ADVANCED LEVEL

COMPUTING (0643/3) & (0982/2) *

** delete as required*

PROJECT LOG SHEET

JUNE EXAMINATION 199....

Centre Name: TAMESIDE COLLEGE OF TECHNOLOGY Centre No: 33511

Candidate's Name: NEIL HALLIDAY Candidate's No: .....

A. Project Proposal *(To be completed by the candidate and returned to the project supervisor before the start of the project).*

PROJECT TITLE: DiskBoot

OUTLINE DESCRIPTION:

A bootsector administration package, which allows the user to protect his/her disk collection from virus infection. The program will also allow the user to mend damaged disks that the program recognises, and if needed, Printed output of the bootsector will be given for future references.

Please indicate which of the following categories apply to this project – tick one box

1. Wholly programmed based ☑ 2. Partly programmed partly package based ☐ 3. Wholly package based ☐

Proposed computer system: ATARI STE

Poposed implementation language and / or proposed package(s) to be used: STOS BASIC & GEN ST 68000 ASSEMBLER

Candidate's Signature: [redacted] Date: 14.12.92

B. Project Approval *(To be approved by the Supervisor)*

I approve this proposal as being a suitable A level project for the above candidate.

Supervisor's Signature: [redacted] Date: 15/12/92

NOTE: A candidate should be advised only to undertake a project that is expected to be within the candidate's capability to complete on time.

This form should be kept with the project work, attached to the front of the report.

P.T.O.

Definition Of Problem.

Most of my friends, people I am in contact with, and myself own an ATARI ST home computer. The ATARI ST is a diskette based machine, that relies heavily on a diskette being placed in the internal 3.5" disk drive most, if not all of the time.

Each time a new diskette is placed into the disk drive, the computer needs to locate all the existing files that are stored on it. The data needed by the computer is located on part of the diskette called the *File Allocation Table* or FAT table. This table contains the following :-

File name.

This is the name of the file that will be displayed by doing a simple DIR command from inside a program. It is made up of eleven characters, and a full-stop that is always located between the eighth and ninth characters. The first eight letters make up the file identifier, and the last three characters make up the file extension, which usually signifies what type the file is, e.g. a file with a ".PRG" extension is most commonly an executable PROGram file, and a filename with a .INF extension is a usually an INFormation file.

File size.

The overall length of the file in bytes.

File Fragmentation.

When a file gets written to a diskette the computer may come across sections of it that has already had information written to it. This means that overwriting this data would be destroying another file, to get around this the computer splits the file up, and spreads it around the diskette. This part of the FAT table holds the location on the diskette that each section of the file is stored.

Every FAT table on a diskette holds different information than any other FAT table, so a FAT table containing information on twelve files will be shorter than a FAT table containing information on sixteen files. Luckily the beginning of all FAT tables are started in the same location on EVERY diskette, the only problem is that the computer does not know the length of each FAT table, and how many FAT tables are on the diskette. Because of this variety of information, the number and length of FATs must be made readily available to the computer on a separate part of the diskette.

The Bootsector.

At the beginning of every diskette is a single sector (512 bytes) of data/program code known as a *BOOTSECTOR* or *BOOTBLOCK*. This bootsector is where the information the computer needs about the FAT table is stored. So to read the diskette's FAT table the computer must first read the diskette's bootsector/bootblock, and this is where our problem occurs.

The bootsector/bootblock is located on **Side 0, Track 0, Sector 1** of all ATARI ST format diskettes, and it's layout is as follows :-

Layout Of A Bootsector.

Address offset	Information
0 - 1	Branch instruction to boot program if exec
2 - 7	'Loader'
8 - 10	24-bit serial number
11 - 12	Bytes Per Sector
13	Sectors Per Cluster
14 - 15	Number of REServed disk sectors
16	Number of FATs on the diskette
17 - 18	Maximum number of directory entries allowed
19 - 20	Total number of sectors on the diskette
21	Media descriptor byte, not used on ATARI ST
22 - 23	Number of sectors in each FAT
24 - 25	Number of sectors per track
26 - 27	Number of sides on the diskette
28 - 29	Number of hidden sectors on the diskette
30 - 509	User programmable space
510 - 511	Checksum

The Problem.

As mentioned above, the bootsector is where our problem lies. You will notice that bytes 30 - 509 on the bootsector is "User programmable space". This means that the user of the computer can create programs using a 68000 assembler language, and save them onto the bootsector. In effect, this creates a tiny program that is executed whenever the diskette is read from, or written to, for any reason. Having this option on each diskette, opens a new way of transmitting viruses around the computer field. It's these viruses that cause our problem.

What is a Virus?

A virus is a program that once installed can destroy information in the computer's memory, or information stored on all other types of read/write media. Their only use is to multiply onto other diskettes. They are mainly tiny programs no longer than 500 bytes long, which come in two forms..

A) Link Viruses.
B) Bootsector/Bootblock Viruses.

But which ever form they come in, they are usually very dangerous to have on your computer system!

Link Viruses.

This type of virus can be located at the beginning, middle or end of any executable file that you have on your diskette. They attach themselves in one of the said places, and then re-create the executable front end of the file to run the link virus before any other part of the infected code. These type of viruses usually copy themselves into memory, and when an uninfected executable file is loaded and run, in the computer's memory, the virus will copy itself to the file, alter the code, and then re-save it to disk. This operation usually happens very fast, so the user is unable to tell what is happening until too late.

This type of virus is VERY uncommon on an ATARI ST.

Bootsector/Bootblock Viruses.

This is the main type of virus on the ATARI ST. They are located in the user programmable bytes of the diskette's bootsector, and when read will be copied into memory. These types of viruses also tend to install themselves onto the computers reset vector, which means it will stay in memory when the computer has been warm booted (by pressing the reset button, but not actually turning the computer off). These viruses work in exactly same way as the link viruses, but instead of attaching themselves to a file, they will re-create an infected bootsector on any diskette placed into the drive that is write enabled (so you can save data onto it).

The bootsector virus has a counter stored into it's code, and every time the virus is spread to a new diskette the counter is incremented. After a certain number of copies the virus will reset it's counter, and while still in memory, get up to some nasty tricks like deleting files and even formatting your hard disk!

Establishment of Objectives.

To get rid of these viruses, we need to find some way of deleting the virus code on the bootsector, without deleting the valuable information needed to keep the diskette working. Simply destroying the code in the bootsector would erase the information the computer needs to read the FAT table, meaning that our disk would be useless, until reformatted. We really need a program that will destroy the virus code, and recreate the disks information without damaging the data in anyway.

Analysis of Problem.

Our problem is one of the most common on home computers to date, nothing can really be done to stop the problem, although precautions can be taken. The fact is that the viruses themselves only plays a small part in the trouble. There are four main points about about how a virus is spread, they are :-

A) The person who actually wrote the virus.

B) The person who wrote the virus, spreading it for the first time.

C) People who find it funny to spread viruses.

D) People who spread viruses unintentionally, not knowing they are on their disks.

If it wasn't for the above reasons hardly any viruses would be around. Possibly the only place they would exists is in backup copies that were created a long time ago, and were stored in a safe or secure area.

Identification Of Possible Solutions.

Our virus problems can be sorted out in a number of ways, in order to find out the best way, I will specify in detail what each separate solution is capable of, and the advantages/disadvantages of them in operation. I will then compare each solution with one another and give them an overall mark out of 10 in a variety of areas, e.g. cost, ease of use, etc.

Solution 1 - Keeping Your Disks Write Protected.

This solution will be by far the simplest, but as simple as it is, this will probably be the most effective method.

Viruses spread themselves around various diskettes by copying/saving over the existing data on a bootsector. The best way to avoid this is by making it physically impossible to write any data onto the diskette. You can do this by moving the black write protection tab up, so you can see through the hole in the diskette. Now, every time the viruses try to copy information onto a diskette, they can't do.

Advantages.

Costs no money at all.

Can be done on any diskette.

Total virus protection in all accounts.

Disadvantages.

Can't save any data to the diskette without write enabling it, this them means your diskette could be infected by a virus while saving your data.

Bit of a pain having to take your diskette in and out of the drive to write enable/write protect it.

Solution 2 - Regular Disk Checking.

Using this solution you can leave your diskettes write enabled, and check it every so often, e.g. once a week. A standard disk editing software package could be used to load the bootsector into memory, where the data contained in the bootsector can be deleted by hand, if needed.

Advantages.

Only have to buy a disk editor, if you haven't got one.

Can be utilised on any of your diskettes.

Your diskette can be write enabled all the time.

Reasonable protection against virus infection.

Disadvantages.

Can be costly to buy a disk editor, around £25.00 in some cases.

A lot of time has to be spent checking all your diskette's.

Possibility of erasing valuable information from the diskette.

May destroy bootsector information that is not a virus, e.g. An auto booting commercial software package.

Solution 3 - Commercial Package.

This solution will meet all the requirements for our problem. Software is available that will destroy the virus information, keeping the diskette's information in tact. These commercial packages consist of disk editors, virus killers, and diskette management software, including features such as the following :-

Information on common ATARI ST viruses.
Ability to re-create damaged diskettes.
Formatting ability incase disk is too damaged.
Print bootsector information.

Many of the commercial virus killers go over board with the features that they include, and are almost always updated within a couple of weeks after the consumer has bought the product. Not only is it the program that goes over board with the functions, but the prices of these programs can range between £9.00 to around £99.00.

Advantages.

You will be getting a quality package.

Can be help full if you want to do things that a normal virus killer can not cope with, e.g. external DOS/GEM commands.

Guaranteed that the software should work.

Regular software updates.

Disadvantages.

Gets very expensive when the software goes above £15.00 to buy.

Most of the time you have to pay for the software updates.

Too many functions if you only want the software to do one thing.

Solution 4 - Specially Written Software.

This solution solves all the problems, it is a specially written program, that can be commissioned to somebody to write, or can be written by myself. This type of software will allow my every virus problem to be full filled. This software can be updated as often as I would like, although if it was commissioned to somebody, I might have to pay for the updates. When new ideas come, or new viruses arise the package can be updated to recognise them, and will be helpful almost straight away.

I can have the resulting program personalised, for my own use, or if it were good enough, maybe even published. The program would be useful to all other ATARI ST users that have the same trouble with viruses as myself.

Advantages.

Update the software whenever necessary.

Have any function I want in the program.

Could be helpful to other users.

Make the program as easy or as hard to understand as I like.

Disadvantages.

Very time consuming.

Program may not work when finished.

Very costly if someone is commissioned to write it for me.

Scoring.

I shall now evaluate the above solutions to my virus problem, giving marks out of 10 for the different areas that the programs can handle. The score 10 shall be the best, down to 1 which shall be the worst.

	Sol. 1	Sol. 2	Sol. 3	Sol. 4
User interface	0	0	10	10
Ease of use	10	5	8	10
Cost	10	7	1	10
Performance	8	3	10	9
Speed	5	1	9	7
Totals :	33	16	38	46

Proposed Solution.

From the above information I have chosen solution 4, as it scores the highest in almost all of the categorised areas. I have chosen to write the system myself, as this works out much cheaper than employing a freelance programmer to program it. The program will be written for the ATARI ST range of computers, and will work on all different types of ST machines that are available in the shops. The following is a system specification to be used during the development if the program.

ATARI 520ST^{E}
2 Megabytes of RAM
720 Kilobyte Disk drive
External real-time clock unit
Citizen Swift 24, 24 pin colour printer
16 Colour display

Questionnaires.

To help in the design of the program, I have written a questionnaire, so existing users of virus killers and people that are suffering from the same problem as myself can give their view of how a virus killing package should be set up. From the results of these questionnaires I will gather together various points, and adapt them into one final program.

The result of the questionnaires can be found in appendix A.

Languages To Be Used.

I have decided to write my program using the STOS Basic language. This software will allow me to create impressive user interfaces using the minimal yet efficient coding routines, as the language is written specifically for the interfacing of graphics.

STOS basic also allows the use of basic extensions, which are small programs that are located in the main STOS program directory, and are automatically loaded when STOS basic is executed. These extensions are written in 68000 assembler, and allow the user to create his/her commands (with full syntax checking) to be placed into the language. For example if the user wanted to format a diskette (which is not currently possible in STOS basic), he/she can create an assembler program using the standard ATARI GEM trap commands, and interface that into the language as a command called "format". If he/she would then want to use the extension commands they would just use her new command as with any normal STOS basic command. e.g.

```
10 format
```

Commands can have parameters sent to them, so instead of just having the word format for a command, you can have something like this

```
10 format 0,1,1,0
```

When writing commands like this, the user has to incorporate syntax checking and validation routines into the extension assembly code. Which means if the command is incorrectly called from within STOS basic, say if a parameter is missed out, or a letter is passed instead of a number, the extension will detect this and display an error message.

If the program is to be compiled then a new version of the extension code is needed for the compiler. This has to be laid out in a different order than the interpreter extension, because STOS uses "Smart Compiling". This means it only copies the code it needs into the final compilation, meaning no redundant code will be present.

My Solution.

After looking through the response that I got from the questionnaires I have decide on the final structure of the program. I have compiled a list of operations that my final virus killer must do. These are:-

Identify A Bootsector/Virus.

This function will test the contents of a bootsector, or check memory to see if a virus is present.

Destroy Bootsector/Virus.

This function will delete all information on the diskettes bootsector, or delete the virus information stored in memory.

Examine Bootsector.

This function will allow the user to print all the information contained within a bootsector to screen or printer, in Hexadecimal or ASCII displays.

Install/Mend Bootsector.

This function will allow the user to install a bootsector stored in a data file onto any diskette, If the mend function is selected then a direct copy of the original bootsector will be dumped to the diskette.

A point that I found could not be missed from a program is a help function, so I also want to include an online help help message that explains what each function does. I also think that if an experienced user is running the software, they should be given the option to turn off the help. As the ATARI ST is built with a "HELP" key, I think it wise to use this to the full capability.

STOS basic also has an error file created, which is in the Public Domain, meaning it can be used freely by anybody. The file is a database of all the errors that can occur in the language, and can be accessed by the error number. I will use this file to give a description of all the errors that happen during the running of the program, this will be a great help to new user to the system, as it prints a written description of the error, can how it occurred.

Analysis of Data Requirements.

In my proposed solution the virus killer will be able to recognise different types of bootsectors. To do this my program will have to keep a data file containing all the bootsectors, for comparison purposes in the identify bootsector/virus function. With the growing number of different bootsectors and viruses, it would be difficult to store them all in one big file, another problem with using one big file is updating the number of bootsectors. Each bootsector on the ATARI ST is 512 bytes in length, so having around 200 bootsectors in memory on a half megabyte machine would be using valuable memory that the computer is likely to need. I therefore think it wiser to store all the bootsectors as separate 512 bytes files, and have another information file, containing the filenames and bootsector descriptions of all the bootsector files on disk. Using this method a half megabyte machine can load only a few of these bootsector files, this limits the amount of bootsectors/viruses that the program can then recognise, but this problem cannot be avoided.

With the requirements of my solution a STOS extension will have to be created, because the program will also have to be compiled I will also have to write a STOS Compiler extension. I have created a list of commands that I think I may need to write for the program of my solution.

Comparison Command.

When I identify a bootsector I will not need to know the contents of it, so this new command will return a value (true or false) if the values passed to it are the same or not.

Testing Reset Vector.

This command will return if the reset vector on the computer (a major storing place for ATARI ST viruses) has been changed. If it has been changed then a virus is in memory.

Working Out A Percentage.

As mentioned before, the bootsector files will be stored in separate 512 bytes files. While the program is loading these, the user will want to know what is going on. With command I can print the percentage of files loaded.

Producing A Bootsector.

At the moment STOS basic will not allow us to construct a standard ATARI GEM bootsector, adding this command will change this.

Data.

To help me with the structure of the data, I will create file structure charts, which will contain varied information concerning the two different types of file that my program will need. The error file is already created, as it is part of the STOS basic language.

User Interface.

As the user will be interacting with the final program, I will need some way of designing a user interface. To help me with this I am going to draw rough screen layouts that the program should stick to. Also the user must sometimes choose what option he wants from the menu selection he has already made, e.g. option 4 is install/mend a disk, once this has been selected the user must then choose install bootsector or mend an existing bootsector. This system I propose to include is called "Alert Boxes". These alert boxes can be used to alert the user to errors, and can also be used to ask the user to input a required option. Most alert boxes work of a point and click method, where the user selects the option by placing the mouse pointer over the text and pressing a mouse key.

Output Data.

The examine bootsector option calls for printing the bootsector to the printer. Again to help me with the design of the output I shall be using a print specification form.

Programming.

To help me with the main points of the programming, I am going to draw structure diagrams. These will then help with any debugging that I need to do after the programming is fully finished.

The design of the above requirements can be located in appendix B

Testing Strategy.

To fully test my program, I am going to create a testing strategy that will cover everything possible that could happen during the running of the application. With using STOS basic, I can create errors by using a built in command, this will help me test my program in most situations. Any error output that is given by the program will be displayed by the error screen, which will contain an error number, and a written description of how the error will have occurred. The points that I will test my program on can be found in appendix C.

```
* Disk Boot V1.0 - STOS Basic Interpreter Extension
* Written By Neil Halliday 1993
* Using Devpac Assembler V2.0 By Hisoft
*

* This extension adds the following commands to STOS Basic :-
*
* produce boot   : Creates a standard GEM bootsector
* perc           : Returns the percentage that value 1 is of value 2
* resvector      : Returns if the reset vector is valid or not
* compare        : Returns if the word boundary values at A & B are the
*                  same


* Extension Header

        bra     init                            ; Branch to init
        dc.b    128                             ; Start token number

* Token List

tokens  dc.b    "produce boot",128              ; Produce bootsector
        dc.b    "perc",129                      ; Percent
        dc.b    "resvector",131                 ; Resvector
        dc.b    "compare",133                   ; Compare
        dc.b    0                               ; End of instruction
                                                ; list

        even                                    ; Even all addresses

* Jump table to locate address of routines

jumps   dc.w    6                               ; Number of extension
                                                ; commands
        dc.l    produce_bootsector              ; Produce bootsector
        dc.l    perc                            ; Perc
        dc.l    dummy                           ; Dummy routine
        dc.l    reset_vector                    ; Resvector
        dc.l    dummy                           ; Dummy routine
        dc.l    compare                         ; Compare


* Welcome/bootup message in 2 languages

welcome dc.b    10,"DiskBoot V1.0 Extension",0  ; English
        dc.b    10,"Extension DiskBoot V1.0",0  ; French
        dc.b    0
        even

* Reserve space to hold address of Basic variables

system  dc.l    0                               ; Space for calling
                                                ; system
return  dc.l    0                               ; Memory address of
                                                ; STOS
```

```
* Initialization routines extecuted when STOS is ran

init    lea     exit,a0                 ; End of the extension
        lea     coldst,a1               ; Address of coldstart
        rts

* Coldstart routine

coldst  move.l  a0,system               ; Copy address of
                                        ; variables
                                        ; into a0

        lea     welcome,a0              ; Address of welcome
                                        ; message
        lea     warmst,a1               ; Address of warmstart
        lea      tokens,a2                ; Address of token
list
        lea      jumps,a3                 ; Address of jump
table
        rts

* Warmstart routine

warmst  rts                             ; Does nothing


* Support routines for passing parameters to and from Basic

* Get an integer argument

getint  move.l  (a7)+,a0                ; Save return address
        movem.l (a7)+,d2-d4             ; Get parameter
        tst.b   d2                      ; Is the argument a
                                        ; number?
        bne     typemis                 ; No!
        jmp     (a0)                    ; Jump to return
                                        ; address


* STOS errors, for user defined error checking

syntax  moveq   #12,d0                  ; Error 12 (syntax)
        bra.s   error                   ; Branch to error rout

typemis moveq   #19,d0                  ; Error 19 (type
                                        ; missmatch)

error   move.l  system(pc),a0           ; Get address of basic
                                        ; routines
        move.l  $14(a0),a0              ; Get address of error
                                        ; routine
        jmp     (a0)                    ; Jump to error
                                        ; handler
```

```
***** START OF MY OWN ROUTINES/COMMANDS *****

* Dummy routine, used to split seperate commands and functions

dummy   rts

* Produce boot

produce_bootsector

        move.l  (a7)+,return            ; Save return value
        cmp     #4,d0                   ; 4 parameters passed?
        bne     syntax                  ; No, create syntax
                                        ; error

        bsr     getint                  ; Buffer
        move.l  d3,a1                   ; Save in address
                                        ; reg 1

        bsr     getint                  ; 24bit Serial number
        move.l  d3,a0                   ; Save in address
                                        ; reg 0

        bsr     getint                  ; Disktype
        move.w  d3,d5                   ; Save in data reg 5

        bsr     getint                  ; Execflag
        move.w  d3,d4                   ; Save in data reg 4

        move.w  d4,-(sp)                ; Execflag
        move.w  d5,-(sp)                ; Disktype
        move.l  a0,-(sp)                ; Serial number
        move.l  a1,-(sp)                ; Buffer
        move.w  #18,-(sp)               ; Protobt
        trap    #14                     ; Call XBIOS
        add.l   #14,sp                  ; Restore stackpoint

        move.l  return,a0               ; Load return address
        jmp     (a0)                    ; Jump to it
```

```
* Perc

perc    move.l  (a7)+,return        ; Save return
        cmp     #2,d0               ; 2 parameters?
        bne     syntax              ; No, syntax error

        bsr     getint              ; Get val2
        move.l  d3,d1               ; Store

        bsr     getint              ; Get val1
        move.l  d3,d0               ; Store

        move.l  #0,d4               ; Zero d4
        move.l  #99,d3              ; Loop value

add_loop
        add.l   d0,d4               ; Add val1 to d4
        dbra    d3,add_loop         ; loop

        move.l  #1,d6               ; One in d6
        move.l  #0,d5               ; Zero d5
        bra     divide_loop         ; Branch to divide
                                    ; loop

divide_loop_2

        add.l   #1,d6               ; Increase Counter

divide_loop

        add.l   d1,d5               ; Add val 1
        cmp     d4,d5               ; Compare d4 & d5
        blo     divide_loop_2       ; Loop if d4 is lower

        move.l  d6,d3               ; Store in return
                                    ; value pos
        move.l  #0,d2               ; Expect integer

        move.l  return,a0           ; Load return
        jmp     (a0)                ; Jump to it

* Resvector

reset_vector

        move.l  (a7),return         ; Save return

        cmp.l   #$31415926,$426     ; Resvalid?
        bne     no_virus            ; No, not been changed

        move.l  #0,d3               ; Move false
        bra.s   leave_reset_vector  ; Leave routine
```

```
no_virus

        move.l  #-1,d3                  ; Move true

leave_reset_vector

        move.l  #0,d2                   ; Expect an integer
        move.l  return,a0               ; Load return addr
        jmp     (a0)                    ; Jump to it

* Compare

compare

        move.l  (a7)+,return            ; Save return
        cmp     #2,d0                   ; 2 parameters?
        bne     syntax                  ; No!

        bsr     getint                  ; Get address 2
        move.l  d3,a1                   ; Store as address

        bsr     getint                  ; Get address 1
        move.l  d3,a0                   ; Store as address

        move.w  (a0),d4                 ; Store contents
        move.w  (a1),d5                 ; Store contents

        cmp.w   d4,d5                   ; Compare values
        bne     not_same                ; Not the same

        move.l  #-1,d3                  ; True value
        bra.s   leave_compare           ; leave routine

not_same

        move.l  #0,d3                   ; False value

leave_compare

        move.l  #0,d2                   ; Expeact an integer
        move.l  return,a0               ; Load return
        jmp     (a0)                    ; Jump to it

* End of extension marker

exit    equ     *
```

```
* Disk Boot V1.0 - STOS Basic Compiler Extension
* Written By Neil Halliday 1993
* Using Devpac Assembler V2.0 By Hisoft
*

* This extension adds the following commands to STOS Basic :-
*
* produce boot  : Creates a standard GEM bootsector
* perc          : Returns the percentage that value 1 is of value 2
* resvector     : Returns if the reset vector is valid or not
* compare       : Returns if the word boundary values at A & B are
                : the same


* Set up system variables

debut   equ     $92c
error   equ     $93c
flagem  equ     $9a0

* Define extension addresses

start   dc.l    para-start                      ; Parameter
                                                ; definitions
        dc.l    data-start                      ; Reserve data areas
        dc.l    lib1-start                      ; Start of library


catalog dc.w    lib2-lib1                       ; Length of routine 1
        dc.w    lib3-lib2                       ; Length of routine 2
        dc.w    lib4-lib3                       ; Length of routine 3
        dc.w    lib5-lib4                       ; Length of routine 4
        dc.w    lib6-lib5                       ; Length of routine 5
        dc.w    libe-lib6                       ; Length of routine 6

para    dc.w    6                               ; Number of library
                                                ; routines
        dc.w    6                               ; Number of extension
                                                ;  commands
        dc.w    protb-para                      ;  Offset  for  param
sets
        dc.w    perce-para                      ;
        dc.w    dumm1-para                      ;
        dc.w    reset-para                      ;
        dc.w    dumm2-para                      ;
        dc.w    compa-para

* Parameter definitions

I       equ     0                               ; Value for integer
F       equ     $40                             ; Value for floating
                                                ; point
S       equ     $80                             ; Value for string
```

```
* "," forces a comma between any commands
* 1   indicates the end of one set of parameters
* 1,0 idicates the end of the commands parameters

protb   dc.b    0,I,",",I,",",I,",",I,1,1,0
perce   dc.b    I,I,",",I,1,1,0
dumm1   dc.b    0,1,1,0
reset   dc.b    I,1,1,0
dumm2   dc.b    0,1,1,0
compa   dc.b    1,I,",",I,1,1,0

* End of parameter definitions

        even                                    ; Even addresses

* Init section

data    bra     init

init    lea     end(pc),a2                      ; Return end of
                                                ; library
        rts

end     rts


***** START OF MY OWN ROUTINES/COMMANDS *****


* Produce boot

lib1    dc.w    0                               ; No library calls
        move.l  (a6)+,d0                        ; Get buffer
        move.l  (a6)+,d1                        ; Get serial number
        move.l  (a6)+,d5                        ; Disktype
        move.l  (a6)+,d4                        ; Execflag

        move.l  d0,a1                           ; Store as address
        move.l  d1,a0                           ; Store as address

        move.w  d4,-(sp)                        ; Execflag
        move.w  d5,-(sp)                        ; Disktype
        move.l  a0,-(sp)                        ; Serial number
        move.l  a1,-(sp)                        ; Buffer
        move.w  #18,-(sp)                       ; Protobt
        trap    #14                             ; Call XBIOS
        add.l   #14,sp                          ; Restore stackpoint
        rts                                     ; Return
```

```
* Perc

lib2    dc.w    0                       ; No library calls
        move.l  (a6)+,d1                ; Get val2
        move.l  (a6)+,d0                ; Get val1

        move.l  #0,d4                   ; Zero d4
        move.l  #99,d3                  ; Loop value

add_loop
        add.l   d0,d4                   ; Add val1 to d4
        dbra    d3,add_loop             ; loop

        move.l  #1,d6                   ; One in d6
        move.l  #0,d5                   ; Zero d5
        bra     divide_loop             ; Branch to divide
                                        ; loop

divide_loop_2

        add.l   #1,d6                   ; Increase Counter

divide_loop

        add.l   d1,d5                   ; Add val 1
        cmp     d4,d5                   ; Compare d4 & d5
        blo     divide_loop_2           ; Loop if d4 is lower

        move.l  d6,-(a6)                ; Store in return
                                        ; value pos
        rts

* Dummy routine

lib3    dc.w    0                       ; No library calls
        rts

* Resvector

lib4    dc.w    0                       ; No library calls

        cmp.l   #$31415926,$426         ; Resvalid?
        bne     no_virus                ; No, not been changed

        move.l  #0,d3                   ; Move false
        bra.s   leave_reset_vector      ; Leave routine

no_virus

        move.l  #-1,-(a6)               ; Move true

leave_reset_vector

        move.l  #0,-(a6)                ; Expect an integer
        rts
```

```
* Dummy routine 2

lib5    dc.w    0                       ; No library calls
        rts


* Compare

lib6    dc.w    0                       ; No library calls
        move.l  (a6)+,d0                ; Address 1
        move.l  (a6)+,d1                ; Address 2

        move.l  d0,a1
        move.l  d1,a0

        move.w  (a0),d4                 ; Store contents
        move.w  (a1),d5                 ; Store contents

        cmp.w   d4,d5                   ; Compare values
        bne     not_same                ; Not the same

        move.l  #-1,-(a6)               ; True value
        bra.s   leave_compare           ; leave routine

not_same

        move.l  #0,-(a6)                ; False value

leave_compare

        rts


* End of extension marker

libe    dc.w    0                       ; End of library
```

```
10 rem ************************************************************
20 rem  |* DISK BOOT V1.0 - WRITTEN BY NEIL HALLIDAY 1992/93 *|
30 rem  |*      WRITTEN WITH STOS BASIC ON THE ATARI ST       *|
40 rem ************************************************************
50  :
60 rem * SET UP THE SCREEN
70 key off : curs off : click off : hide : on error goto 3010
80 mode 1 : palette $777,$70,$700,$0
90 scroll off : dim MNU$(5),HLP$(5)
100 DF_DRV=drive : DF_PATH$="\"+dir$ : home : pen 3
110  :
120 rem * READ MENU DATA
130 restore 3510 : for LP=0 to 5 : read MNU$(LP) : next LP : for LP=0
to 5 : read HLP$(LP) : next LP
140  :
150  :
160 rem * FIND DRIVE CONTAINING DISKBOOT.DAT DIR, AND SET WORKING DIR
170 ND=ndrv : dec ND : CNT=-1 : X$="" : repeat : inc CNT : drive=CNT :
X$=dir  first$("\DISKBOOT.DAT",16)  :  until  val(mid$(X$,43,2))=16  or
CNT=ND : dir$="\DISKBOOT.DAT"
180  :
190 rem * OPEN INFORMATION FILE AND GRAB AMOUNT OF LIBRARY FILES
200 cls : home : pen 3 : print "Loading library files :   0%" : open
in #1,"DISKBOOT.INF" : input #1,NUMFILES
210 dim FILENAME$(NUMFILES),DESC$(NUMFILES)
220  for  LOOP=1  to  NUMFILES  :  input  #1,FILENAME$(LOOP)  :  input
#1,DESC$(LOOP) : next LOOP : DESC$(0)="UNKNOWN boot sector" : close #1
230  :
240 rem * LOAD 1ST LIBRARY FILE INTO BANK 15, APPEND OTHERS
250 reserve as work 15,NUMFILES*512 : MEM=start(15) : for LOOP=1 to
NUMFILES  :  bload  FILENAME$(LOOP),MEM  :  locate  24,0  :  print  using
"###";(perc(LOOP,NUMFILES));"%" : MEM=MEM+512 : next LOOP
260  :
270 rem * SET DEFAULT MENU VALUES
280 C_MNU=0 : OPTION=0 : HLP=1
290  :
300 rem ** MAIN DRAW SCREEN
310  :
320 rem ** INVERSED BORDERS
330  hide  :  cls  :  ink  3  :  bar  0,0  to  639,ygraphic(3)-1  :  bar
0,ygraphic(22) to 639,ygraphic(24)+7
340  :
350 rem ** PRINT TEXT
360 pen 3 : paper 0 : inverse on : locate 1,1 : print "DISK BOOT V1.0"
370 locate 0,23 : centre "Select options using numbers or arrow keys &
SPACE/RETURN"
380 locate 45,1 : print "Time :          Date : "+date$
390 inverse off
400  :
410 rem * MAIN MENU LOOP
420 if mode=1 then flash 2,"(700,30)(777,15)"
430 YPOS=7 : for LP=1 to 6 : locate 25,YPOS : print str$(LP)+".. " :
YPOS=YPOS+2 : next LP
440 gosub 650 : repeat : KY$=inkey$ : SC=scancode
450 if SC=98 then if HLP=1 then HLP=0 : pen 3 : shade off : inverse
off : locate 0,20 : print space$(80) : else HLP=1 : gosub 670
```

```
460 if SC=72 then dec C_MNU : OPTION=1 : if C_MNU<0 then C_MNU=5
470 if SC=80 then inc C_MNU : OPTION=1 : if C_MNU>5 then C_MNU=0
480 if OPTION>0 then gosub 650 : OPTION=0
490  :
500 rem * CHECK FOR KEY PRESS OR SPACE PRESS
510 if KY$="1" or (C_MNU=0 and (KY$=" " or KY$=chr$(13))) then gosub
710 : goto 690
520 if KY$="2" or (C_MNU=1 and (KY$=" " or KY$=chr$(13))) then gosub
1340 : goto 690
530 if KY$="3" or (C_MNU=2 and (KY$=" " or KY$=chr$(13))) then gosub
1590 : goto 690
540 if KY$="4" or (C_MNU=3 and (KY$=" " or KY$=chr$(13))) then gosub
2260 : goto 690
550 if KY$="5" or (C_MNU=4 and (KY$=" " or KY$=chr$(13))) then gosub
2760 : goto 690
560 if KY$="6" or (C_MNU=5 and (KY$=" " or KY$=chr$(13))) then gosub
2900 : goto 690
570  :
580 rem * PRINT TIME
590 shade off : inverse on : pen 3 : paper 0 : locate 52,1 : print
time$ : inverse off
600  :
610 rem * MAKE EVERYTHING RUN ON THE SCREEN REFRESH RATE
620 wait vbl : until false
630  :
640 rem * REDRAW MENU WITH NEWLY CHOSEN OPTION & HELP
650 YPOS=7 : for LP=0 to 5 : locate 24,YPOS : if LP=C_MNU then shade
off : pen 3 : print ">" : shade off : pen 2 : else print " " : shade
on : pen 3
660 locate 30,YPOS : print MNU$(LP) : YPOS=YPOS+2 : next LP : clear
key
670 if HLP=1 then pen 3 : shade off : inverse off : locate 0,20 :
print space$(80) : locate 0,20 : centre "Help : "+HLP$(C_MNU)
680 return
690 goto 300
700  :
710 rem * IDENTIFY BOOTSECTOR/VIRUS
720 flash off : colour 2,$700 : MSG$="What media do I check|for a
virus??" : BUT$="Disk|Memory" : TLT$=">> IDENTIFY BOOTSECTOR <<" :
NB=2 : gosub 3160
730 if ALERT=1 then goto 750 : else goto 1290
740  :
750 rem * IDENTIFY BOOTSECTOR/VIRUS ON DISK
760 MSG$="Which drive do I check??| " : BUT$=" A | B " : TLT$=">>
IDENTIFY BOOTSECTOR << " : NB=2 : gosub 3160
770 DRV=ALERT-1 : drive=DRV
780 erase 3 : reserve as work 3,512 : floprd start(3),1,0,0,1,DRV
790 gosub 2950 : home : wait 30
800  :
810 rem * MAKE SOME WORK SPACE ON SCREEN
820 cls logic,0,0,150 to 640,200
830  :
840 rem * PRINT SCREEN
850 pen 3 : locate 4,5 : under on : print "Disk Information" : under
off
```

```
860 pen 3 : locate 4,7 : print "Number of tracks :    " : locate 4,9 :
print "Number of sides  :    " : locate 44,7 : print "Sectors per track
:    " : locate 44,9 : print "Total number of sectors :     "
870 locate 4,11 : print "Volume label      :    " : locate 44,11 : print
"Disk space free          :   "
880  :
890 rem * GET DISK INFORMATION FROM MEMORY BANK
900 NAME$=dir first$("*.*",8) : if NAME$="" then NAME$="None" : else
NAME$=left$(NAME$,12)
910 MEM=start(3) : poke MEM,peek(MEM+20) : poke MEM+1,peek(MEM+19)
920  NUMSEC=deek(MEM)  :  SECPTRK=peek(MEM+24)  :  NSIDES=peek(MEM+26)  :
NTRAKS=(NUMSEC/NSIDES)/SECPTRK : on error goto 930 : DF=dfree/1024 :
dec NTRAKS
930  :
940 rem * PRINT INFO
950 pen 2 : locate 23,7 : print "0 -";NTRAKS : locate 22,9 : print
NSIDES : locate 23,11 : print NAME$
960 locate 69,7 : print SECPTRK : locate 69,9 : print NUMSEC : locate
69,11 : print DF;" k"
970  :
980 rem * LOAD BOOT SECTOR AGAIN
990 floprd start(3),1,0,0,1,DRV
1000  :
1010 rem * PRINT BOOTSECTOR INFO
1020 hide : pen 3 : locate 4,14 : under on : print "Boot Information"
: under off
1030 locate 4,16 : print "Searching.. "
1040  :
1050 rem * SET LOOP TO GO THROUGH AMOUNT OF FILES
1060 MEM=0 : TYP=0 : YM=0 : NM=0 : for TYPLOOP=1 to NUMFILES : NOPE=0
:      YEP=0      :      locate      22,15      :      print      using
"###";(perc(TYPLOOP,NUMFILES));"%"
1070  :
1080 rem * SET LOOP TO GO THROUGH 512 BYTES OF CURRENT DATA
1090  if leek(start(3)+32)=leek(start(15)+MEM+32) then for CURLOOP=30
to 508 step 2 : gosub 1110 : next CURLOOP : goto 1140 : else goto 1140
1100  :
1110 rem * GET DATA FROM BANK 3 AND COMPARE WITH DATA FROM BANK 15
1120 if compare(start(3)+CURLOOP,start(15)+MEM+CURLOOP) then inc YEP :
else inc NOPE
1130 return
1140 MEM=MEM+512 : if YEP>210 then YM=YEP : TYP=TYPLOOP : goto 1180
1150 next TYPLOOP : YM=0
1160  :
1170 rem * PRINT RESULT OF SEARCH
1180  PRC=perc(YM,240)  :  cls  logic,0,0,120  to  640,199  :  cls
back,0,0,120 to 640,199 : pen 3 : locate 4,16 : print "Boot sector
type : "; : pen 1 : print DESC$(TYP)
1190 pen 3 : locate 4,18 : print "Compare ratio    :"; : pen 2 : print
YM; : pen 3 : print " Correct blocks from "; : pen 2 : print "240 "; :
PRC$=str$(PRC) : pen 3 : print "(";right$(PRC$,len(PRC$)-1);"%)"
1200 locate 4,20 : print "Information       : "; : pen 2
1210  VIRUS=false  :  DMV=false  :  if  instr(DESC$(TYP),"VIRUS!")>0  then
VIRUS=true
```

```
1220 if VIRUS then print "DESTROY IMMEDIATLY!" : else if PRC=100 then
print "Boot sector is virus free" : else if PRC<100 and PRC>0 then
print "Boot sector is not 100%, may be damaged somewhere" : else print
"Bootsector unknown, send it to me please"
1230 cdown : cdown
1240  :
1250 rem * WAIT FOR RETURN TO BE PRESSED
1260 pen 3 : print "    Press ["; : pen 2 : print "RETURN"; : pen 3 :
print "] or "; : pen 1 : print "LEFT MOUSE KEY"; : pen 3 : print " to
continue.."
1270 inverse on : pen 3 : repeat : locate 52,1 : print time$ : wait
vbl : until hardkey=28 or mouse key=1 : inverse off : show on : return
1280  :
1290 rem * IDENTIFY BOOTSECTOR/VIRUS IN MEMORY
1300 TLT$=">> IDENTIFY VIRUS IN MEMORY <<" : NB=1 : if resvector then
MSG$="The reset vector has no programs|residing in it." : BUT$="Phew!"
: else MSG$="The reset vector has been changed, it is|likely that a
virus is causing this!" : BUT$="I'll kill it!"
1310 gosub 3160
1320 return
1330  :
1340 rem * DESTORY BOOTSECTOR/VIRUS
1350  flash off : colour 2,$700 : MSG$="What media do I use to
destroy|the virus??" : BUT$="Disk|Memory" : TLT$=">> DESTROY
BOOTSECTOR/VIRUS <<" : NB=2 : gosub 3160
1360 if ALERT=1 then goto 1380 : else goto 1540
1370  :
1380 rem * DESTORY BOOTSECTOR/VIRUS FROM A DISK
1390 MSG$="Which drive do I use??| " : BUT$=" A | B " : TLT$=">>
DESTROY BOOTSECTOR/VIRUS << " : NB=2 : gosub 3160
1400 DRV=ALERT-1 : drive=DRV
1410 erase 3 : reserve as work 3,512 : floprd start(3),1,0,0,1,DRV
1420  :
1430 rem * DESTROY ALL MACHINE CODE INFORMATION BY COPYING 0 TO IT
1440 doke start(3),0 : for LP=30 to 510 step 2 : doke start(3)+LP,0 :
next LP
1450  :
1460 rem * PRODUCE A NEW BOOT SECTOR FOR THE DISK
1470 produce boot 0,-1,SERIAL,start(3)
1480  :
1490 rem * SAVE NEW BOOT INFORMATION TO THE DISK
1500 flopwrt start(3),1,0,0,1,DRV
1510 M$="on disk"
1520 goto 1560
1530  :
1540 rem * DESTROY VIRUS IN MEMORY
1550 loke $426,0 : M$="in memory"
1560 MSG$="Any viruses "+M$+"|have been destroyed" : BUT$="YIPPEE" :
TLT$=">> DESTROY A VIRUS << " : NB=1 : gosub 3160
1570 return
1580  :
1590 rem * EXAMINE A BOOT SECTOR
1600 flash off : colour 2,$700
1610 MSG$="Which drive shall I grab the|bootsector from??" : BUT$=" A
| B " : TLT$=">> EXAMINE BOOTSECTOR << " : NB=2 : gosub 3160
1620 DRV=ALERT-1 : drive=DRV
```

```
1630  :
1640 DISPLAY=2
1650  :
1660 rem * LOAD BOOTSECTOR
1670 erase 3 : reserve as work 3,512 : floprd start(3),1,0,0,1,DRV
1680  :
1690 rem * DISPLAY BOOTSECTOR ON SCREEN, IN BYTE FORMAT, IN HEX OR
ASCII
1700 gosub 2950 : scroll off : MEMCOUNT=start(3) : pen 3 : inverse off
: for Y=4 to 20 : for X=7 to 72 step 2 : locate X,Y
1710 VAR=peek(MEMCOUNT) : inverse off : pen 3 : if DISPLAY=0 then if
len(hex$(VAR))=3     then     MES$=right$(hex$(VAR),2)     :     else
MES$="0"+right$(hex$(VAR),1)
1720 if DISPLAY=1 then if VAR>31 then MES$=chr$(VAR) : else MES$="."
1730  if  DISPLAY=2  then  if  VAR>31  then  MES$=chr$(VAR)  :  else  if
len(hex$(VAR))=3  then  MES$=right$(hex$(VAR),2)  :  pen  1  :  else
MES$="0"+right$(hex$(VAR),1) : pen 1
1740 print MES$ : pen 3
1750 inc MEMCOUNT : next X : next Y
1760  :
1770 rem * PRINT OPTIONS AT BOTTOM OF SCREEN
1780 inverse on : pen 3 : locate 0,23 : print " (A).. Ascii   (H)..
Hex   (B).. Both   (R).. ReRead   (P).. Print   (E).. Exit "
1790  :
1800 rem * LOOP UNTIL EXIT
1810 EXIT=0 : repeat : KY$=upper$(inkey$)
1820 if KY$="A" then DISPLAY=1 : goto 1690
1830 if KY$="H" then DISPLAY=0 : goto 1690
1840 if KY$="B" then DISPLAY=2 : goto 1690
1850 if KY$="R" then goto 1670
1860 if KY$="P" then goto 1920
1870 if KY$="E" then EXIT=1
1880 inverse on : pen 3 : locate 52,1 : print time$ : wait vbl
1890 until EXIT=1
1900 return
1910  :
1920 rem * PRINT BOOTSECTOR
1930  :
1940 rem * OPEN OUTPUT CHANNEL 1 TO THE PRINTER
1950  open  #1,"PRT"  :  inverse  on  :  pen  3  :  locate  0,23  :  print
space$(80)  :  locate  0,23  :  centre  "Printing  bootsector,  Please
wait..."
1960  :
1970 rem * SET PRINTER (EPSON COMPATIBLE) TO CONDENSED PRINT STYLE
1980 if pready then print #1,chr$(15) : else close #1 : error 10
1990  :
2000 rem * SEND DATA, OUTPUT IS IN HEX, AND ASCII
2010 MEMCOUNT=start(3) : for LP=0 to 15 : FPT1$="" : FPT2$="" : for
LP2=0 to 32
2020     VAR=peek(MEMCOUNT)     :     if     len(hex$(VAR))=3     then
PT1$=right$(hex$(VAR),2) : else PT1$="0"+right$(hex$(VAR),1)
2030 if VAR>31 then PT2$=chr$(VAR) : else PT2$="."
2040 FPT1$=FPT1$+" "+PT1$ : FPT2$=FPT2$+PT2$
2050 inc MEMCOUNT : locate 52,1 : print time$ : next LP2 : if pready
then print #1,FPT1$+"  "+FPT2$ : else error 10
2060 next LP : inverse off
```

```
2070  :
2080 rem * SET PRINTER BACK TO NORMAL
2090 if pready then print #1,chr$(18) : else error 10
2100  :
2110 rem * DO A FEW LINE FEEDS
2120 if pready then print #1,chr$(10)+chr$(10)+chr$(10) : else error
10
2130  :
2140 rem * PRINT DISKBOOT MESSAGE ON PAGE
2150  if  pready  then  print  #1,chr$(27)+"x1"+"Printed  using
"+chr$(27)+"w1"+"DiskBoot"+chr$(27)+"w0"+"  V1.0  on  "+date$+"  at
"+time$
2160  :
2170 rem * PAGE EJECT
2180 if pready then print #1,chr$(12) : else error 10
2190  :
2200 rem * CLOSE OPEN CHANNEL
2210 close #1
2220  :
2230 rem * RETURN TO MAIN EXAMINE SCREEN
2240 goto 1700
2250  :
2260 rem * INSTALL/MEND BOOT SECTOR
2270 flash off : colour 2,$700 : MD=0
2280  MSG$="Do  you  want  to  install  or  mend|a  bootsector??"  :
BUT$="Install|Mend" : TLT$=">> INSTALL/MEND BOOTSECTOR << " : NB=2 :
gosub 3160
2290 if ALERT=1 then MEND=0 : else MEND=1
2300 MSG$="Which drive do I use??| " : BUT$=" A | B " : TLT$=">>
INSTALL/MEND BOOTSECTOR << " : NB=2 : gosub 3160
2310 DRV=ALERT-1 : drive=DRV
2320 erase 3 : reserve as work 3,512 : floprd start(3),1,0,0,1,DRV :
wait 100
2330  :
2340 rem * DRAW SELECTION SCEEN
2350 gosub 2950 : show on : change mouse 2 : CURR=1 : locate 0,5 : pen
3 : centre "Install bootsector to drive "+chr$(65+DRV)
2360 pen 2 : locate 13,8 : print "<<  <" : locate 62,8 : print ">  >>"
2370  pen  1  :  locate  20,8  :  print  mid$(DESC$(CURR),1,40)  :
YPOS=ygraphic(8)-2 : YPOS2=YPOS+10
2380 pen 3 : locate 0,12 : centre "Select boot type using pointer and
left mouse key" : locate 0,13 : centre "Press both mouse keys when
ready to install" : locate 0,14 : centre "or right mouse key to abort"
2390 ink 3 : box xgraphic(13)-2,YPOS to xgraphic(14)+10,YPOS2 : box
xgraphic(17)-2,YPOS to xgraphic(17)+10,YPOS2 : box xgraphic(20)-2,YPOS
to  xgraphic(59)+10,YPOS2  :  box  xgraphic(62)-2,YPOS  to
xgraphic(62)+10,YPOS2  :  box  xgraphic(65)-2,YPOS  to
xgraphic(66)+10,YPOS2
2400  :
2410 rem * SET UP MOUSE DETECTION ZONES
2420 set zone 50,xgraphic(13)-2,YPOS to xgraphic(14)+10,YPOS2
2430 set zone 51,xgraphic(17)-2,YPOS to xgraphic(17)+10,YPOS2
2440 set zone 52,xgraphic(62)-2,YPOS to xgraphic(62)+10,YPOS2
2450 set zone 53,xgraphic(65)-2,YPOS to xgraphic(66)+10,YPOS2
2460  :
2470 rem * GET BUTTON PRESSED AND PRINT TEXT
```

```
2480 repeat : MK=mouse key : ZN=zone(0)
2490 if MK=1 and ZN=50 then PRESSED=1 : wait 10 : bell : if CURR>2
then CURR=CURR-2 : else CURR=1
2500 if MK=1 and ZN=51 then PRESSED=1 : wait 10 : bell : if CURR>1
then dec CURR : else CURR=1
2510 if MK=1 and ZN=53 then PRESSED=1 : wait 10 : bell : if
CURR<(NUMFILES-1) then CURR=CURR+2 : else CURR=NUMFILES
2520 if MK=1 and ZN=52 then PRESSED=1 : wait 10 : bell : if
CURR<NUMFILES then inc CURR : else CURR=NUMFILES
2530 if PRESSED=1 then pen 1 : locate 20,8 : print space$(40) : locate
20,8 : print mid$(DESC$(CURR),1,40) : PRESSED=0
2540 inverse on : pen 3 : locate 52,1 : print time$ : inverse off :
wait vbl : if MK=2 then goto 2740
2550 until MK=3
2560 :
2570 rem * MAKE SURE NOT CHOSEN TO INSTALL A VIRUS
2580 pen 3 : if instr(DESC$(CURR),"VIRUS!")>0 then locate 0,20 :
centre "HEY, YOU CAN BE PROSECUTED FOR SPREADING VIRUSES!!" : bell :
wait 50 : goto 2340
2590 :
2600 rem * GET THE ADDRESS OF THE BOOTINFO STORED IN MEMORY
2610 MEM=start(15)+((CURR-1)*512) : BSR=deek(MEM)
2620 :
2630 rem * CREATE NEW BOOT SECTOR IN MEMORY BANK 14
2640 erase 14 : reserve as work 14,512 : copy MEM,MEM+512 to start(14)
: if MEND=0 then for LP=0 to 29 : poke start(14)+LP,peek(start(3)+LP)
: next LP : doke start(14),BSR
2650 :
2660 rem * MAKE NEW CHECKSUM
2670 if MEND=0 then CHECKCNT=0 : CHECKSUM=0 : for LP=0 to 508 step 2 :
CHECKCNT=CHECKCNT+deek(start(14)+LP) : next LP : CHECKSUM=$1234-
CHECKCNT : doke start(14)+510,CHECKSUM
2680 :
2690 rem * DUMP NEW BOOTSECTOR TO DISK
2700 flopwrt start(14),1,0,0,1,DRV
2710 :
2720 MSG$="Boot sector has been|successfully installed/mended " :
BUT$="Brill" : TLT$=">> INSTALL BOOT SECTOR <<" : NB=1 : gosub 3160
2730 :
2740 return
2750 :
2760 rem * INFORMATION SCREEN
2770 gosub 2950 : bell : pen 3
2780 locate 0,5 : centre "Disk Boot V1.0" : cdown : cdown
2790 centre "(C)GBP Software 1992/1993" : cdown : cdown
2800 centre "Designed and Programmed by Neil Halliday 92/93" : cdown :
cdown : print
2810 print "Disk Boot is a bootsector administration utility, that
incorporates a database"
2820 print "of the most common bootsector and virus types on the Atari
ST. Disk Boot also"
2830 print "allows the user to destroy, create or mend any bootsector
type (not viruses!)"
2840 print "that is contained in the library files."
2850 print : print "This version has a library containing information
on";(NUMFILES-1);" different bootsector"
```

```
2860 print "and virus types."
2870 pen 2 : cdown : centre "PRESS ANY KEY.."
2880 wait key : return
2890  :
2900 rem * EXIT PROGRAM
2910  flash off : colour 2,$700 : MSG$="Exit, are you sure?| " :
BUT$="Yes|No" : TLT$=">> EXIT FROM PROGRAM <<" : NB=2 : gosub 3160
2920 if ALERT=1 then drive=DF_DRV : dir$=DF_PATH$ : end
2930 return
2940  :
2950 rem * CLEAR WORKAREA ON SCREEN
2960   cls   logic,0,0,ygraphic(3)   to   640,ygraphic(21)   :   cls
back,0,0,ygraphic(3) to 640,ygraphic(21)
2970 clear key : return
2980  :
2990 rem * ERROR TRAPPING ROUTINE
3000  :
3010 rem * OPEN DATABASE FILE
3020 NUM=errn : open #2,"R","\DISKBOOT.DAT\ERRORS.REC"
3030  :
3040 rem * SET FILE STRUCTURE
3050 field #2,40  as N$,250  as D$
3060  :
3070 rem * GET RECORD USING ERRORNUMBER
3080 get #2,NUM
3090  :
3100 rem * PRINT RESULT
3110 inverse off : pen 3 : gosub 2950 : locate 0,5 : print N$ : print
: print D$ : print : print "Press ANY key.." : wait key
3120  :
3130 rem * RESUME
3140 close #2 : resume 330
3150  :
3160 rem * ALERT BOX ROUTINE
3170 bell : show on : change mouse 2 : for ZN=100 to 110 : reset zone
(ZN) : next ZN
3180 I=instr(MSG$,"|") : B$=mid$(MSG$,1,I-1) : C$=mid$(MSG$,I+1)
3190 A=max(len(B$),max(len(C$),max(len(BUT$),len(TLT$))))
3200 ST1=(80/divx-A)/2-2 : Y1=(24/divy+7)/2
3210    ink    0    :    bar    xgraphic(ST1),ygraphic(Y1+1)    to
xgraphic(ST1+A+4),ygraphic(Y1+7)
3220    ink    3    :    box    xgraphic(ST1),ygraphic(Y1+1)    to
xgraphic(ST1+A+4),ygraphic(Y1+7)
3230          bar          xgraphic(ST1),ygraphic(Y1)-2          to
xgraphic(ST1+A+4),ygraphic(Y1+1)
3240 paper 0 : pen 3 : locate ST1+2,Y1+2 : print B$ : locate
ST1+2,Y1+3 : print C$
3250 paper 3 : locate ST1+(A+5-len(TLT$))/2,Y1 : pen 0 : print TLT$ :
paper 0 : pen 3
3260 on NB goto 3280,3320,3340
3270  :
3280 rem * ONE BUTTON
3290 T$=BUT$ : M$=T$ : goto 3370
3300  :
```

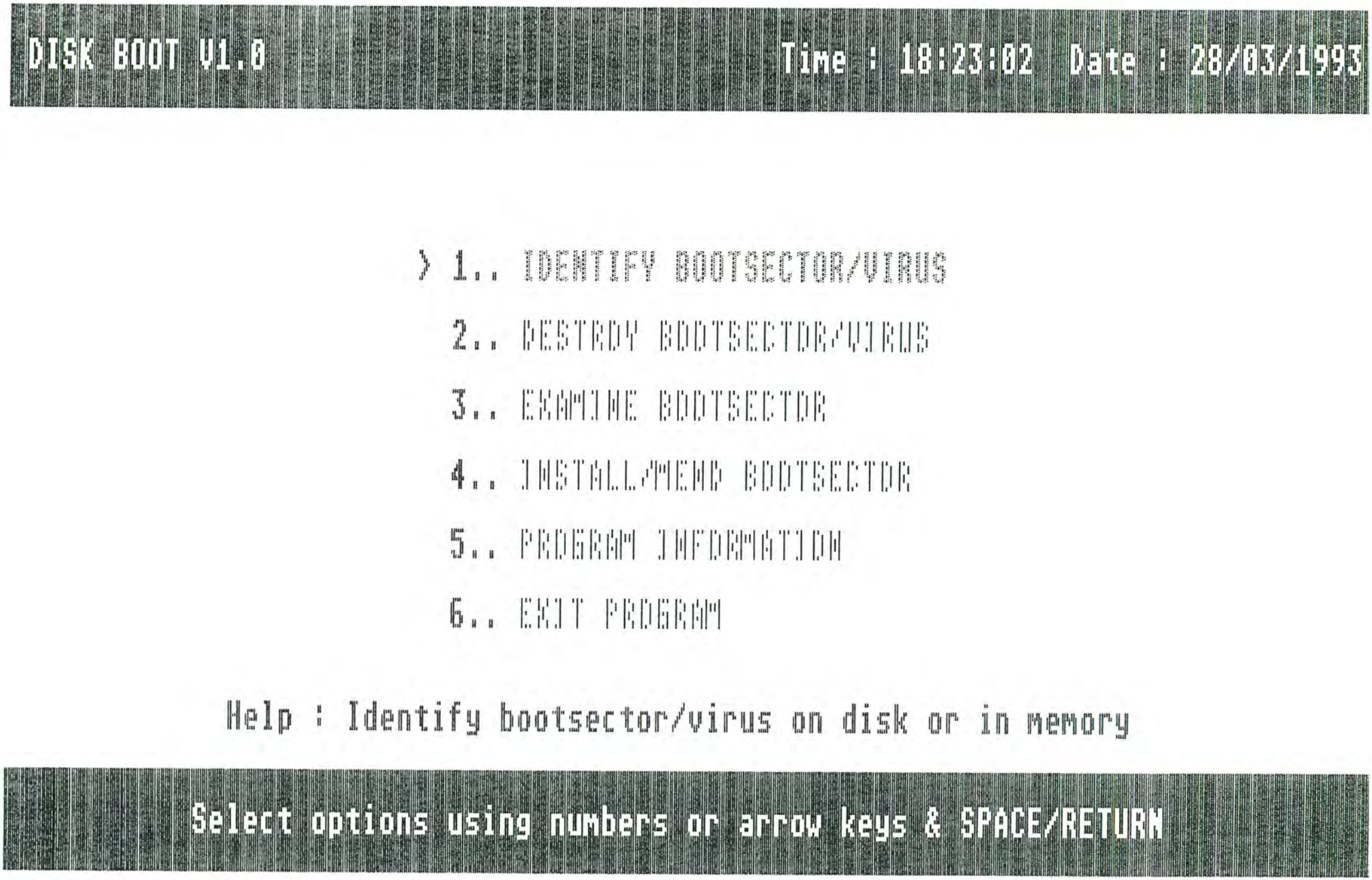
```
3310 rem * TWO BUTTONS
3320 I=instr(BUT$,"|") : M$=mid$(BUT$,1,I-1) : M2$=mid$(BUT$,I+1) :
SP=A+5-len(BUT$) : SP=SP/2 : T$=M$+space$(SP)+M2$ : goto 3370
3330 :
3340 rem * THREE BUTTONS
3350 I=instr(BUT$,"|") : M$=mid$(BUT$,1,I-1) : I2=instr(BUT$,"|",I+1)
: M2$=mid$(BUT$,I+1,I2-I-1) : M3$=mid$(BUT$,I2+1) : SP=(A+6-
len(BUT$))/3 : T$=M$+space$(SP)+M2$+space$(SP)+M3$
3360 :
3370 rem * PRINT BUTTONS AND DRAW BOX
3380 ink 3 : ZN=120 : POS=ST1+(A+5-len(T$))/2 : locate POS,Y1+5 :
print T$ : YG=ygraphic(Y1+5)-2 : BX1=POS : LMS=len(M$) : gosub 3420 :
POS=POS+len(M$)
3390 if NB>1 then BX1=POS+SP : LMS=len(M2$) : gosub 3420 :
POS=POS+len(M2$)
3400 if NB>2 then BX1=POS+SP*2 : LMS=len(M3$) : gosub 3420
3410 paper 0 : goto 3450
3420 box xgraphic(BX1)-3,YG to xgraphic(BX1+LMS)+2,YG+10 : set zone
ZN,xgraphic(BX1)-3,YG to xgraphic(BX1+LMS)+2,YG+10 : inc ZN : return
3430 :
3440 rem * TEST FOR MOUSE BUTTONS AND RETURN
3450 MK=mouse key : Z2=zone(0)
3460 if MK=1 and Z2>=119 then ALERT=Z2-119 : goto 3480
3470 inverse on : pen 3 : locate 52,1 : print time$ : inverse off :
wait vbl : goto 3450
3480 clear key : hide : return
3490 :
3500 rem * DATA FOR MAIN MENU
3510 data "IDENTIFY BOOTSECTOR/VIRUS","DESTROY
BOOTSECTOR/VIRUS","EXAMINE BOOTSECTOR","INSTALL/MEND
BOOTSECTOR","PROGRAM INFORMATION","EXIT PROGRAM"
3520 :
3530 rem * DATA FOR MENU HELP
3540 data "Identify bootsector/virus on disk or in memory","Destroy
all bootsector/virus data on disk or in memory","Examine the contents
of a bootsector on disk","Create or mend and existing bootsector on
disk","A few notes about this program","Leave Disk Boot and retutn to
GEM"
```

My Own Commands.

To show the use of the extension commands I have written, I have highlighted them on the Basic source code pages.

System Testing.

The results of the system testing can be found in appendix D.


DISK BOOT V1.0
Time : 18:23:02 Date : 28/03/1993
> 1.. IDENTIFY BOOTSECTOR/VIRUS
2.. DESTROY BOOTSECTOR/VIRUS
3.. EXAMINE BOOTSECTOR
4.. INSTALL/MEMB BOOTSECTOR
5.. PROGRAM INFORMATION
6.. EXIT PROGRAM
Help : Identify bootsector/virus on disk or in memory
Select options using numbers or arrow keys & SPACE/RETURN

DISK BOOT V1.0 Time : 18:37:52 Date : 28/03/1993

Disk Information

Number of tracks : 0 - 79
Number of sides : 2
Volume label : None

Sectors per track : 10
Total number of sectors : 1600
Disk space free : 253 k

Boot Information

Boot sector type : FLOPPY COPY CREW VFB V3 - PROTECTOR
Compare ratio : 240 Correct blocks from 240 (100%)
Information : Boot sector is virus free

Press [RETURN] or LEFT MOUSE KEY to continue..

DISK BOOT V1.6 Time : 18:01:06 Date : 28/03/1983

(A).. Ascii (H).. Hex (B).. Both (R).. ReRead (P).. Print (E).. Exit

```
60 1C 90 49 42 4D 20 20 0B 29 BF 00 02 02 01 00 02 70 00 C4 05 F9 03 00 09 00 02 00 00 00 4D F8 82   '.IBM  .)?.....p.D.y.....Mx
40 3D 7C 07 77 00 1E 41 FA 00 C4 61 00 00 AA 20 78 04 2E B1 FC 00 3F FF FC 6F 04 41 E8 FF FC 22 48   @=|.w..Az.Da..* x..1|.?|o.Ah|"H
0C 99 12 12 34 56 67 4C 41 E8 FE 00 B1 FC 00 00 00 00 6E EA 0C B8 31 41 59 26 04 26 67 28 3C BC 04   ...4VgLAh~.1|....nj.81AY&.&g(<<.
04 41 FA 00 C5 61 6E 20 3C 00 05 00 00 53 80 66 FC 21 FC 26 3C 00 00 01 40 3C BC 07 77 3D 7C 00 00   .Az.Ean <....Sf|!|&<...@<<.w=|..
00 1E 4E 75 3C BC 04 00 41 FA 00 BB 61 46 42 47 60 10 2A 48 3C BC 04 00 3E 3C 00 01 41 FA 00 E2 61   ..Nu<<..Az.;aFBG'.*H<<..><..Az.ba
32 41 FA 01 23 61 2C 3F 3C 00 07 4E 41 54 8F 48 40 0C 00 00 3B 67 C0 0C 00 00 3C 66 EA 42 B8 04 26   2Az.#a,?<..NATH@...;g@...<fjB8.&
42 B8 04 2A B0 7C 00 00 67 AC 42 9D 20 78 00 04 4E D0 2F 08 3F 3C 00 09 4E 41 5C 8F 4E 75 3F 3C 00   B8.*0|..g,B x..NP/.?<..NA\Nu?<.
07 4E 41 54 8F 48 40 1B 45 1B 66 1B 59 20 25 54 68 65 20 4D 65 64 77 61 79 20 42 6F 79 73 20 50 72   .NATH@.E.f.Y %The Medway Boys Pr
6F 74 65 63 74 6F 72 20 49 56 1B 59 22 2A 49 6D 70 72 6F 76 65 64 20 42 79 20 54 68 65 20 42 42 43   otector IV.Y"*Improved By The BBC
21 00 1B 59 2A 28 59 6F 75 20 61 72 65 20 46 52 45 45 20 6F 66 20 76 69 72 75 73 65 73 20 21 00 1B   !..Y*(You are FREE of viruses !..
59 26 23 54 68 65 20 52 45 53 45 54 20 56 45 43 54 4F 52 20 68 61 73 20 62 65 65 6E 20 63 68 61 6E   Y&#The RESET VECTOR has been chan
67 65 64 1B 59 28 2B 49 74 20 4D 41 59 20 62 65 20 61 20 76 69 72 75 73 00 1B 59 26 2D 52 45 53 45   ged.Y(+It MAY be a virus..Y&-RESE
54 20 52 45 53 49 44 45 4E 54 20 70 72 6F 67 20 69 6E 20 6D 65 6D 6F 72 79 1B 59 28 2B 49 74 20 69   T RESIDENT prog in memory.Y(+It i
73 20 4D 4F 52 45 20 74 68 61 6E 20 6C 69 6B 65 6C 79 20 61 20 56 49 52 55 53 20 21 21 00 1B 59 2C   s MORE than likely a VIRUS !!..Y,
2C 46 31 20 43 6F 6E 74 69 6E 75 65 1B 59 2E 2C 46 32 20 43 6C 65 61 72 00 74 69 6E 20 42 61 63 6B   ,F1 Continue.Y.,F2 Clear.tin Back
73 63 68 61 74 20 0D 0A 1B 71 0D 0A 00 07 00 2E BE 00 00 4E 4E 4E 4E 4E 4E 00 00 00 00 02 02 01 00   schat ...q.....>..NNNNNN........
02 70 00 16 03 F8 05 00 0A 00 01 00 00 00 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E   .p...x.....NNNNNNNNNNNNNNNNNNN
```

Printed using DiskBoot V1.0 on 28/03/1993 at 18:56:56

```
60 38 90 49 42 4D 20 20 C5 FB 4A 00 02 02 01 00 02 70 00 40 06 F9 03 00 0A 00 02 00 00 00 48 79 00
0B FF FF 4E 4D 48 7A 01 D4 C0 3C 00 07 B0 3C 00 04 62 0A 48 7A 00 12 60 04 26 4F 60 E0 3F 3C 00 09
4E 41 2E 4B 4E 75 07 0D 0A 28 56 65 72 2E 20 31 38 2E 34 2E 39 31 29 0D 0A 1B 70 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 48 65 6C 6C 6F 20 21 21 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0D 0A 20 20 20 20 20 49 20 61 6D 20 79 6F 75 72 20 70 65 72 73 6F 6E 61 6C 20 62 6F 6F 74 20 73 65
63 74 6F 72 20 20 20 20 0D 0A 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 47 55 41 52 44 49 41 4E 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 0D 0A 20 20 20 41 73 20 6C 6F 6E 67 20 61 73 20 49
20 64 69 73 70 6C 61 79 20 74 68 69 73 20 6D 65 73 73 61 67 65 20 20 20 0D 0A 20 20 20 20 20 20 20
20 20 20 59 4F 55 52 20 42 4F 4F 54 20 53 45 43 54 4F 52 20 49 53 20 20 20 20 20 20 20 20 20 20 0D
0A 20 20 20 20 20 20 20 4E 4F 54 20 49 4E 46 45 43 54 45 44 20 42 59 20 41 4E 59 20 56 49 52 55 53
20 20 20 20 20 20 20 0D 0A 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 0D 0A 20 20 20 20 20 54 68 69 73 20 67 75 61 72 64 69
61 6E 20 77 61 73 20 70 6C 61 63 65 64 20 6F 6E 20 20 20 20 20 20 20 0D 0A 20 20 20 20 79 6F 75 72
20 64 69 73 6B 65 74 74 65 20 62 79 20 46 41 53 54 43 4F 50 59 20 50 52 4F 20 20 20 20 20 20 0D 0A
20 28 63 29 20 31 39 39 31 20 49 43 50 20 56 65 72 6C 61 67 2C 20 4D 61 72 74 69 6E 20 42 61 63 6B
73 63 68 61 74 20 0D 0A 1B 71 0D 0A 00 07 00 58 BB 00 00 4E 4E 4E 4E 4E 4E 00 00 00 00 02 02 01 00
02 70 00 16 03 F8 05 00 0A 00 01 00 00 00 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E
```

```
'8IBM  E(J......p.@.y........Hy.
.NMHz.T@<..0<..b.Hz..'.&0''?<..
NA.KNu...(Ver. 18.4.91)...p
          Hello !!
..     I am your personal boot se
ctor    ..              GUARDIAN
                ..  As long as I
 display this message   ..
   YOUR BOOT SECTOR IS           .
.       NOT INFECTED BY ANY VIRUS
        ..----------------------
---------------..     This guardi
an was placed on       ..    your
 diskette by FASTCOPY PRO      ..
 (c) 1991 ICP Verlag, Martin Back
schat ...q.....X;..NNNNNN........
.p...x........NNNNNNNNNNNNNNNNNN
```

Printed using DiskBoot V1.0 on 28/03/1993 at 18:55:14

DISK BOOT V1.0 Time : 20:07:36 Date : 28/03/1993

Install bootsector to drive A

<< < ACTION SERVICE - BOOT SECTOR > >>

Select boot type using pointer and left mouse key
Press both mouse keys when ready to install
or right mouse key to abort

Select options using numbers or arrow keys & SPACE/RETURN

D I S K B O O T V 1 . 0

CONTENTS

Introduction.

Disk Boot is a versatile virus killing package that allows the user to keep a check of all the contents of their diskette collection. Disk Boot also contains an extensive database of bootsectors allowing the program to identify around 92 different bootsectors on the ATARI ST. The software package contains easy to understand screen layouts, and text that is simple. For those users who are still a little uneasy about using programs that modify your diskettes then don't worry, as Disk Boot contains online help, that is easily accessible from the main menu.

If any diskette errors arise please make a note of the error message that is displayed, and send it, along with your Disk Boot diskette to the following address :

Disk Boot Faults,
GBP Software,
115 Heaton Street,
Denton,
Manchester,
M34 3RY.

Legalities.

Although this program is released into shareware, GBP Software still retain the full copyright on the program, it's contents and the source code. No part of the registered version of the product and it's accompanying documentation may be sold, copied, or hired without the strict permission of GBP Software.

The program has been tested to our fullest capabilities on most of that ATARI ST range of computers. The programmer of Disk Boot and GBP Software will take no responsibility for any damages to disks, computers or any other hardware, software or data information loses due to the usage or design of this program.

Loading Disk Boot.

To load Disk Boot, turn the power switch on your ATARI ST to the off position. Place the Disk Boot diskette into the computers internal disk drive, switching the computers power on afterwards. You should now be presented with the medium resolution GEM Desktop, and an open file window. If this file window does not appear please consult the GEM Operators manual supplied with your computer.

Place the mouse pointer over the program icon that reads 'DISKBOOT.PRG', and double click the left mouse key. This will invert the icon, and then display a blank screen with the words 'DISKBOOT.PRG' centred at the very top of the screen. The program will now load. If you have any trouble loading the program please consult your GEM Operators manual supplied with your computer.

After a while the screen will clear, and the Disk Boot heading should be displayed at the top of the screen, with current time* and date*. The program will then inform you that it is scanning the library files, followed by the loading of the library files. If any errors occur during this point of the program you should make a note of the error given, and send your diskette, along with the error to the address on the introduction page.

* PLEASE NOTE : The time and date will only be correct if your computer system has a battery backed clock, or an external clock cartridge installed.

Using The Online Help Function.

This option is automatically switched on when first booting Disk Boot. The option can be controlled from the main menu by pressing the [HELP] key on your keyboard. The [HELP] key is located above, and a little left of the up arrow key.

Pressing the [HELP] for the first time after loading Disk Boot will switch the help function off, and will remain switched off until the [HELP] key is pressed for a second time.

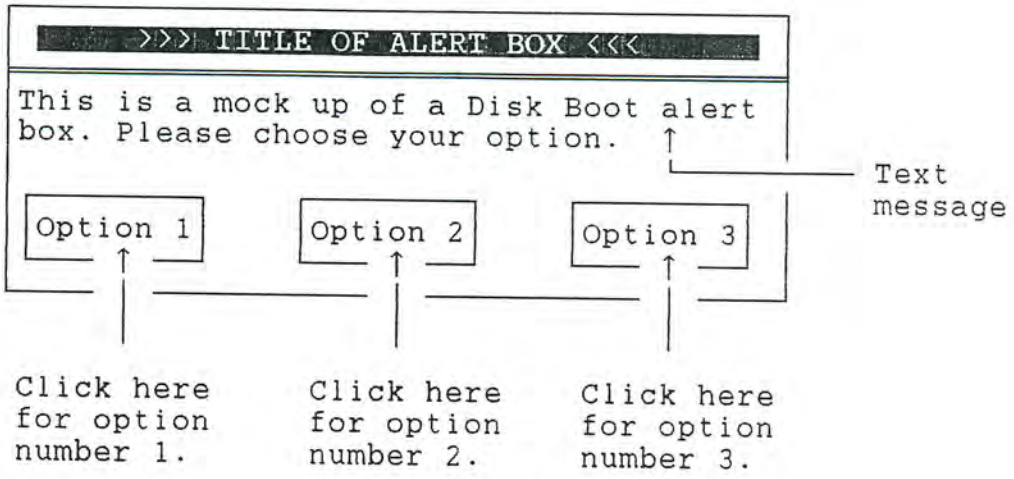
Using The Alert Boxes,

During the execution of Disk Boot you will be asked a number of questions, for example which disk drive to use if you are examining a diskettes bootsector. Disk Boot uses "Alert Boxes" for these functions, if you are used to the GEM desktop then you can skip this section of the manual. For those of you who are new to the computer, I will explain what an alert box, and how you can use it.

An alert box is a box that holds information, and usually has options that can be selected from them. The GEM alert boxes contain a symbol, being either a question mark "?", an exclamation mark "!" or a graphic saying STOP. The Disk Boot alert boxes are very similar to the GEM desktop alert boxes, but they lack a symbol. Instead of this the Disk Boot alert boxes have an inversed title.

Using The Alert Boxes. Cont.

In the centre of the alert box is the main text, and underneath this are the "Radio Buttons". This buttons can come on their own, in two's, or in three's. The "Radio buttons" are the options for that particular alert box, and the option can be chosen by moving the mouse point over the text, and pressing the left mouse key. Here is a mock up alert box for you to have a look at..



The Main Menu.

This is the heart of Disk Boot, where all the operations of the program can be accessed. The following options are available from the main menu.

Identify Bootsector/Virus

Destroy Bootsector/Virus

Examine Bootsector

Install/Mend Bootsector

Program Information

Exit Program

You will notice that one of the options is flashing red and white. This is the current option selected, and will be executed when [SPACE] or [RETURN] is pressed. Options can be selected in two ways..

1.. By pressing the corresponding number to the left of the option

2.. By moving the current option display to the required function, and then pressing [SPACE] or [RETURN].

The current option (The one that's flashing) can be select by using the up and down arrow keys. Pressing the up arrow key will move the current option up, and pressing the down arrow key will move the current option down. Pressing the [SPACE] or [RETURN] keys will result in the flashing option being executed.

Identify Bootsector/Virus.

This function can do one of two things..

1.. Check memory for any viruses

2.. Identify a bootsector or a virus that might be on a diskette

Check Memory For A Virus.

Once the function has been selected, click the memory option from the alert box. The program will then check for any viruses that might be located anywhere in memory. An alert box will display the result of the search.

Identify Bootsector/Virus on Diskette.

Once the function has been selected, click the disk option from the alert box. You will then be asked a further question, being which disk drive where the diskette you would like to check is located. Click on either. A, for the computers internal disk drive, or B, for your external drive. If you click on B and you have not got an external disk drive then don't worry, the program will just read from dive A anyway.

The diskette's bootsector will then be loaded, and the standard disk information will be taken from the file. The bootsector will then be loaded once again for reference purposes.

The screen of this function is split into two sections, Disk information, and Boot information.

Disk Information.

The following information about your diskette is printed in this section of the screen..

Number of tracks	Sectors per track
Number of sides	Total number of sectors
Volume label	Disk space free

Boot Information.

The computer will search it's memory of bootsectors to see if the bootsector you have loaded into memory matches any on record. This operation should only take around 5 seconds at the most. After the search the following information is printed about the bootsector.

Bootsector type

Compare ratio

Information

If the bootsector you loaded is recognised then the bootsector type will contain the name of the bootsector, otherwise it will read "Unknown Bootsector". The compare ratio is that amount of blocks that your loaded bootsector contained, that matched a bootsector in the database. If your bootsector is not recognised then this ratio will be 0. Information, is any further information that you may need to know about the bootsector, e.g. if it is damaged in someway.

To exit from the function once the operation is complete, simply press the [RETURN] key, or press the left mouse key once.

Destroy Bootsector.

Entering this function you will be asked if you want to destroy a virus in memory, or on a diskette.

Destroying A Virus In Memory.

This operation is almost instantaneous, and the result will be an alert box telling you that the operation is complete. The function deletes any virus information that is in memory, where it shouldn't be.

Destroy A Virus On Diskette.

Please make sure the diskette in the drive you want to destroy the virus on is write enabled, as yet there is no way of telling if the drive id write protected or not. As before this operation if almost instantaneous, the bootsector is loaded, and then a standard GEM bootsector is created, can then saved back onto the diskette. An alert box will be printed telling you that the operation is finished, and successful.

Examine Bootsector.

This function allows you to examine the contents of any bootsectors that you may have, or those that are not recognised by Disk Boot. Entering this option confronts you with a display of the current bootsector in ASCII, and a small menu at the bottom of the screen. These options are..

(A).. ASCII

Function is executed by pressing "A" and will display the current bootsector in ASCII.

(H).. Hex

Function is executed by pressing "H" and will display the current bootsector in Hexadecimal values.

(B).. Both

Function is executed by pressing "B" and will display the current bootsector in ASCII, and with any non ASCII characters being displayed in hex.

(R).. ReRead

Function is executed by pressing "R" and will load a new bootsector from the selected drive into memory. The bootsector will then be displayed in the current display mode selected.

(P).. Print

Function is executed by pressing "P" and will print the current bootsector to the EPSON compatible printer connected to the computers parallel port. The printed output is in Hexadecimal, with an ASCII layout.

(E).. Exit

Function is executed by pressing "E" and will return the user to the main menu.

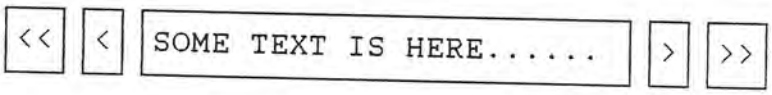
You may find that the printing of some bootsectors will not be very successful, this will be because the bootsector contains printer control codes that will turn all the functions of the printer on and off. This however, can not be helped at the moment.

Install/Mend Bootsector.

Firstly choose if you want to install or mend a bootsector.

Choose install if you want to keep the current file information on your diskette kept in tact. The mend function should ONLY be used if the identify bootsector/virus function says that the bootsector may be damaged somewhere, also this function should ONLY be used if the bootsector name is followed by "- BOOTSECTOR". If the bootsector name is followed by "- PROTECTOR" then you should stay away from the mend option, otherwise you are likely to damage your disk, and loose all the information on it.

After you have chosen what function you want to do you will be presented with a screen that has this located at the top of it..



The central box contains the name of the bootsector currently selected to install/mend. The bootsector can be change in two ways..

1.. Increase/Decrease in single steps, by pressing the "<" for decrease and the ">" for increase.

2.. Increase/Decrease in double steps, by pressing the "<<" for decrease and the ">>" for increase.

The selected bootsector can then be installed by pressing the left and right mouse keys at the same time. To exit from the function simply press the right mouse key on it's own.

If you select to install the bootsector then the new bootsector will be created, and then saved onto the diskette on the chosen disk drive. You will then be presented with an alert box to say that all has gone well.

Program Information.

This function tells you a little about the program, and will display the current number of bootsectors that Disk Boot can recognise. Pressing any key will leave this function and return to the main menu.

Exit Program.

You will be presented by an alert box asking if you are sure that you what to leave Disk Boot. If you are sure you want to leave then click on "YES" otherwise click on "NO".

Acknowledgements.

My thanks are sent to the following people and places..

Manderin Software, for STOS basic and STOS compiler.

Hisoft, for GenST 68000 Assembler.

Abacus Software, for a brilliant inside the ST book.

Atari Uk.

Many thanks for your interest in this software...

Updating Current System.

In the event that the system needs updating, main code re-writing, this section will discuss how to do so.

The main program structure diagrams, file specifications, print and screen layouts, data flow diagrams and logical data models can be found in appendix B.

Adding New Extension Commands.

This would probably be the most difficult of the update operations to perform. STOS extensions must be in the correct format to work perfectly, if any imperfections are found in the source, not only will it effect that particular routine, but it will also make STOS think that an incorrect extension format is being loaded, leading to STOS doing a spectacular system crash. The easiest way of programming the extension is to follow the small amounts of documentation that are located in the assembly file. Another good source of information are the ST Internals book published by Abacus Software, and The Games Makers Manual published by Sigma Press.

Updating The Main Menu.

As you will have seen from the screen layout of the main menu, that not all that much space is left on the screen for new options. I suggest that if new options are to be added to the main menu, that the whole main menu is re-written. This may seem like a tedious task, but at the moment the menu routine is specifically written to cope with only 6 options. Adding more options would mean a quite a lot of screen modification, and a fair amount of code alterations.

Updating The Actual Code.

You will have noticed that the STOS programming environment uses line numbers instead of procedures. STOS incorporates a function called renumber, that renumbers your code into increments of ten lines. When Disk Boot was finally finished, I used the renumber function, and because of this the program does not leave a very large amount of space for inserting any new code. The only way around this is to use the renumber function once again, but in this special way..

Lets say we wanted to insert a routine between the main menu, starting at line 150, and the identify bootsector/virus section, which starts at line 600. We can use the following command..

```
renum 10000,10,600-9999
```

This will renumber all the lines between 600 and 9999, with increments of 10 lines, starting with the line number 10000. This will then allow you to program routines in the lines 600-9999 that were previously taken up by the program.

Writing For The Alert Box Routine.

The alert box routine can be accessed by a gosub command. parameters must be passed to the routine. These variables control the output, selection options, and the title of the alert box. The routine itself will automatically draw the boxes to the correct size, and display all the text. It will also wait until an option is selected, and a return variable will contain a number, which corresponds to the selected option. To call the alert box routine the code looks like this..

```
MSG$="Your message goes here|Use a pipe ("|") to split separate lines" : BUT$="Button 1|Button 2" : TLT$=">>> TITLE OF ALERT BOX <<<" : NB=2 : gosub 3160
```

MSG$ This is the actual message to appear in the alert box

BUT$ This is the button string, each button is separated by a pipe ("|").

TLT$ This is the title of the alert box

NB This is the number of buttons that you have in the BUT$ string.

The alert box routine is then called by the gosub 3160 command. When finished, the number of the button pressed is returned in the variable ALERT.

Installing New Bootsectors.

It is possible to add extra bootsectors to the program. To do this run the ADD.BAS file under STOS basic. The program will ask you to place a diskette in the drive, and will then dump it's bootsector. You will then be asked to insert the Disk Boot diskette, and asked to type in a description of the bootsector. This description should be followed by one of the following three

"- VIRUS!"

This should be present at the end of the description if the bootsector is one of a virus.

"- BOOTSECTOR"

This should be present at the end of the description if the bootsector is for back up purposes, so it can be used to mend bootsectors later on.

"- PROTECTOR"

This should be present as the end of the description if the bootsector is a protecting bootsector. You can identify protecting bootsectors very easily, as they nearly always print a message on the screen saying your disk is virus free. These types are stored, so they can be installed onto any diskette, leaving the file data in tact.

Appraisal.

On the whole the program did what it was supposed to do, only two bugs came to my notice during the execution of it. The bugs are not that serious, and can be easily corrected.

The Error File When Compiled.

When compiled Disk Boot crashed if an error occurred, and the user had changed the diskette. This was because the error file could not be found. To get around this, I could install a RAM Disk, and write a little program that automatically copies the error file into it. I would then need to alter the program to load the error file from C:\ instead of A:\, which is the default.

Printer Controls Codes.

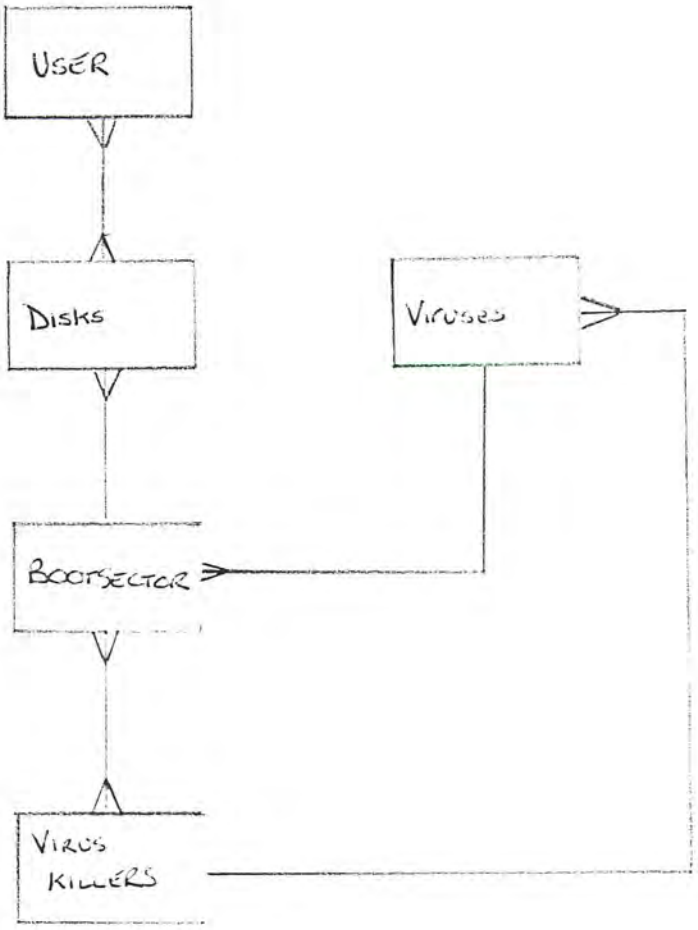
This one was the only major problem. It arose from the print bootsector option in examine bootsector. The printers use control codes to handle the style of text that is output, and some bootsectors contained the codes instead them. This resulted in when the bootsector was printed, the printer was taken out of condensed, which is what it was originally placed in to print correctly, and was placed into all the other printer options, e.g. bold, italics, underlined etc.

This problem can be solved by using a routine similar to the routine used to display the bootsectors on the screen. The routine will examine the contents of the bootsector, and anything that is the same as a printer control code, will be sent as a string instead of a number, e.g. 10 in hex would make the printer cartridge return, but if I sent it as a string containing "10" then the printer would think that it is text, and not a number, thus not changing the current print style.

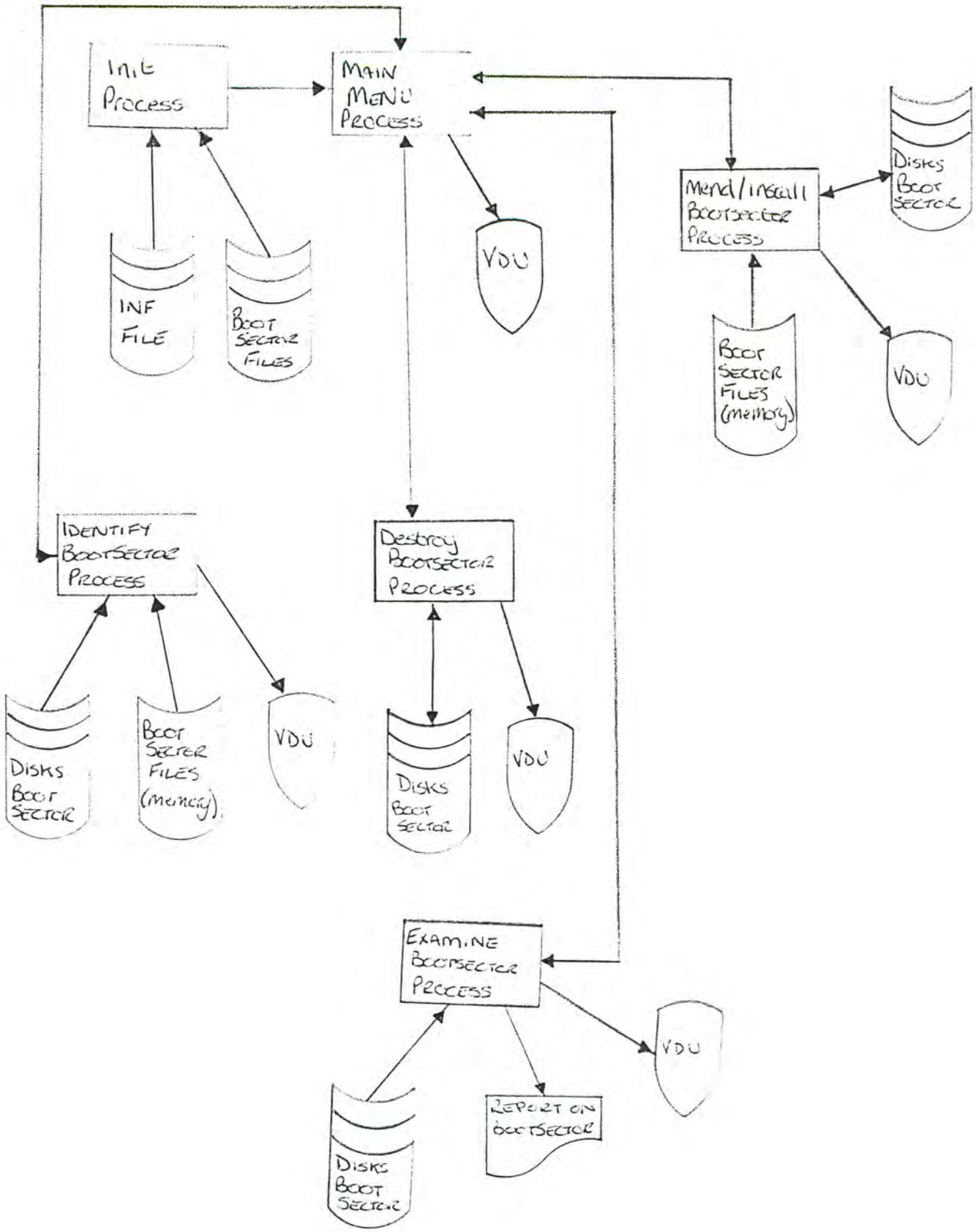
Problems With The Project.

The main problem that I found with the project, was the printing of the screen dumps. My printer is a 24 pin Citizen, but the only printer drivers I have are for 9 pin printers, this has made my output quite blocky, and also has sometimes made the positioning of the screen different. The only way I can get around this is to buy some new printer drivers.

I also found it quite hard writing the manual, as I am not used to writing my own manuals. When I write a program normally, the manual writing is handled by the other members of GBP Software. We always do this, as we think that if a person who knows nothing about the program can write the manual better.



Appendix B - Logical Data Model



Appendix B - Data Flow Diagram

FILE SPECIFICATION FORM

File	DiskBoot.INF	Medium	Disk
Organisation	Sequencial	Key field	Numfiles
General Contents	Boorsector Information + Filenames		

Record name

Field	Length (max)	Type	Always present (Y/N)	Permitted Contents
Numfiles	4	N	Y	Ø-9999
INFORMATION	4Ø	AN	Y	All chars
FILENAME	12	AN	Y	8 chars + ".Sec"
				↑ ↑ ↑
				These are repeated for the number of files/records on the disk (numfiles).

FILE SPECIFICATION FORM

File	#.Sec	Medium	Disk
Organisation	Sequential	Key field	N/A.
General Contents	A 512 bytes Bootsector Datafile.		

Record name

Field	Length (max)	Type	Always present (Y/N)	Permitted Contents
Branch Instruction	2	N	Y	ØØØØ or 60??
Loader	6	AN	Y	"Loader" or Spaces
Serial	3	N	Y	24 Bit serial number.
BPS	2	N	Y	Bytes Per Sector.
SPC	1	N	Y	Sectors Per Cluster.
RES	2	N	Y	Reserved Disk Sectors
FAT	1	N	Y	FAT tables on the disk.
Directories	2	N	Y	max number of directory entries
SEC	2	N	Y	Sectors on the disk
MED	1	N	Y	Media descripter byte.
SECFAT	2	N	Y	Sectors per FAT
SPT	2	N	Y	Sectors Per Track
SIDES	2	N	Y	SIDES on the Disk
HID	2	N	Y	HIDDEN SECTORS
UPS			Y	User Programable Space.
CHECK	2	N	N	CHECKSUM.
				▲ ▲ ▲
				These 2 are only present
				if the branch instruction
				is 60ØØ or more.

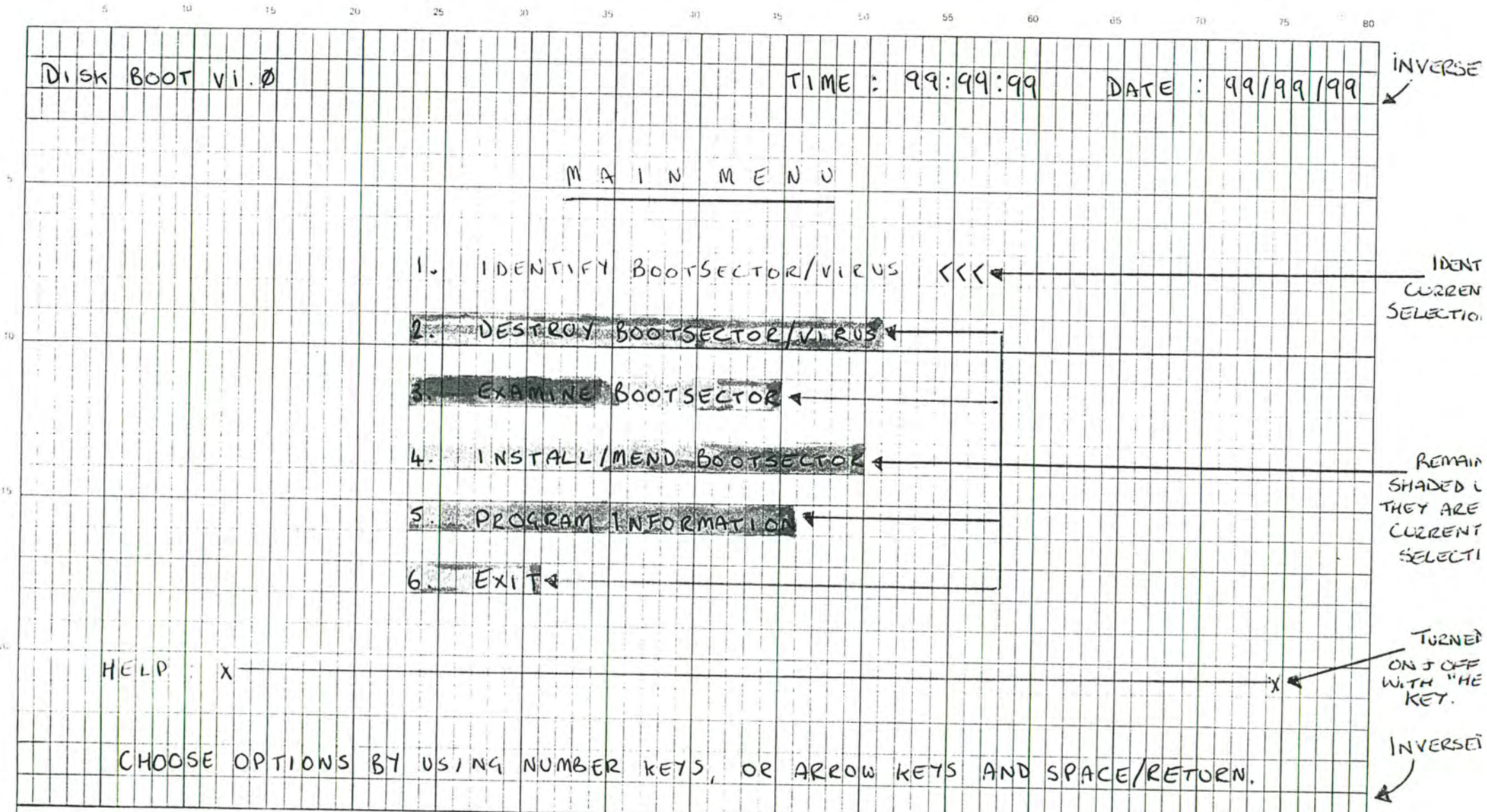
FILE SPECIFICATION FORM

File	ERRORS.REC	Medium	DISK
Organisation	RANDOM	Key field	NONE
General Contents	ERROR INFORMATION		

Record name

Field	Length (max)	Type	Always present (Y/N)	Permitted Contents
NAME	40	AN	Y	All chars
DESCRIPTION	250	AN	Y	All chars
				fields are repeated 88 times (Total number of Possible errors).

Visual display unit layout



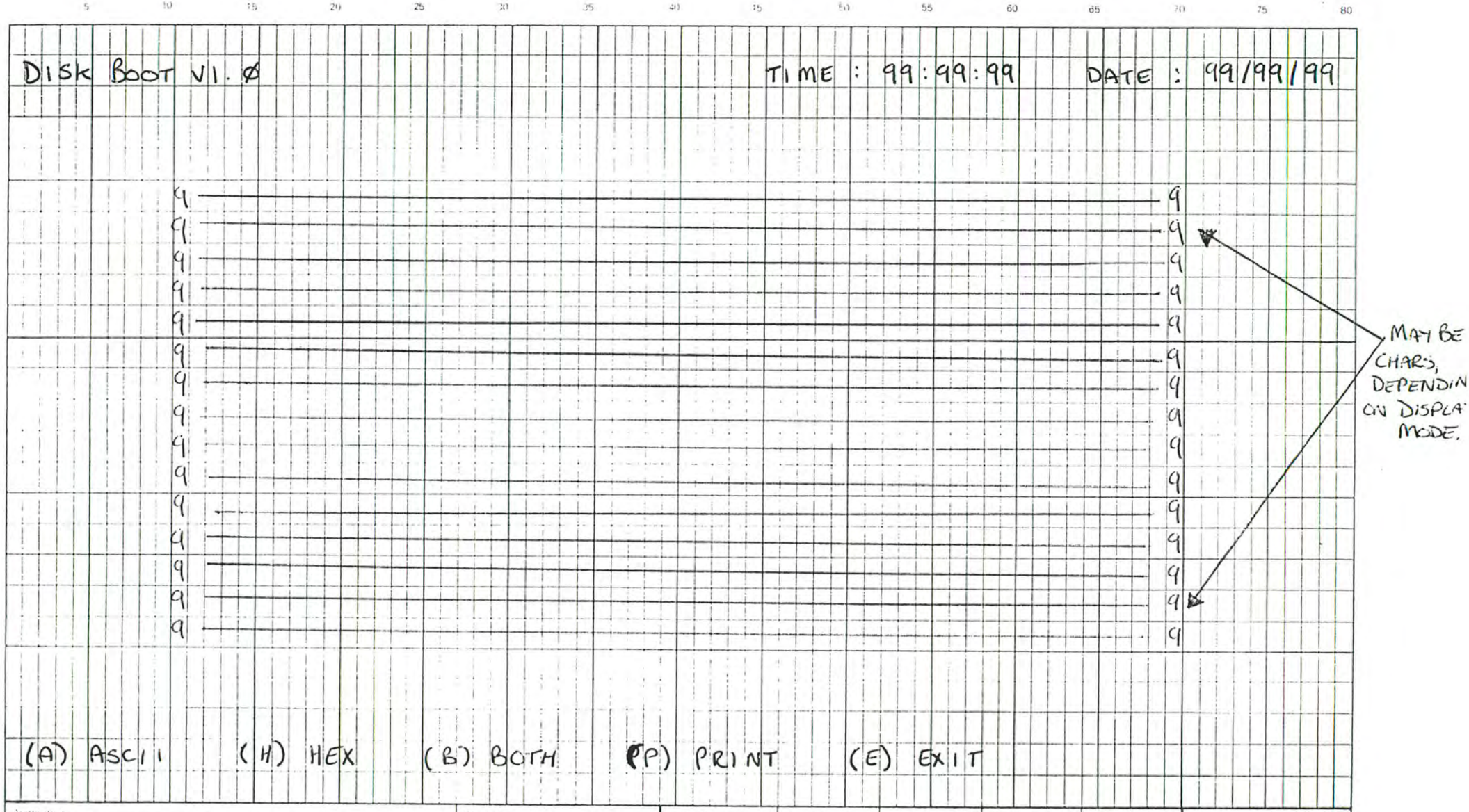
Application	Program	Characters to invest	Minimum	Maximum	Absolute	Average	Sheet
MAIN MENU	DISKBOOT						
Transaction name	Transaction code	Keystrokes					Issue
Designed by		Total letters, numbers, punctuation					Date

Visual display unit layout

```
DISK BOOT V1.0                                  TIME : 99:99:99    DATE : 99/99/99   INVERSED

DISK INFORMATION

NUMBER OF SIDES   9                                  NUMBER OF SECTORS : 9999
VOLUME LABEL      : X--------X.X-X                   NUMBER OF TRACKS  : 9-99
DISK SPACE FREE   : 999 k                            SECTORS PER TRACK : 99

BOOT INFOKMATION

BOOTSECTOR TYPE   : X-------------------------------------------------X
COMPARE RATIO     : FOUND 499 BLOCKS CORRECT FROM 512 (999%)
INFORMATION       : X-------------------------------------------------X
```

Application	IDENTIFY BOOTSECTOR/VIRUS	Program	DISKBOOT	Characters received	Minimum	Maximum	Absolute	Average	Sheet
Transaction name		Transaction type		Any strokes					Issue
Designed by				Characters transmitted					Date

Visual display unit layout

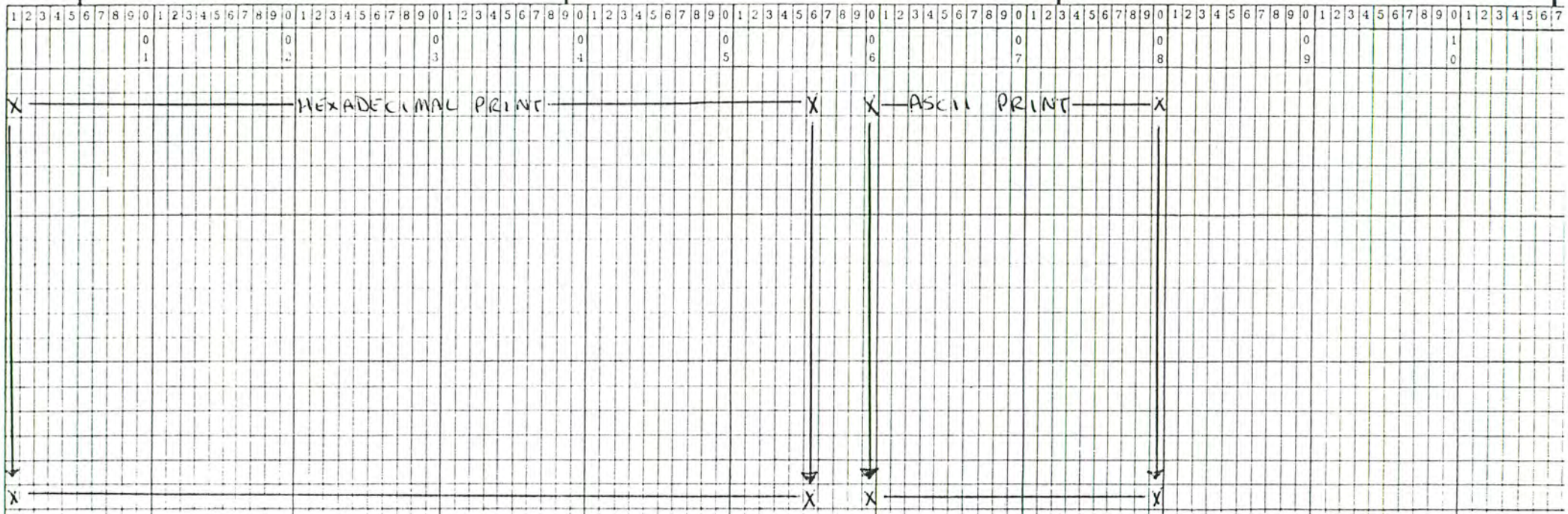


Application: EXAMINE BOOTSECTOR	Program: DiskBoot	Characters to insert	Minimum	Maximum	Absolute	Average	Sheet
Transaction name	Transaction code	Variations					Issue
Designed by		Characters to be stuffed					Date

PROGRAM/ DiskBoot
SYSTEM Examine BootSector.

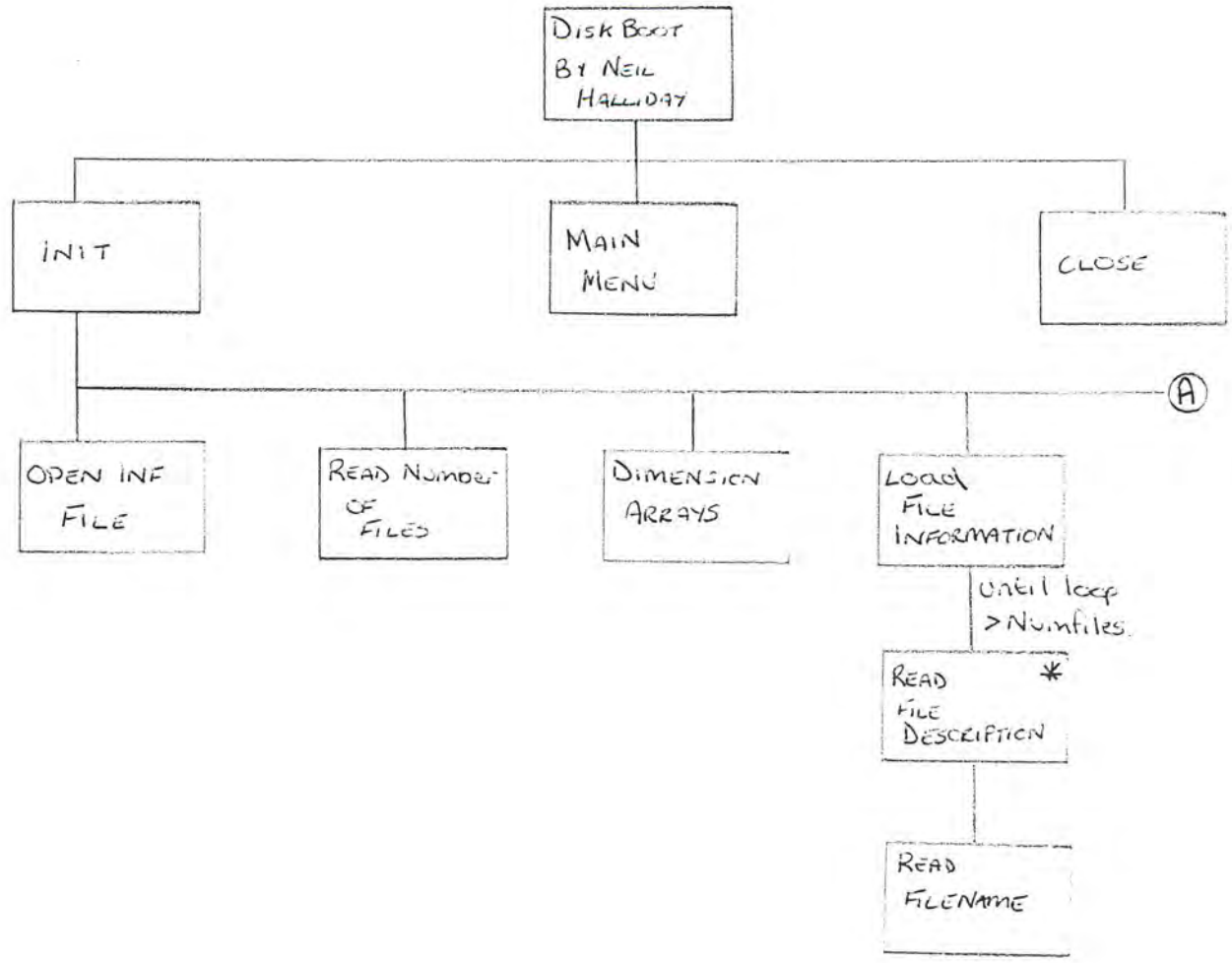
PRINT LAYOUT CHART

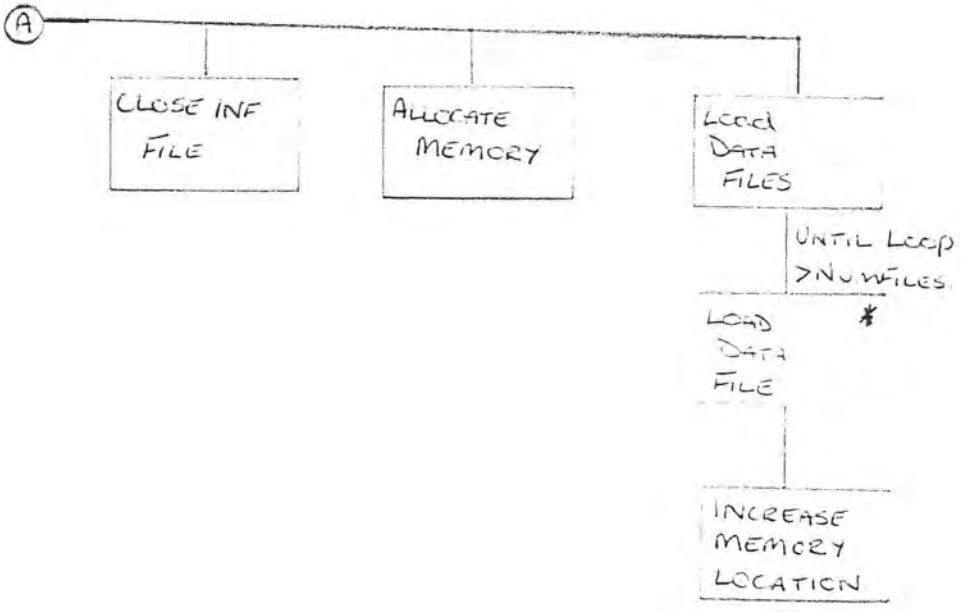
Design by



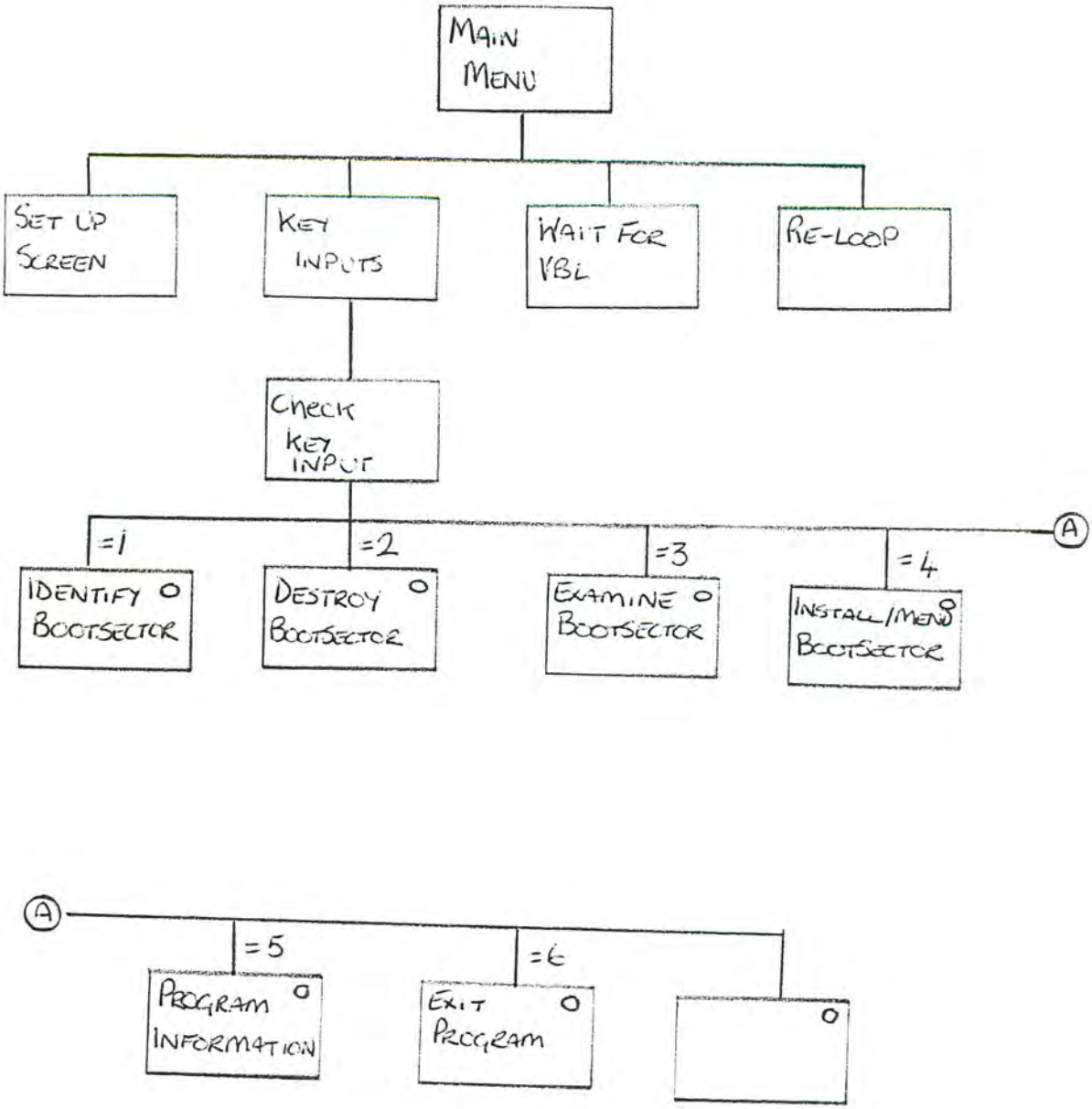
PRINTED OUTPUT GIVEN BY DISK BOOT V1.Ø ON 99/99/99 AT 99:99:99.

DOUBLE HEIGHT CHARS, ON EPSON PRINTERS

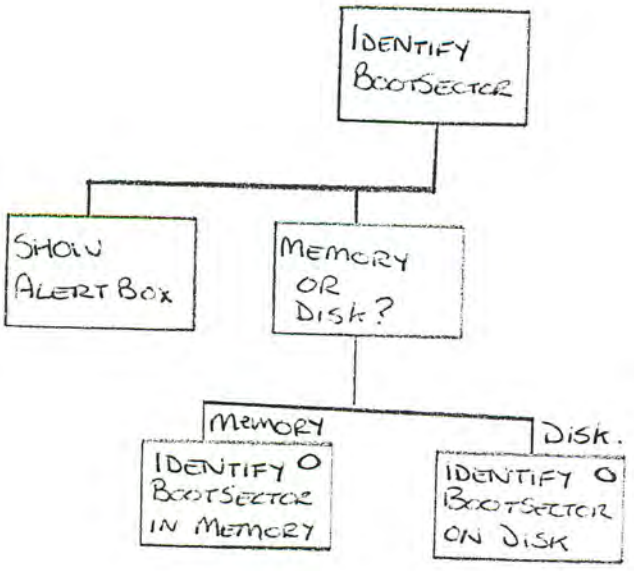


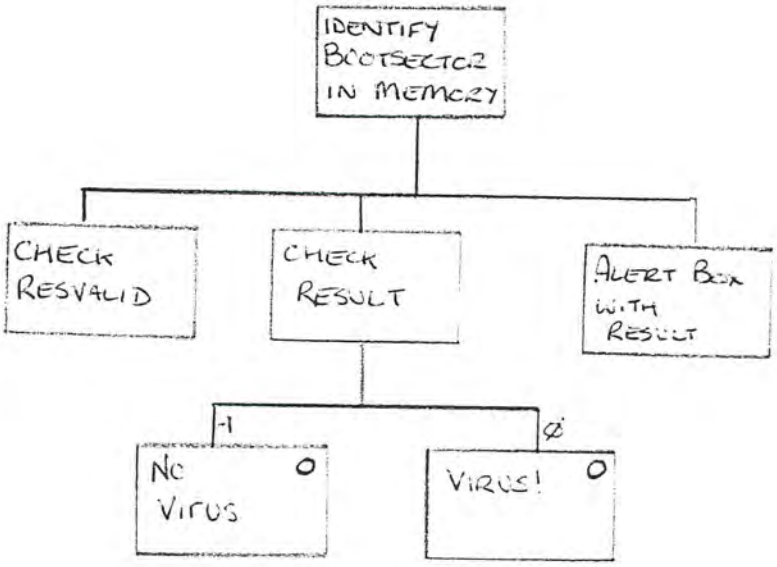


Appendix B - Structure Diagrams

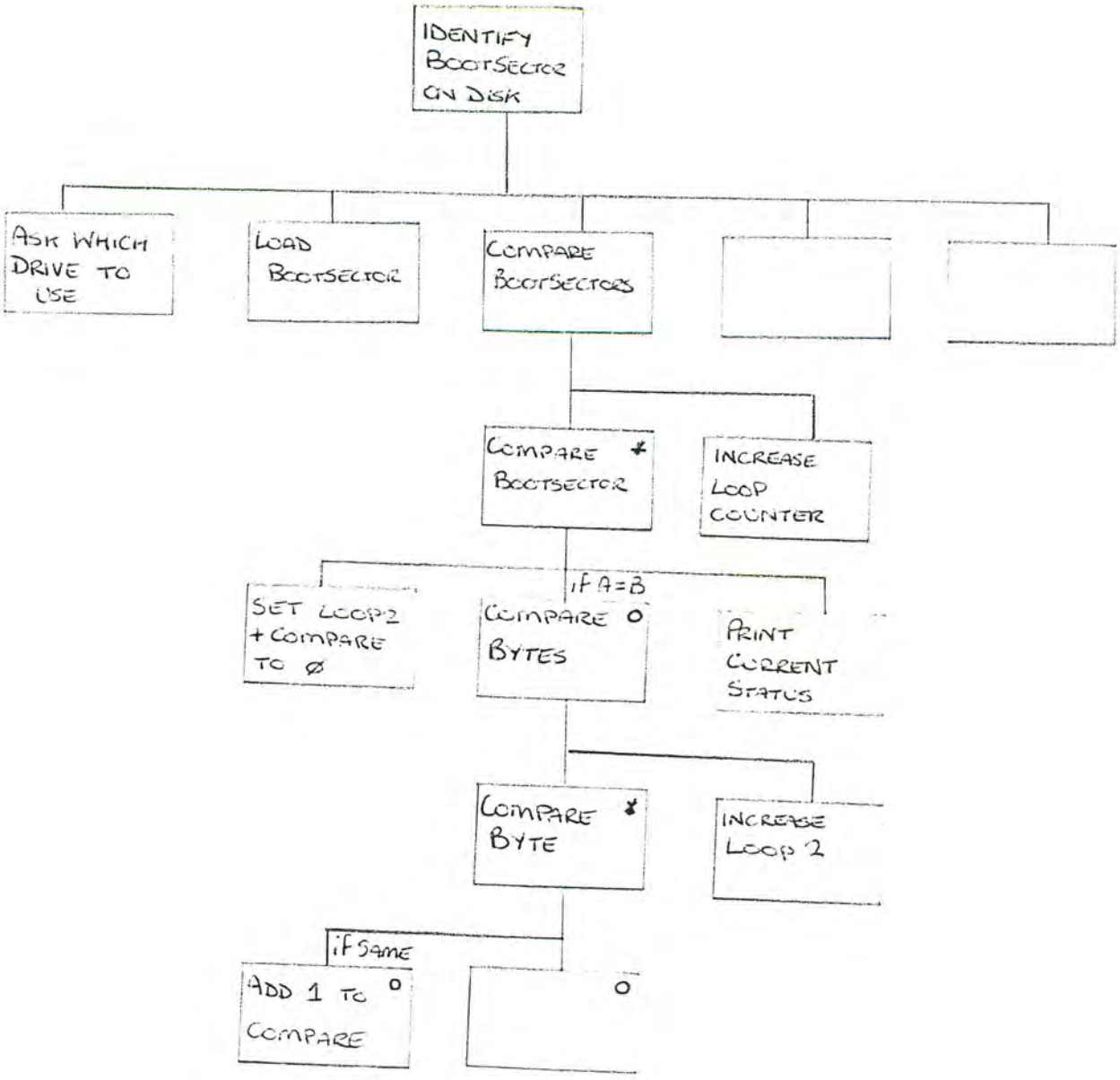


Appendix B - Structure Diagrams

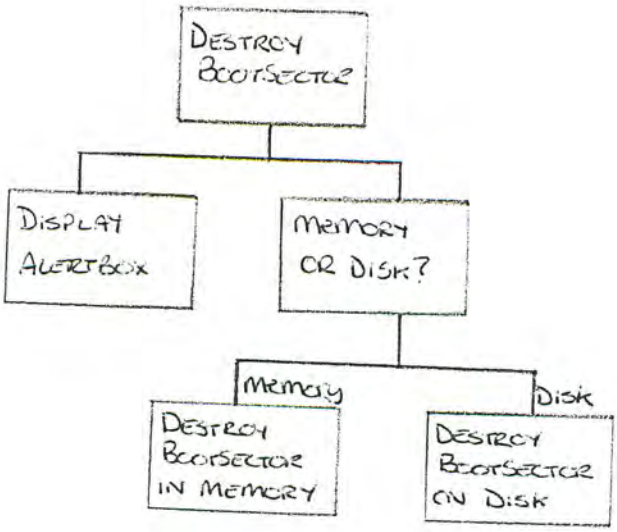


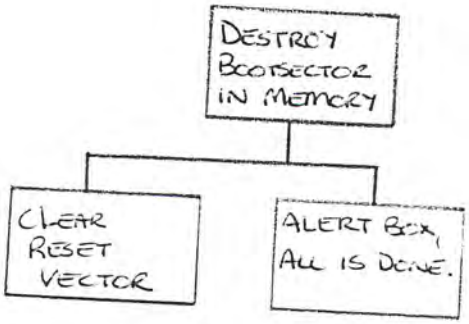


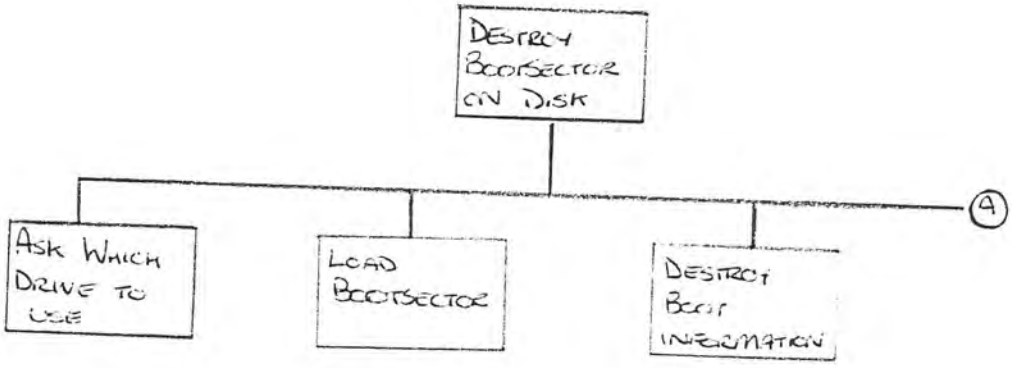
Appendix B - Structure Diagrams



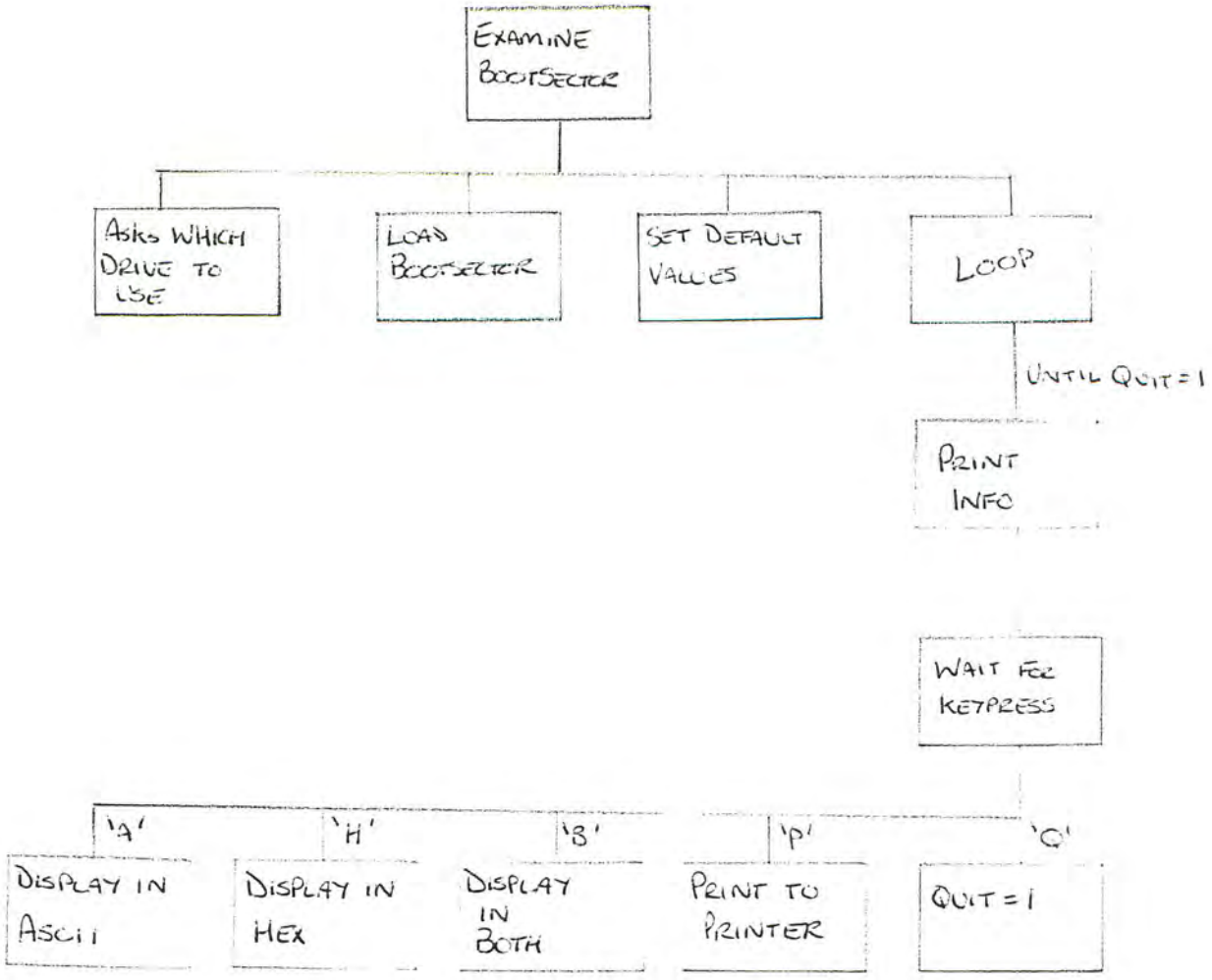
Appendix B - Structure Diagrams



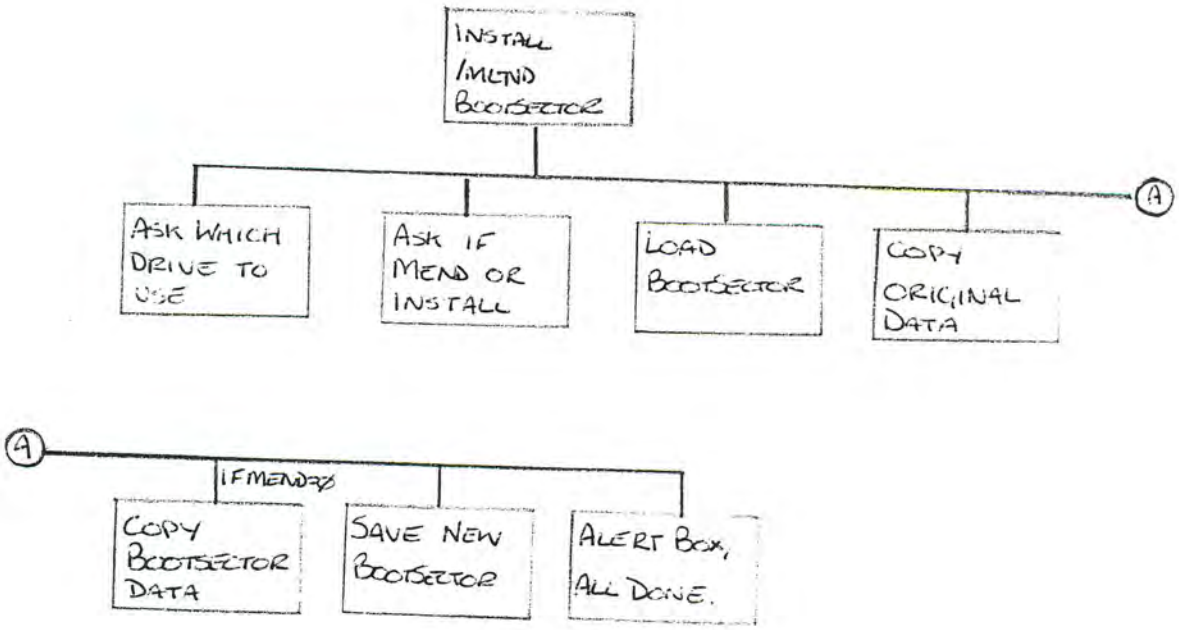




Appendix B - Structure Diagrams



Appendix B - Structure Diagrams



Appendix B - Structure Diagrams

The following is a list, with a small description, of the points that will be tested during the running of my program. Each list is in the sections of the program, e.g. init, main menu etc.

Init.

0.0 Pressing Ctrl C while running the program.

0.1 Loading the program in the wrong screen resolution.

0.2 Program cannot find data files to load.

0.3 Taking the diskette out of the drive, while the program is loading the data files.

0.4 Invalid inputs/key presses at the menu screen.

0.5 Pressing the 'HELP' key to turn the help option on & off.

Identify Bootsector/Virus.

0.6 Trying to select an option outside of the alert box.

0.7 Check for virus in memory works.

0.8 No diskette in chosen drive while loading a bootsector.

0.9 Calculation of number of tracks, sectors, sides, etc.

1.0 Program recognising bootsector/virus.

1.1 Compare ratio being correct.

1.2 Further information working correctly.

1.3 Pressing any other keys to return to main menu.

Destroy Bootsector/Virus.

1.4 Trying to select an option outside of the alert box.

1.5 Reset vector is cleared.

1.6 No diskette in chosen drive while loading bootsector.

1.7 Data is removed from bootsector.

1.8 No diskette in chosen drive while saving bootsector.

1.9 Trying to select an option outside of the alert box.

Examine Bootsector.

2.0 Trying to select an option outside of the alert box.

2.1 No diskette in chosen drive while loading bootsector.

2.2 Data is displayed correctly in ASCII.

2.3 Data is displayed correctly in Hexadecimal.

2.4 Data is displayed correctly in ASCII and Hexadecimal.

2.5 Entering invalid option to print menu.

2.6 Trying to print while the printer is offline.

2.7 Turning the printer offline while printing.

2.8 Printing data that contains printer control codes.

2.9 Exiting the print menu.

Install/Mend Bootsector.

3.0 Trying to select an option outside of the alert box.

3.1 No diskette in the chosen drive while loading bootsector.

3.2 Pressing a mouse key while the pointer is not over an arrow.

3.3 Trying to install a virus.

3.4 Pressing right mouse key.

3.5 Pressing left mouse key.

3.6 Pressing left & right mouse keys.

Program Information.

3.7 Pressing a key.

3.8 Pressing any mouse keys.

Appendix C - Testing Strategy

Exit Program.

3.9 Selecting an option outside of the alert box.

4.0 Selecting YES.

4.1 Selecting No.

Appendix C - Testing Strategy

The following, is the result of the testing strategy planned in appendix C. Each point has a description of the resulting output when the error occurred, I have supplied screen dumps where appropriate.

Init.

0.0 Pressing Ctrl C while running the program.

While running the program in the interpreter, doing this caused the program to BREAK, this problem was solved when the program was compiled.

0.1 Loading the program in the wrong screen resolution.

When this error occurred, my error handling routines took control, and the error message displayed was correct. Figure 0.1 shows a screen dump of the error.

0.2 Program cannot find data files to load.

This error again causes my error handling routines to take control, figure 0.2 shows a screen dump of this.

0.3 Taking the diskette out of the drive, while the program is loading the data files.

My error handling routine took control, see figure 0.3.

0.4 Invalid inputs/key presses at the menu screen.

This had no effect on the program at all, Mainly because the program code was only able to recognise correct inputs, anything else would just be ignored.

0.5 Pressing the 'HELP' key to turn help option on & off.

The default for the help option is on, so pressing the 'HELP' key when first loaded, turns the help option off. Pressing this key again results in the help option being turned back on.

Identify Bootsector/Virus.

0.6 Trying to select an option outside of the alert box.

Doing this results in nothing what so ever, as the program is told to ignore anything but a mouse click inside the correct boxes.

0.7 Check for virus in memory works.

When selecting this option and the reset vector is valid (not equal to $31415926) then the alert box displayed the correct message. When the rest vector was invalid (virus in memory) the alert box again displayed the correct message. See figures 0.4 & 0.5

0.8 No diskette in chosen drive while loading bootsector.

When this happens, the program just carries on as usual. This is because the routines that allow the disk sector accessing have not got disk error routines contained in them. This is a major problem in the language.

0.9 Calculation of number of tracks, sectors, sides, etc.

I tested around 10 diskettes, and each time the correct output was given. To make sure of this, I cross checked the diskettes with a public domain copier that I have.

1.0 Program recognising bootsector/virus.

I tested 10 diskettes on this function. Nine of the diskettes that I tried were held in Disk Boot's database, and one was not. All 9 of the stored bootsectors/viruses were recognised, and the other one made Disk Boot display this message "Unknown Bootsector."

1.1 Compare ratio being correct.

This function was fully operational.

1.2 Further information working correctly.

I tested this function with complete bootsectors, and bootsectors that I had altered using a disk editor. When the unaltered bootsectors were tested, Disk Boot said the disks were 100% virus free. When the altered bootsectors were tested, Disk Boot told me.. "Bootsector is not 100%, may be damaged somewhere". When a diskette containing a virus was tested Disk Boot told me.. "DESTROY IMMEDIATLY!".

1.3 Pressing any other keys to return to main menu.

Only by pressing RETURN or by pressing the left mouse button, could I return to the main menu. All other inputs were ignored.

Destroy Bootsector/Virus.

1.4 Trying to select an option outside of the alert box.

See *Identify Bootsector/Virus* 0.6

1.5 Reset vector is clear.

I loaded the program with a virus in memory. I then destroyed the virus in memory, and when checked with the identify function, I was told that the reset vector was valid, and there were no programs residing on it.

1.6 No diskette in chosen drive while loading bootsector.

See *Identify Bootsector/Virus* 0.8

1.7 Data is removed from bootsector.

After destroying all the information from the bootsector I used my examine function, and all that was on the diskettes bootsector was the first 29 bytes of file data. Everything else was full of zeros. See figure 0.6

1.8 No diskette in chosen drive while saving bootsector.

See *Identify Bootsector/Virus* 0.8

1.9 Trying to select an option outside of the alert box.

See *Identify Bootsector/Virus* 0.6

Examine Bootsector.

2.0 Trying to select an option outside of the alert box.

See *Identify Bootsector/Virus* 0.6

2.1 No diskette in chosen drive while loading bootsector.

See *Identify Bootsector/Virus* 0.8

2.2 Data is displayed correctly in ASCII.

See figure 0.7

Appendix D - System Testing

Examine Bootsector. (Cntd..)

2.3 Data is displayed correctly in Hexadecimal.

See figure 0.8

2.4 Data is displayed correctly in ASCII & Hexadecimal.

See figure 0.9

2.5 Entering invalid option to print menu.

Nothing happens.

2.6 Trying to print while the printer is offline.

My error handling routine takes control, and displays printer not ready error. See figure 1.0

2.7 Turning the printer offline while printing.

Most printer buffers are more than 2k, the data that is sent to the printer is no more than 1.4k, so the data just gets sent into the printers buffer.

2.8 Printing Data that contains printer control codes.

This created a major problem, as the control codes made the printer come out of the condensed mode, making the text too large to fit onto the page. I only had this problem occur when using one specific bootsector type. See figure 1.1

2.9 Exiting the print menu.

Not allowed to exit until the printing has stopped.

Install/Mend Bootsector.

3.0 Trying to select an option outside of the alert box.

See *Identify Bootsector/Virus* 0.6

3.1 No diskette in the chosen drive while loading bootsector.

See *Identify Bootsector/Virus* 0.8

Appendix D - System Testing

Install/Mend Bootsector. (Cntd..)

3.2 Pressing a mouse key while the pointer is not over an arrow.

If I pressed the left mouse key it had no effect, but pressing the right mouse key exited the function.

3.3 Trying to install a virus.

See figure 1.2

3.4 Pressing the right mouse key.

This exited the function.

3.5 Pressing the left mouse key.

This had no effect.

3.6 Pressing both mouse keys.

This installed/mended the bootsector.

Program Information.

3.7 Pressing a key.

Doing this took us back to the main menu.

3.8 Pressing any mouse keys.

This had no effect.

Exit Program.

3.9 Selecting an option outside of the alert box.

See *Identify Bootsector/Virus* 0.6

4.0 Selecting YES.

Left the program.

Appendix D - System Testing

Exit Program. (Cntd..)

4.1 Selecting NO.

Took me back to the main menu.