin
l
m cs:0000 l 4000 d000:0000
m cs:4000 l 4000 d000:4000
m cs:8000 l 4000 d000:8000
m cs:c000 l 4000 d000:c000
g=d000:e05b
```

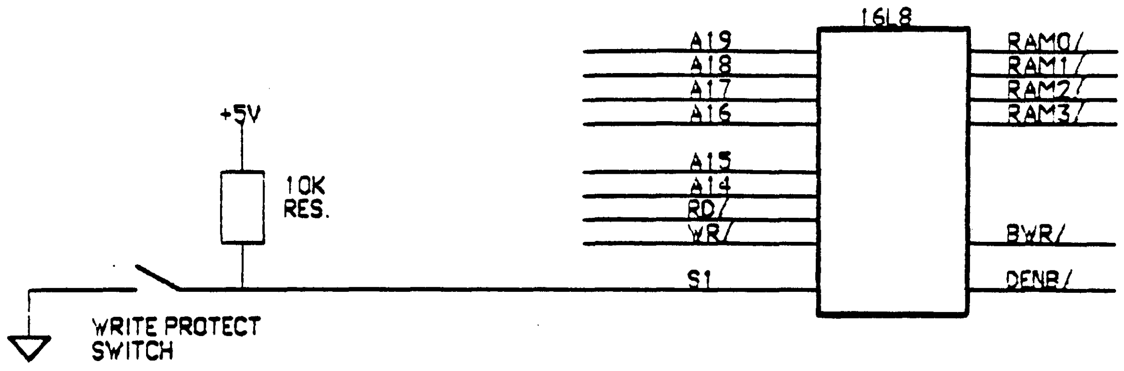
To perform this loading procedure, enter **debug <loadat.inp**

Next, to avoid typing the full command **debug <loadat.inp** every time you want to load the Ram, create a file such as **load.bat** which contains the one line as shown below:

```
debug <loadat.inp
```

Now all that is required to perform the complete copy, load, and go procedure is to enter **load** and allow the batch file to invoke the process.

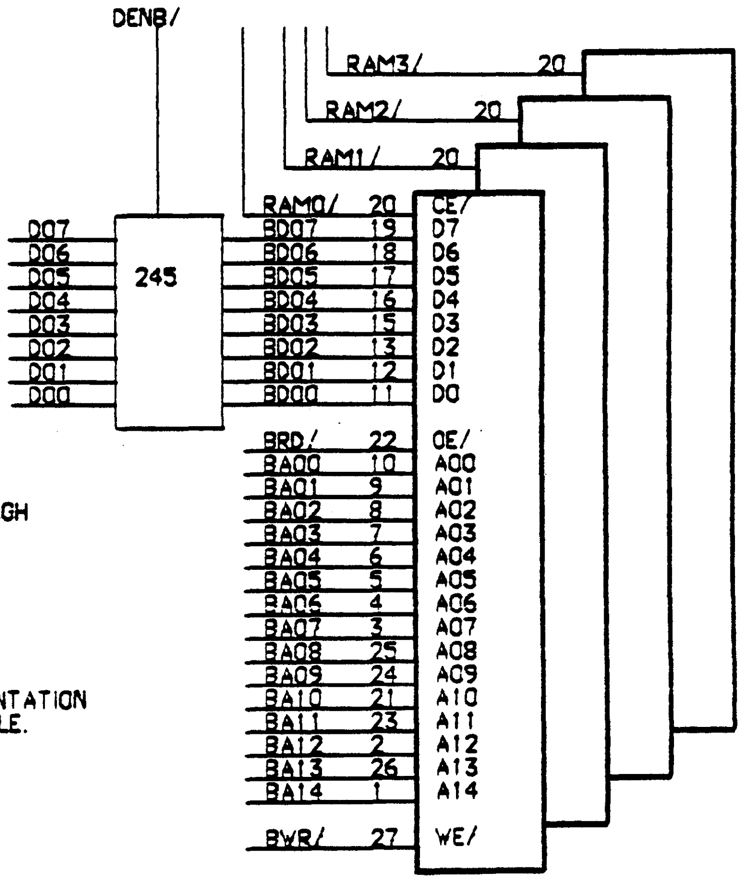
A sample schematic is included for a static ram adapter which you may wish to use as a design guideline in case you decide to construct your own adapter. This schematic shows the general philosophy you can use in your design. Fosco and Annabooks have not constructed an adapter from this schematic, but have used the same design approach in the past. The functions shown in the programmable logic device (16L8) can be duplicated with discrete logic, but a PAL approach is the simplest. PAL's are a specific family of programmable devices by MMI, and PAL is a registered trademark of MMI. An adapter of this type will also accommodate the Dallas Semiconducter battery backed ram modules, so a non-volatile ram board can be configured.



EQUATIONS.
BWR/ = WR/ * S1/
RAM0/ = A19/ * A18/ * X * Y * A15/ * A14/
RAM1/ = A19/ * A18/ * X * Y * A15/ * A14
RAM2/ = A19/ * A18/ * X * Y * A15 * A14/
RAM3/ = A19/ * A18/ * X * Y * A15 * A14
DENB/ = RD/ * RAM0/
+ RD/ * RAM1/
+ RD/ * RAM2/
+ RD/ * RAM3/

WRITE PROTECT
RAM SELECTS

DATA ENABLE FOR DATA BUFFER



NOTES.

FOR SEGMENT D000:
X = A17/
Y = A16

FOR SEGMENT E000:
X = A17
Y = A16/

28-PIN SOCKETS WILL ACCEPT
62256 RAM's
DALLAS 1235 RAM's

DATA LINES SHOULD BE BUFFERED THROUGH
A 245 IN ORDER TO DRIVE THE BUS.

ADDRESS LINE BUFFERING IS OPTIONAL

DECODER IS MMI 16L8 PAL.

THIS SCHEMATIC IS FOR EXAMPLE ONLY.
FOSCO & ANNABOOKSMAKE NO REPRESENTATION
AS TO THE CORRECTNESS OF THE EXAMPLE.

BiosKit Static Ram Board	FOSCO	28035 MOUNTAIN MEADOW ROAD ESCONDIDO, CA. 92026 619+744-8086

TWO

The Indent Program

The Indent program is used to indent the lines of a source file according to the nesting of the left and right bracket characters ({ and }) to ease the readability of the code. Some editors have built-in features to do this automatically for you, so this Indent program may not be needed. If you are using an ordinary editor, you will find this program useful.

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: Indent
*
* Version: 1.01
*
* Author:FOSCO
*
* Date: 10-10-89
*
* Filename: Indent.c
*
* Functional Description:
*  This program is a  utility which is used to indent C source code lines according to the nesting
*  level of the left and right bracket characters ( { } ). It improves readability of the souce
*  file, but does not affect the logic flow. It is useful when a general purpose editor is being
*  used to prepare source files.
*
* Arguments:
*  File name
* Return:
*
* Version History:
* 1.01
*  Updated under QuickC 2 for convenience.
*  General clean-up and simplification.
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

#include <stdio.h>

/* G L O B A L   V A R I A B L E S */

FILE *instream;
FILE *outstream;
unsigned int length,i,numread,numwritten, tab = 0;
#define blen 512
unsigned char instring[blen],outstring[blen],buffer[16384];
#define true  -1
#define false  0
unsigned write_ok = true;

/* L O C A L   D E F I N I T I O N S */

#define tabchar 9
#define rt_brace 125
#define lf_brace 123
#define indent_level 1 /* # of columns to indent */

/* P R O G R A M */

/* indent c programs according to {} characters */

main(int argc, char *argv[])
{
 printf("FOSCO/Annabooks Indent Source Utility V1.01\n\r");
 if (argc <2) { printf("Not enough arguments on command line\n"); exit(0); }

 if ((instream = fopen(argv[1],"r")) == NULL)
         printf("Could not open file for reading\n");
 else
 {
  length = filelength(fileno(instream));
if (length == -1) printf("Bad Length on input file\n");
else
{
 if ((outstream = fopen("$id.$$$","w")) == NULL)
        printf("Could not open file for writing\n");
 else
 {
  tab=0;
  while (fgets(instring,blen,instream) != 0)
  {
   // skip over leading spaces
   i=0; while (instring[i++] == ' '); i--;
   // check for right brace here so we de-indent the current line
   if ((instring[i] == rt_brace) && (tab >= indent_level))
        tab -= indent_level;

   // pad front of output string with required # of spaces
   strnset(outstring,' ',tab);outstring[tab] = 0;

   // now copy instring[i] to outstring[tab]
   strcat(&outstring[tab],&instring[i]);

   // write to the temp file and mark any errors
   if(fputs(outstring,outstream) != 0) write_ok = false;;
   // check for left braces here so we indent the following line
   if  (instring[i] == lf_brace)
   {
    tab += indent_level;
    // check for closing brace on same line
    for(; i<strlen(instring);i++)
    {
     if ((instring[i] == rt_brace) && (tab >= indent_level))
     {
      tab -= indent_level;
      break;
     }
    }
   }
  }
 }
}
fclose(instream);
fclose(outstream);

if(!write_ok)
{
 printf("\n\rError in writing temporary file, processing aborted.");
 printf("Original file %s remains unchanged.\n\r",argv[1]);
}
else
{
 // now copy the $id.$$$ to the original file
 if (((instream = fopen("$id.$$$","r+b")) != NULL) &&
 ((outstream=fopen(argv[1],"w+b")) != NULL))
 {
  do
  {
   numread = fread((char *)buffer, sizeof(char),16384,instream);
   numwritten = fwrite((char *)buffer, sizeof(char),numread,outstream);
   if (numwritten != numread)
   {
    printf("Transfer error on file %s. Use $ID.$$$ as recovery file./n",argv[1]);
    write_ok = false;
   }
  }
  while (numread > 0);
  fclose(instream);
  fclose(outstream);
  if(write_ok)
        if ((numread = remove ("$id.$$$")) != 0)
                printf("\n\rError deleting temp file $ID.$$$\n\r");
 }
 }
}
}
```

THREE

The Hilite Program

The Hilite program is used to display program comments in intensified video on the display. It scans the source file and looks for pairs of comment delimiters (/* and */), and intensifies the video between these marks. It is useful for detecting incorrect comment delimiter pairs, which may comment out wanted source code statements. When strange errors show up in the compilation of a 'slightly' edited program, comment delimiter errors may cause drastic errors. If a program compiles properly but program operation changes radically, a comment check with Hilite may help in determining the cause(s).

```
/*****************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: Hilite
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 1-1-89
*
* Filename: hilite.c
*
* Functional Description:
*  This program is used to hilite the comments in a C source program as it is displayed on
*  the screen.
*
* Arguments:
*   File name
* Return:
*
* Version History:
*
*****************************************************************************************************/

/* I N C L U D E   F I L E S */

/* F U N C T I O N   P R O T O T Y P E S */

/* G L O B A L   V A R I A B L E S */

/* G L O B A L   C O N S T A N T S */

/* L O C A L   D E F I N I T I O N S */

/* L O C A L   C O N S T A N T S */

/* P R O G R A M */

/* highlight c programs according to / * * / characters */

#include <stdio.h>
#include <dos.h>
#include <ctype.h>
#define cr 13
#define lf 10
#define space 32
#define on 1
#define off 0
#define star '*'
#define slash 47
#define bspace 8;

FILE *instream;
unsigned int length;
unsigned int i;
unsigned char comment,ch,last_char;
unsigned char buffer[16384];
union REGS regs;

main(argc,argv)
int argc;
```

```
char *argv[];
{
 if (argc <2)
 {
  printf("Not enough parameters on command line\n");
  exit(0);
 }
 if ((instream = fopen(argv[1],"r+b")) == NULL) printf("Could not open file for reading\n");
 else
 {
  length = filelength(fileno(instream));

  for (i=0; i<length && ((ch = getc(instream)) != EOF); i++)
  {
   if ((last_char == slash ) && (ch == star ))
   {
    comment = on; /* backspace and turn on last slash */

    regs.h.ah = 0x0e;
    regs.h.al = bspace;
    regs.h.bh = 0;
    int86(0x10,&regs,&regs);
    regs.h.bl = 0x0f;

    regs.h.ah = 0x09; /* write attribute at cursor */
    regs.h.al = 0x20; /* display without highlight */
    regs.h.bh = 0x00;
    regs.x.cx = 0x0001;
    int86(0x10,&regs,&regs);

    regs.h.ah = 0x0e;
    regs.h.al = slash;
    regs.h.bh = 0;
    int86(0x10,&regs,&regs);
    regs.h.bl = 0x0f;
   }

   if (comment == on)
   {
    regs.h.bl = 0x0f;
   }
   else
   {
    regs.h.bl = 0x07;
   }
   regs.h.ah = 0x09; /* write attribute at cursor */
   regs.h.al = 0x20; /* display without highlight */
   regs.h.bh = 0x00;
   regs.x.cx = 0x0001;
   if (isprint(ch)) int86(0x10,&regs,&regs);

   regs.h.ah = 0x0e;  /* write tty   */
   regs.h.al = ch;
   regs.h.bh = 0x00;
   int86(0x10,&regs,&regs);
   if ((last_char == star ) && (ch == slash )) comment = off;
   last_char = ch;
  }
  fclose(instream);
 }
}
```

FOUR

The Parens Program

The Parens program is used to display parenthesized text in intensified video on the display. It scans the source file and looks for parentheses characters and intensifies the video between these marks. It is useful for detecting incorrect parenthesizing.

```
/*******************************************************************************************************
*
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
*
* Module Name: Parens
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 1-5-89
*
* Filename: parens.c
*
* Functional Description:
* This progam is used to display parenthesized text on the screen in itensified video.
*
* Arguments:
*  File name
*
* Return:
*
* Version History:
*
*******************************************************************************************************/

/* I N C L U D E   F I L E S */

/* F U N C T I O N   P R O T O T Y P E S */

/* G L O B A L   V A R I A B L E S */

/* G L O B A L   C O N S T A N T S */

/* L O C A L   D E F I N I T I O N S */

/* L O C A L   C O N S T A N T S */

/* P R O G R A M */

/* highlight c programs according to () characters */

#include <stdio.h>
#include <dos.h>
#include <ctype.h>
#define cr 13
#define lf 10
#define space 32
#define on 1
#define off 0
#define lparen 40
#define rparen 41

FILE *instream;
unsigned int length;
unsigned int i;
unsigned char ch,depth;
unsigned char buffer[16384];
union REGS regs;

main(argc,argv)
int argc;
char *argv[];
{
 if (argc <2)
 {
  printf("Not enough parameters on command line\n");
  exit(0);
 }
 if ((instream = fopen(argv[1],"r+b")) == NULL) printf("Could not open file for reading\n");
 else
{
 length = filelength(fileno(instream));
 depth = 0;
 for (i=0; i<length && ((ch = getc(instream)) != EOF); i++)
 {
  if (ch == lparen) depth++;
  if (depth != 0)
  {
   regs.h.bl = 0x0f;
  }
  else
  {
   regs.h.bl = 0x07;
  }
  regs.h.ah = 0x09; /* write attribute at cursor */
  regs.h.al = 0x20; /* display without highlight */
  regs.h.bh = 0x00;
  regs.x.cx = 0x0001;
  if (isprint(ch)) int86(0x10,&regs,&regs);

  regs.h.ah = 0x0e;  /* write tty   */
  regs.h.al = ch;
  regs.h.bh = 0x00;
  int86(0x10,&regs,&regs);

  if (ch == rparen) depth--;

 }
 fclose(instream);
 }
}
```

FIVE

The Bin2Hex Filter

The Bin2Hex filter is used to convert a binary image file into Intel Hexadecimal Format. This is a common format used by many prom programmers such as Data I/O, etc. Since this is a filter, the output may be directed to a file, an output port, or the screen.

```
;**********************************************************************************************
;
; Copyright (c) FOSCO 1988, 1989 - All Rights Reserved
;
;
; Module Name: Binary to Intel hexadecimal filter
;
; Version: 1.00
;
; Author: FOSCO
;
; Date: 1-5-89
;
; Filename: bin2hex.asm
;

; Functional Description:
;
;  This program converts the contents of a file to Intel hexadecimal format. Since many Prom
;  Programmers will accept an Intel Hex download format, .BIN files may be converted.
;  Filter outputs can be directed as shown in the examples below.
;
;
; Arguments:
;  bin2hex <infile.bin >outfile.hex
;  bin2hex <infile.bin >COM1
;  bin2hex <infile.bin >CON
;
; Return:
;
; Version History:
; 1-5-89 - font revision
;
;**********************************************************************************************
;

; P R O G R A M

          .model    compact

stack     segment word stack 'stack'
stack     ends

stdin     equ       0
stdout    equ       1

          .data
ibuff     db        16 dup(0)
obuff     db        32 dup(0)
line_count dw       0
char_count db       0
checksum db         0
ochar     db        0
eof       db        13,10,':00000001FF'

          .code
          assume    cs:_text,ds:_data

bin2hex proc
          mov       ax,_data
          mov       ds,ax
          mov       line_count,0
$$DO1:
            mov       bx,stdin                    ; get stdin
            mov       cx,16
            mov       dx,offset ibuff
            mov       ah,3fh
            int       21h
          JC $$EN1
```

```
            or          ax,ax                                   ; done
          JZ $$EN1
            mov         char_count,al
            call        write_line
          JMP SHORT $$DO1
$$EN1:
          call        write_eof             ; write eof
          mov         ax,4c00h
          int         21h                           ; return to dos
bin2hex endp

write_line proc
          mov         checksum,0
          mov         cl,char_count
          xor         ch,ch
          mov         al,13                         ; carriage return
          call        putc
          mov         al,10                         ; line feed
          call        putc
          mov         al,":"
          call        putc
          mov         al,char_count
          sub         checksum,al
          call        convert
          mov         ax,line_count
          mov         al,ah
          sub         checksum,al
          call        convert
          mov         ax,line_count
          sub         checksum,al
          call        convert
          mov         al,00
          sub         checksum,al
          call        convert
          add         line_count,16
          mov         si,offset ibuff
$$DO5:
            lodsb
            sub         checksum,al
            call        convert
          LOOP $$DO5
          mov         al,checksum
          call        convert
          ret
write_line endp

convert proc
          push        ax
          push        ax
          shr         al,1
          shr         al,1
          shr         al,1
          shr         al,1
          and         al,0fh
          add         al,'0'
          cmp         al,'9'
          JNA $$IF7
            add         al,7
$$IF7:
          call        putc
          pop         ax
          and         al,0fh
          add         al,'0'
          cmp         al,'9'
          JNA $$IF9
            add         al,7
$$IF9:
          call        putc
          pop         ax
          ret
convert endp

putc      proc
          push        ax
          push        bx
          push        cx
          push        dx
          mov         ochar,al
          mov         bx,stdout
          mov         cx,1
          mov         dx,offset ochar
          mov         ah,40h
          int         21h
          pop         dx
          pop         cx
          pop         bx
          pop       ax
          ret
putc      endp

write_eof proc
          push      ax
          push      bx
          push      cx
          push      dx
          mov       bx,stdout
          mov       cx,13
          mov       dx,offset eof
          mov       ah,40h
          int       21h
          pop       dx
          pop       cx
          pop       bx
          pop       ax
          ret
write_eof endp

          end       bin2hex
```

SIX

The Splitbin Program

The Splitbin Program is used to Split the at.bin file into an at.evn and at.odd file for programming the two Proms normally needed for an AT Bios.

Some of the newer Chipsets used in AT clones may allow you to use a 64k x 8 prom for the complete Bios. These Chipsets are based on the idea that the Bios would be moved to Shadow Ram which usually operates with fewer wait states than Prom, allowing higher performance operation. This may provide you with a design advantage if you are designing your own CPU board. Check with the VLSI chipset vendor for more information on this possible feature, and what additional software that might be required. If your AT motherboard has this feature, then the Splitbin won't be needed.

The Splitbin command line argument is the filename without the extension ("at" not "at.bin"). Splitbin adds the ".bin" to the input filename argument and the ".evn" and ".odd" extensions to the output files.

```
/****************************************************************************
*
* Copyright (c) FOSCO 1989 - All Rights Reserved
*
* Module Name: Bin file splitter to even/odd
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 10-1-89
*
* Filename: Splitbin.c
*
* Language: MS C 5.1
*
* Functional Description:
* This program reads the .bin file specified by the argument,
* and splits it into an .evn and a .odd file.
*
*
* Version History:
*
****************************************************************************/

/* I N C L U D E   F I L E S */

#include <stdio.h>


/* G L O B A L   V A R I A B L E S */

#define blen 8192
FILE *instream,*evenstream,*oddstream;
unsigned char inbuffer[2*blen], evenbuffer[blen], oddbuffer[blen];
unsigned char inname[16], oddname[16],evenname[16];
unsigned numread,i,j;


/* P R O G R A M */

unsigned main(int argc, char *argv[])
{
 printf("\n\rFOSCO/Annabooks Split xx.bin to xx.evn and xx.odd files Utility\n\r");
 if(argc == 2)
 {
  strcpy(inname,argv[1]); strcat(inname,".bin");
  strcpy(evenname,argv[1]); strcat(evenname,".evn");
  strcpy(oddname,argv[1]); strcat(oddname,".odd");
  if((instream = fopen(inname,"rb")) != NULL)
  if((evenstream = fopen(evenname,"wb")) != NULL)
  if((oddstream = fopen(oddname,"wb")) != NULL)
  {
   while(
         (numread = fread((char *)inbuffer,sizeof(char),2*blen,instream))
         != NULL)

   {
    for(i=0,j=0; j < numread; i++,j+=2)
    {
     evenbuffer[i] = inbuffer[j];
     oddbuffer[i] =  inbuffer[j+1];
    }
    fwrite((char *)evenbuffer,sizeof(char),numread/2,evenstream);
    fwrite((char *)oddbuffer,sizeof(char),numread/2,oddstream);
   }
  }
 }
 else
 {
  printf("No .bin file specified in command line\n\r");
 }
 return(0);
}
```

SEVEN

The Mergebin Program

The Mergebin Program is the companion to the Splitbin Program and may be used to combine ".evn" and ".odd" files to a ".bin" file. The command line argument is the filename (without extension). Mergebin looks for the ".evn" and ".odd" files and creates a ".bin" file. Mergebin is handy if you want to read existing Bios proms and create a complete image, as well as providing a means to verify the even/odd conversion process (Splitbin) by merging a split and comparing results.

```
/***************************************************************************
*
* Copyright (c) FOSCO 1989 - All Rights Reserved
*
* Module Name: Merge evn/odd files to a bin file
*
* Version: 1.00
*
* Author: FOSCO
*
* Date: 10-1-89
*
* Filename: Mergebin.c
*
* Language: MS C 5.1
*
* Functional Description:
* This program merges the .evn and .odd file into a combined .bin file.
*
* Version History:
*
***************************************************************************/

/* I N C L U D E   F I L E S */

#include <stdio.h>

/* G L O B A L   V A R I A B L E S */

#define blen 8192
FILE *outstream,*evenstream,*oddstream;
unsigned char outbuffer[2*blen], evenbuffer[blen], oddbuffer[blen];
unsigned char outname[16], oddname[16],evenname[16];
unsigned numread,i,j;

/* P R O G R A M */

unsigned main(int argc, char *argv[])
{
 printf("\n\rFOSCO/Annabooks Merge xx.evn and xx.odd to xx.bin files Utility\n\r");
 if(argc == 2)
 {
  strcpy(outname,argv[1]); strcat(outname,".bin");
  strcpy(evenname,argv[1]); strcat(evenname,".evn");
  strcpy(oddname,argv[1]); strcat(oddname,".odd");
  if((outstream = fopen(outname,"wb")) != NULL)
  if((evenstream = fopen(evenname,"rb")) != NULL)
  if((oddstream = fopen(oddname,"rb")) != NULL)
  {
   while(
   ((numread = fread((char *)evenbuffer,sizeof(char),blen,evenstream)) != NULL) &&
   ((numread = fread((char *)oddbuffer,sizeof(char),blen,oddstream)) != NULL)
   )
   {
    for(i=0,j=0; i < numread; i++,j+=2)
    {
     outbuffer[j] = evenbuffer[i];
     outbuffer[j+1] = oddbuffer[i];
    }
    fwrite((char *)outbuffer,sizeof(char),numread*2,outstream);
   }
  }
 }
 else
 {
  printf("No .bin file specified in command line\n\r");
 }
 return(0);
}
```

EIGHT

The Patch.asm File

If you have a problem with the initial installation of the generic version of the AT BiosKit, this chapter may be of interest to you. Normally the generic Bios image "at.bin" that is supplied on the distribution diskette will run on almost all AT system boards, so that is where most readers start. If you are planning to make modifications of the BiosKit, read this chapter anyway, as it will give you some ideas on how to speed up checkout (also refer to the Chapter on "Loading Static Ram") by eliminating the Prom Erase/Program cycle for your Bios Under Test. The contents of the patch.asm program are a normal feature of the BiosKit, so the material in this Chapter may help you to appreciate the BiosKit methodology.

This patch procedure may be useful to you if you are modifying the standard AT Bioskit to you particular hardware configuration and you have access to a previously configured Bios from another source. Its purpose is to provide a method to sense the existence of a "test" copy of Bios in Ram, and transfer control to the "test" copy. This patch will permit control to the transferred before any significant checks or initialization is done by the other Bios.

This patch program was created during the early days of testing the AT Bios in static ram. We were using a system with "another" Bios chip, and loading and running the development Bios in Ram as described in Chapter F-1. The typical AT Bios has the restart routines locked into the segment at F000, because they do not check for rom-scan or secondary Bios until well into the Bios. In order to wrest control away from the primary Bios residing at segment F000 immediately upon a hardware reset signal, it was necessary to intercept the control flow at the very beginning of the code. To provide for this, the patch sequence in this file was created. It is intended to be loaded into a modified "other" Bios.

The typical Bios will have a reset control flow similar to the example shown below:

```
F000:FFF0 Jmp Far  F000:E05B

F000:E05B Jmp Near - reset_code

F000:reset_code CLI                     ; disable interrupts
                   mov    al,8F             ; disable NMI's
                   Out    70,al             ; write to CMOS
                   xxx                      ; and so on
                   xxx
```

The patch is intended to be patched in between the jump at F000:E05B and the reset_code, so it looks like the example shown below:

```
F000:FFF0 Jmp Far  F000:E05B

F000:E05B Jmp Near - patch

F000:patch      ...                         ; patch program in this file
                   ...                      ; checks for test Bios and
```

```
                    ...                          ; transfers control if valid
                    ...
                    jmp     reset_code


F000:reset_code CLI                      ; disable interrupts
                    mov     al,8F                ; disable NMI's
                    Out     70,al                ; write to CMOS
                    xxx                          ; and so on
                    xxx
```

To install the patch, make a binary image copy of the "other" Bios using Debug. This can be done by the following
sequence:

```
Debug <enter>                   (call up debug)
nother.bin <enter>                          (name the file to be created)
rcs       <enter>                           (examine the CS register)
xxxx                                        (displays current value)
rcs xxyx <enter>                            (xxyx = xxxx + 10)
m f000:0000 l 4000 cs:0000 <enter>
m f000:4000 l 4000 cs:4000 <enter>
m f000:8000 l 4000 cs:8000 <enter>
m f000:c000 l 4000 cs:c000 <enter>
                                            (this copies the bios to ram)
rbx 1 <enter>                   (this sets the length of the file)
rcx 0 <enter>
w cs:0 <enter>                  (this writes the file on disk)
q <enter>                                   (this exits debug)
```

Now that you have an image of the "other" bios on disk, you may reload it (using Debug), and find an unused area large enough to accommodate the patch program. When a space has been found, the patch program may be inserted by using the "asm" command and loading the code. Replace the old jump at E05B with a jump to the patch, and use that original destination for the jump out of the patch.

After carefully checking that the patch is entered and linked correctly, use the "w" command to rewrite the file to disk.

The modified file will need its checksum corrected. The Biossum program may be used to calculate a new checksum. First though, you need to check the size of the bios. Remember, we wrote a full 64k file for the binary file. Reload it using debug and determine where code starts. If it is a typical assembly language version Bios, the code will start at location 8000. Fill the area from 0000 to 7FFF with "FF" using the debug "fill" command. The Biossum program will skip over "ff" bytes, until it finds a non-"ff" byte and assume that checksumming should then begin. After you have performed this fill (if required), then calculate a new checksum with the following command:

```
Biossum other.bin <enter>
```

Finally, if two proms are required (even and odd bytes), run "Evenodd other" to produce the two binary images required. Program a set of modified proms (we strongly recommend you retain the original unaltered proms) and install and test them in you machine.

This modified bios will then adhere to the standard Annabooks conventions for secondary and test Bios', and you may proceed with developing and testing your personalized version of the AT BiosKit.

The patch file below is intended to provide you with a guide to creating the patching code you will need, not necessarily an executable .EXE file. By assembling file with a "MASM patch,,; <enter> " command, the listing file generated will illustrate the object code required. The actual patching is most easily done by using the "asm" command in the DOS Debug as described above. Therefore this module is provided for reference only.

```
;*************************************************************
;
; Copyright (c) FOSCO 1988 - All Rights Reserved
;
;
; Module Name: Patch for "other" bios' for system checkout
;
; Version: 1.00
;
; Author: FOSCO
;
; Date: 12-01-88
;
; Filename: patch.asm
;
; Language MS MASM 5.1
;
; Functional Description:
;
;         This patch is intended to be patched into a non
;         Annabooks Bios to allow the secondary bios feature to
;         be used. The Annabooks Bios under development may then
;         be loaded into and executed from High-Ram in the D000
;         segment. The order of priority for selection
;         is:
;         1. D000 segment - test (usually Ram, may be Prom)
;         2. F000 segment - primary (Prom)
;
; Version History:
;
;
;*************************************************************

          .model    small
          .code

;----------- check for bios' patched in ----------
;
; If we find a Bios patch (by checking its signature),
; then we go to it.
;

; patch a jump at F000:E05B to jump to "patch"

patch:
          mov       ax,0d000h ; check seg d000
          mov       ds,ax
          xor       si,si     ; check for bios signature
          cmp       word ptr ds:[si],05b1h
          jne       check2    ; jumps if bad signature
                              ;then check for length count
          cmp       byte ptr ds:[si+2],80h
          jne       check2    ; jump if bad length count
          xor       cx,cx     ; set 64k count
          xor       al,al     ; clear checksum register
check1:
          add       al,ds:[si] ; build checksum for 64K bytes
          inc       si
          loop      check1
          or        al,al     ; zero = good
          jne       check2    ; jumps if bad checksum
          db        0eah              ; jump to bios at d000
          dw        0fff0h
          dw        0d000h
```

```
check2:
          jmp       1234h      ; jump to E05B's original destination
                                          ; you must determine this location !

          end
```