U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

A method for the recovery of data after a computer system failure: The development of Constant Random Access Memory (CRAM)® recovery system

Brevett, Renford Adolphus Benito, Ph.D.

Iowa State University, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A method for the recovery of data after a computer system failure: The development of Constant Random Access Memory (CRAM)® Recovery System

by

Renford Adolphus Benito Brevett

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirement for the Degree of
DOCTOR OF PHILOSOPHY

Major: Industrial Education and Technology

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1992

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I. INTRODUCTION

Over the past decade, much computerization has occurred in industries, schools, and homes. Computers are being used as tools, tutors, and tutee. Many people have become dependent on computers to perform a variety of calculations, bookkeeping, word processing, and other tasks. In order for the computer to reliably and continuously perform its assigned tasks the hardware and software must be designed to operate with a high degree of reliability. Since modern computers are generally reliable, most improvements by computer designers and programmers are directed towards building faster machines and higher storage capability. Although the storage capacity and speed of the machines is important, the designers neglect the area of system recovery and data loss. Data loss from a power outage can be catastrophic in that very important information, which may either be very expensive or time consuming to obtain may be lost. The following questions could be posed:

1. What are the major causes of computer system failure?
2. What methods can be used to save the system's data for easy access in case of a system failure?
3. How can data be recovered from a failed system?
4. What are the costs associated with system recovery by alternative methods?

5. What measurable effects do alternative systems have on system performance?
6. Can a recovery system be developed that utilizes only software in contrast to hardware solutions?
7. What are the major problems to be encountered in developing a software-based recovery system?

Statement of the Problem

Regardless of the computer's general reliability, unfortunate incidents, such as keyboard lock-up, software failures, and power outages, over which the user has no control, sometimes lead to the loss of data in memory. Most existing software products do not have the capability to restore data lost from memory. The user is compelled to either develop habits of frequently saving their data or repeat the work if there is a failure. Computer failures occur in a learning situation or from power interruption. The new computer student or new software user may make incorrect inputs or connections of peripherals. This can lead to a computer failure or interruption. It is important that an inexpensive method be developed for use in such situations in order to restore the system from a failure. This project will investigate the development of a software solution to restore the system from an interruption. The approach used employs a hard disk drive for memory storage. To be considered a practical recovery system, it should have little effect on the execution time of other applications running in the system. This study raised three basic questions:

1. Can a software system be developed that will provide recovery from a system failure?
2. What problems exist in achieving a software-only recovery system?
3. What is the degradation in application program performance when utilizing a software recovery system?

Purpose of the Study

The purpose of this study is twofold. First it will concentrate on development of a software based solution to a computer system recovery. The goal is to reduce frustrations, delays and industrial process down-time.

Secondly, it is the intention that this project will demonstrate the feasibility of incorporating system recovery methods directly in the design of the computer's operating system and, by doing so, reduce the total cost of computer systems with high reliability.

Need for the Study

Attempts have been made by some computer system designers to use Uninterruptible Power Supplies (UPS) or a fast switching power generator, to supply electricity when the power is off, to help in system recovery. Another method involves periodic "dumps" of files to the tape or disk for use in the event of a failure. A third method is the use of static memory that can retain data when power is turned off. These methods can be used to recover some of the system data if there was a power failure. Also according to Beizer (1988) some UPS do a fine job of switching from the power line to the battery and keeping things going during blackout, but when the power is restored, they may throw in their own nasty piece of electrical noise, thereby creating the very same problem one was trying to avoid. There are, however, many other reasons that a system may need to be recovered, such as keyboard lockup, memory parity error, failed UPS, and badly behaved software. It is the belief of this investigator, that the added expense of a UPS system and/or a PC board with static memory can be avoided. This study will investigate a method to accomplish a system recovery using software and the available hard disk subsystem. This method would be inexpensive and utilize exiting parts of the system. The experimental system developed in this study will be referred to as the CRAM (Constant Random Access Memory) Recovery System.

Delimitations of the Study

To develop a system that is portable and compatible to all existing computer systems would take a very long time and may even be impossible due to the rapid change of operating systems. The following delimitations were imposed to speed up development and testing time:

1. The software was designed to run on IBM and compatible microcomputers with the following specifications:

   a) Only the IBM-PC XT version is fully supported.

   b) The system memory is 640k.

   c No extended or expanded memory was used in the test version.

   d. The video modes supported are MDA, HGA, CGA, & EGA.

   e. Graphic modes of PGA, VGA or above are not supported.

   f. The operating system used was DOS 3.x (DOS 4.0 or above were not supported in the test version).

No attempt will be made to generalized to other processing units operating under different conditions.

2. Time did not permit field testing of the software so testing of the software was done only by the researcher.
3. The programs used for benchmark testing were limited to eight readily available programs and DOS utilities. Any number or kinds of programs could be used but the ones chosen are frequently used by computer operators and thus likely to be the kind encountered by the CRAM system when installed.

Limitation of the Study

Information about some DOS functions are not available and thus may affect performance of the CRAM system, if changes are made to those functions in future releases of DOS.

Procedure of the Study

The study was conducted in the Department of Industrial Education and Technology at Iowa State University. The literature was reviewed to identify the current methods used, if any, for system recovery. The various methods of system backup such as use of UPS, tape backup and software timed backup were investigated. Procedures for system recovery were also studied. At the time of this study no method was found that could provide system recovery from a power outage without the use of a UPS system. It is the hypothesis of this study that the system can be recovered by using software, thus eliminating the added cost of a UPS. The software would also have the added advantage of being able to backup and restore the system without an action by the user. Some UPS systems require the user to be present to save or backup the system within a limited time.

The experimental design required development of software to

a. install a program in the computer that will monitor and backup the system at either a predefined interval or intervals depending on usage,

b. attempt to detect internal and external system failure,

c. upon detection of such failure, the program will attempt to close down the system gracefully and warn the user, and

d. restore the system to a state prior to the failure.

Following development of the experimental software, it was tested for reliability and validity. The ability of the program to restore the system using different applications determined the reliability of the system. Validity of the system was determined by examining the degradation in the system's performance using representative benchmark programs.

The theoretical basis for the experimental software development are discussed in chapter 2 and include:

1) Computer system and software failures.

2) Software testing methods.

3) Computer reliability and validity testing.

4) Operating system and software development.

5) The use of Terminate-and-stay-Resident (TSR) software.

6) The use of Undocumented Disk Operating System functions.

7) Currently available data recovery systems and problems of data recovery.

8) Systems development methods.

Definitions of Terms

ASCII code
American Standard Code for Information Interchange. Most commonly used method of encoding the alphabetic and numerical characters into bits.

Chip
An electronic component inside the computer that is identified by many pins projecting down from its sides into a socket.

Context Switching
The process of rapidly switching one processor among several operations to give the illusion of concurrent processing. This applies also to a simple switching of the process to execute a different process from the current one.

CRAM
Constant Random Access Memory. This is the area where memory is saved for restoration on the hard disk. The software for this project is also referred to by this name.

Crash
A computer is said to crash when its software or hardware caused the system to be so confused that it no longer performs useful functions.

DPT Disk Parameter Table. A table of values that describes the layout of the hard disk.

DTA Data Transfer Area. A memory buffer used by a program to temporary store data to be sent to the disk or read from the disk.

FAT File Allocation Table. A table of numbers found on every MS-DOS disk which tells the status and use of a section of the disk.

Firmware Software not intended to be modified and is electronically programmed into a chip inside the computer.

Interrupt (INT) Routines inherent to the operating system that can be accessed by other programs and are activated by hardware or software.

IRQ Interrupt Request. Interrupts generated by external devices and are tied directly to a pin on the CPU.

Lock-up A computer crash when the keyboard will not accept any input.

MCB Memory Control Block. MS-DOS divides the first mega-byte of memory into contiguous blocks with a paragraph of control codes at the beginning of each.

PSP Program Segment Prefix. A collection of unrelated data elements required by programs.

SDA Swappable Data Area. This is a portion of memory used by DOS to store the current context of the system.

Sector A formatted disk is divided into circular regions called track. The track is subdivided into sectors. Each sector holds the minimum block of data DOS writes to the disk (current minimum is 512 bytes).

TSR Terminate-and-Stay-Resident. Program that stays in memory and operates in the background.

UPS Uninterruptible Power Supply. Supplies the computer with electricity, from recharged batteries, for a period of time after a power failure.

CHAPTER II. LITERATURE REVIEW

Computer reliability, the causes and solution to hardware and software failures and, data loss and procedures to restore data are all current research topics (Levy & Silberschatz, 1991). Industry is making demands on developers to develop fail-safe systems by introducing recovery systems, such as a resource recovery system (Bacon, 1991; Klopp, 1990; Maslak, Showalter & Szczygielski, 1991; Tam & Hsu, 1990) and file recovery system (Barnes, et al., 1991). These recovery systems could use the cache technology for fast transfer and transparent operation (Grossman 1985; Rich, 1986 & Sando, 1985). The current state of the art involves various backup techniques that are found to be either relatively slow or unreliable (Kaczeus, 1990). Some of the constraint on developing this type of system is speed of operation, automation and reliability.

Researchers in this area need to rely on current developments and have an aptitude to peruse many unpublished and unrefereed literature sources to amass information that will keep up with current findings. Currently government and business organizations are studying the problems that will be encountered towards the end of this decade, and which is believed will involve the storage of gigabytes (GB) of data per day (Lopez, 1992). Lopez states that if problems like these are to be solved by the turn of the century, it is important to have a knowledge of the current and advancing hardware and software technologies for archival, retrieval, and distribution of large volumes of data. The prediction by Mason (1984) and Crecine (1986) was that computers would be used more intensely as the technology increase.

The approach used to review the literature for this type of experimental design was examine previous work done in related areas such as

a. Computer system and software failures,

b. software testing,

c. computer reliability and validity testing,

d. operating system and software development,

e. terminate-and-stay-resident (TSR) method,

f. undocumented DOS functions, and

g. data recovery system.

Computer System and Software Failures

According to Chantico (1991) the automation of daily business has brought about an element of computer dependence. The loss of data and the annoying unscheduled down time can be very expensive. For educators' computer lockup, system failure or power failure can be very frustrating and embarrassing at times. In order to develop a fail-safe system one needs to know the environment in which that system exists and what causes the system to fail. There are many different ways to classify failure in computer systems. Transient errors have been estimated to occur at the rate of 5 to 100 times that of permanent failures (Adams, 1991). Researchers are always investigating methods for error detection (Spector, 1984). Tasch (1990) identify humans as important in error detection. According to Tash, failure detection has been recognized as one of the functions where computers and human operators can complement one another. Bondavalli (1990) discussed the classification of computer failures. He outlined three major guidelines to computer systems failure classification:

1. A failure classification should be able to be applied to every system.
2. The classification must be as detailed as possible.
3. The treatment of failures following a detection, are to be considered as a second step.

Classification of failures has one major advantage of characterizing systems with respect to their failure modes, that is providing designers with the a way to choose the most appropriate detection techniques for each particular system.

A summary of reasons for system failures, from Bondavalli (1990), Lua (1990) and Garcia-Molina (1990) includes:

- *Inadequate input validation*: This occurs when the software allows an illegal data input to slip through and sent for processing.

- *Design miscalculation*: This is the inadequate safety margins built into the system to cope with exceptional conditions.
- *System control faults*: This may result from faulty logic in either the operating system or the user's own control software.
- *Hardware faults*: Any component of the system may develop a fault.
- *Software faults*: Programs are written by people and prone to fault at any stage. The longer and more complex a program is, the more faults it is likely to have.
- *Human mistakes*: The major source of error is the human operator. Faults such as using the wrong version of a file or program, responding incorrectly to console messages from the operating system or user programs.

Preventing failures is much less costly than correcting errors in the system after a failure. Measures have be taken to minimize the occurrence of failures. Methods used to prevent errors include:

Surge protector: These devices are designed to stop spikes that occur in the power line from lighting bolt or sudden voltage change in the power line.

Power line filter: These are like surge protectors but generally more expensive (Mace, 1988). They smooth out peaks and valleys in the voltage which can affect the performance of the computer or lead to random failures.

Uninterruptible Power Supply (UPS): Alternative methods for supplying uninterruptible power have been sought for the past five years. A UPS is a battery system that provides power to the computer for 10 minutes to an hour after the power goes out (Mace, 1988). The governing factor in the type of UPS is the Ampere/hour rating. The UPS needed for a system depends on the power requirements of equipment that will be plugged into the UPS, and how many boards are installed inside each computer. Proper backup procedures and the use of Uninterruptible Power Supplies (UPS) are some of the many measures taken by industries to curb this problem (Mace, 1988). One example uses a UPS on a plug-in board inside the computer. A further development of this product was made to include the saving of the system to the hard disk.

ITT Power Systems developed a similar product called the PowerSave 500. According to Harold Ramsey (1992), systems developer at ITT, the major differences between these two products are the way in which the system is interrupted and the amount of memory that each program occupies when loaded into the system. Ramsey, the major technical designer of the PowerSave 500, said the use of Interrupt Request (IRQ) and the ability to use its own on board memory makes his product more efficient. There are also differences in the ability of each product to capture extended and expanded memory. These systems can be very expensive, bulky, noisy and inconvenient.

A recent development is a battery-free rotary options for UPS system (Lengefeld, 1990). This system described by Lengefeld can provide up to one minute of uninterrupted power without the use of batteries.

Data integrity and the ability to restore the computer to its prior state after a system lock-up or power outage are of increasing concern to end users both in education and industry. Some of the methods used to save data for future retrieval are:

Floppy backup: This is the storing of information in a compressed form on several floppy disk. Most fast backup programs fail at times. Mace, in his book said "I have never had a floppy backup program that didn't fail me." These programs use direct memory access (DMA) transfer. DMA refers to chips inside the computer that can send data to and from memory without using the central processing unit (CPU) (Tammaru, 1985). This means that information can be sent to the diskette while the program is requesting input from the keyboard.

Bernoulli cartridges: This provides storage on removable cartridges and are close to the hard disk in performance. The cartridges can wear out when used on a daily bases.

Tape backup: Magnetic tapes that are similar to audio tapes are used to store data. This method is common for use with local area networks. This method of backup, which stores all files on the hard disk to the tape is slow. The backup procedure is often done at the end of the day, month and year. A separate tape is usually used for each day. According to Beaudin (1992), the older system are unreliable and tends to cancel out very often. A new technology is growing in this area using 4mm digital audio tapes rather than the old reel to reel and the more expensive 8mm tapes.

Periodic save by some software: This is the saving of the data files that the program is using to the disk. Some word processors like WordPerfect use this method to keep a running journal to the disk of the activity of the file that is being edited.

Disk Mirroring: This method uses two similar disk to save identical information. Both disk are updated at the same time.

Redundancy in system: Many database systems according to levy (1990), Lucente (1991) and Yanney (1986) increase reliability by employing a fair amount of system redundancy. This involves both software and hardware redundancy. More than one copy of the system are kept at remote sites in case of disaster or system failure.

The choice of the system to implement will be determined from a careful procedure of software testing.

Software Testing

One of the major areas of software development that is becoming of great importance is Software Testing (Gary, 1986; Han, 1986; Kinoshita & Saluja, 1986). Hetzel (1984), in a quote from the preface to Glen Myers, Art of Software Testing, noted that approximately 50% of the elapsed time and over 50% of the total cost are associated with testing a program or system being developed. There is however, a fuzziness in the measurement of computer systems. Software testing therefore, includes various definitions such as: executing a program or system with the intent of finding errors, verifying that the software satisfies specified requirements, identifying differences between expected and actual results, and evaluating an attribute or capability of a program or system. Hetzel further emphasized that testing could be divided into three groups and various sub-groups thus forcing quality to become tangible and visible as the final outcome to testing. The three groups and corresponding sub-groups are shown below:

Functionality (exterior quality)

Correctness

Reliability

Usability

Integrity

Engineering (interior quality)

Efficiency

Testability

Documentation

Structure

Adaptability (future quality)

Flexibility

Reusability

Maintainability

It was also noted that any proposed testing methodology must provide a means of answering the following major questions:

1. What should be tested?
2. When should testing stop?
3. Who does the testing?

Falk (1987) states that bug hunting is a serious business, however it is an unfortunate fact that it's impossible to eliminate every bug from a program. He further notes that software developers can now turn to specialists such as AGS Information Services (Cincinnati, Ohio) and Programming Environments (Tinton Falls, New Jersey) for software testing.

Computer Reliability and Validity Testing

The need for reliability and validity testing of computer systems is a major concern to industries (Daniels, 1987). He further noted that the safe operation of computer systems, in both their software and hardware continues to be a key issue in many real time applications, when people, environment, investment or goodwill can be at risk.

The majority of Software designers and users are from a wide range of backgrounds such as computer science, engineering, mathematics, or physics, which leads to differing views on the usefulness and applicability of software testing. Daniels cited two major approaches: the computer science approach and the engineering modeling approach. The computer science approach aims at achieving logically correct and error-free software by stringent testing of competently produced code. The engineering modeling approach, on the other hand, seeks to model the failure mechanisms in software with a view to fitting the model to a developing body of code and thereby making a quantified assessment of its reliability. It is the latter approach that was taken by this project and thus the role of statistics in software reliability was assessed. Reliability in a software context remains controversial both in human factors applications and in computer software science. Some forms of reliability include: *producers perceived reliability*, which is the quantification somehow arrived at by the producer; *user perceived reliability*, initially based on that of the producer, modified subjectively according to past experience; *inherent reliability*, a true, but unknown, measure of the closeness of the software to an ideal version; *in-use reliability*, a true, but unknown, measure of the extent to which the software will perform correctly in the user environment; *adaptive reliability* which is like in-use reliability except that the user adapt their behavior, i.e. modify the user environment to side-step deficiencies in the software which might otherwise be reported as faults.

Koren (1986) pointed out that the most obvious characteristic of a reliable system is the time the system is available and operating normally. Reliability and integrity, as summarized from Beaudin (1992), Belli (1991), Kaczeus and Lion (1990) has six contributing factors:

(1) Availability can be measured and thus is a contributing factor to reliability. Availability is the proportion of the total time scheduled for operation that the system is actually available for normal service, and expressed as:

$$Availability = (MTBF)/(MTBF + MTTR)$$

- Mean Time Before Failure (MTBF)
- Mean Time To Repair (MTTR)

(2) Graceful degradation of the system happens when failed components are in the system but operation continues in a restricted mode. It is preferred to have a degraded service rather than a collapsed system.

(3) Fail-safety is the preclusion of certain potentially disastrous events from occurring in the system.

(4) Data integrity is the ability of the system to prevent errors in its data-base, to detect them as early as possible and correct them or confine their effects.

(5) System integrity is the ability of the system to detect faults in its own operation, and to correct them at least to limit the damage they cause.

(6) Recovery capability is the most important factor of all in reliability and integrity; whatever kind of difficulties the system gets into, it must be possible to get it out again, in a reasonable time and at acceptable cost.

Koren (1986) and Belli (1991) suggested the introduction of redundancy into the system to improve reliability and availability. The suggestion was to use instruction retries and program rollbacks. Koren further noted that several researchers have analyzed recovery technique and all differ in their assumptions and objectives.

Data Recovery System

The mechanism and responsibility for recovery are distributed to many different programs, files, hardware devices, procedures and people. Gibbons (1976) suggested five subsystems which are still considered today in systems development:

System supervision: Monitor the system to detect errors as early as possible, before they cause serious damage. When an error is detected it may inform a human operator or take action to correct the error.

Activity recording: Information about the processing activity of the system must be recorded, to support the recovery functions. Periodic 'snapshots' of the system are placed on recorded journal in the form of file dumps or checkpoints. The record of the changing state of processing provides a basis for recovery from future failures; it may also be useful in diagnosing the cause of the failure.

Investigation and decision: Error information is important and should be collected and analyzed, with a view to improve the system to prevent its recurrence. It is also important to collect data on the performance of the recovery programs and procedures, both on their speed and on their effectiveness.

Repair and recovery: The aftermath of a computer system failure is a plan for recovery. A combination of human effort and computer procedures is put into effect to design a plan for recovery. This include reformatting, repairing and altering of both hardware and software.

Maintenance and improvement: Errors are diagnosed to obtain information about the cause of failure. An improvement on the system, using the historical information will attempt to prevent the failure from recurring.

Johnson (1991), Bennett (1991) and Upahyaya shared the view of using rollback recovery system to restore system failures. Information is stored on stable storage media during failure-free execution, allowing certain states of each process to be recovered after a failure.

Many data recovery systems encountered are geared at disk file recovery. There are two assumptions that were made by developers: the users will have access to the tools and know how to use them, and that the operating system and application software will maintain a certain level of standards. Both assumptions were endorsed by the kind of software supplied with the operating system such as COPY, CHKDSK, RECOVER, DEBUG, BACKUP, and RESTORE, and the available third party software. Even more recently the introduction of MIRROR, REBUILD, UNFORMAT, and UNDELETE were added commands to DOS version 5.0. Even though some are useful, the DOS utilities are usually poorly documented and users are unaware of their potential. Mueller (1991) states that many users are not aware that CHKDSK command, which is usually used for simple inspection of a disk file structure, can be used to repair a damaged file structure. He also noted that users do not know that the COPY command can be used for recovering from file or file system (file allocation table [FAT] and directory) damage.

Several levels of disk or data damage may exist. Only two primary levels can be recovered by present data recovery techniques: Unreadable sectors and Corrupted sectors.

Unreadable sectors are caused by physical damage, or magnetic damage. Physical damage is irreversible while magnetic damage is reversible. In both cases, however, data is lost in these sectors. Corrupted sectors are sectors with corrupted data (data with invalid information or data not linked to any part of the file system) that can be restored to the original condition by DOS utility programs or a third party disk recovery software.

These products are also faced with one other major problem-that of the changing Disk Operating System (DOS).

Operating System and Software Development

The operating system is a set of software tools designed to make it easy for people and programs to make optimum use of a computer (King, 1988). All computers must have an operating system installed in order to run user's programs. MS-DOS is an operating system for 16-bit, 8086/8088-based computers. DOS is an acronym for Disk Operating System. DOS was designed with the assumption that files will be found on the disk and that a disk is needed before useful work can be done. Since the introduction of MS-DOS in 1981 there has been enhancement to accommodate new hardware environments, fix problems, and generally improve its operation (Angermeyer, 1986). These enhancements, although resulted in more powerful capabilities, produce incompatibility with older versions.

The introduction of DOS version 5.0 has introduced an even more difficult problem-that of knowing where in the memory space important information lies. The use of the so called high memory area, memory above 640k, kept many board and software designers busy and left them puzzled about how to design for the future. Most new products are designed for the new, more powerful 286, 386, 486, etc., and what ever the next generation of computers turns out to be. Many schools and home computer users are using the IBM-PC XT or compatible and are thus left out when new designs are introduced. This design project utilized the IBM-PC XT. The method selected to provide system recovery in this study seems to pose some interesting problems according to Jim Kyle (1992), co-author for the book "Undocumented DOS". He said in his electronic mail:

> This principle is simple enough, but the problem is that the memory you try to save and restore ALSO includes the program and variables that are controlling the save process, so when they get restored they will suffer amnesia, forget that they are restoring, and start saving again (p. 1).

The technique used in this study attempts to alleviate that problem. A program is needed that can reside in memory, save and restore memory without affecting its own code. The DOS operating system, running as a single user single task system supports two ways to accomplished this task: using a terminate-and-stay-resident program, and using a device driver.

Terminate-and Stay-Resident (TSR) Method

A TSR program has the ability to operate in the background while other programs are running in the foreground. According to Boling (1992), TSR programs have been the mainstay of the DOS operating system since its introduction in DOS version 2.0. The main problem with a TSR program is that it robs the system of some memory and thus may not be well behaved with memory hungry software. Boling noted that:

> TSRs should be totally unobtrusive to the system and at the same time available to the user. This contradictory goal causes TSR programmers to jump through what seems to be endless series of hoops that are

> necessary for compatibility with an operating system not originally designed to accommodate TSRs (p. 44).

The problem of compatibility will soon be overcome with the emerging standards proposed by the TSR programming community. According to Wadlow (1987), Borland International, one of the first companies with a commercial set of resident applications, has proposed a standard that has been well received in some places.

Steven Baker (Waite Group, 1988), differentiates two types of TSR programs. First, there are the simplest extensions to MS-DOS that extend the hardware features and do not need to use any MS-DOS function calls. Once resident, this type of TSR makes no DOS function calls.

The second type of TSRs is more complex in that, once resident, it must make DOS function calls (such as disk I/O). The difficulty in using TSRs is determining when DOS or an application program can be interrupted. This is a major problem because MS-DOS function calls are not reentrant or recursive, i.e., one cannot make several calls within calls. If a TSR interrupts a MS-DOS function call that is in progress to make another MS-DOS call, the first call will be trashed and lost with a crash of the system or other unpleasant result.

The second method to launch a program that can work in the background is using device driver. This method differ from TSR only in the way it is initiated and where in memory it is loaded.

This project uses a TSR program. To accomplish the extraordinary task of TSRs and to adapt them for this project in particular it is imperative to use some of the "undocumented" DOS functions.

Undocumented DOS

Undocumented DOS according to Schulman et al. (1990) is:

> ...the body of functions and data structures that can reasonably be considered part of MS-DOS or PC-DOS but that are either not mentioned in the microsoft or IBM documentation or that are marked "Reserved" (p. xiii).

These functions according to Boling (1992) were in fact documented everywhere but in the MS-DOS technical reference. It was however, a pleasant sight and an indication of the stability of these functions in future versions of MS-DOS, when they were included in the Programmer's Reference in recent release of MS-DOS version 5.0. Michels (Waite Group, 1988), also see DOS as partly undocumented. He contended that there are still several MS-DOS functions that are either poorly documented or not documented at all. He also recommended the use of a "break-out switch" debugger (a resident debugger that can be activated with a special hardware switch) in order to stop the machine and examine the computer system state at any time. Knowledge of the system at that level will promote better programs that take advantage of all of MS-DOS internal functions. The art of system reliability is the recovery of the system to the point it was before an interrupt.

This experimental design capitalized on the systems development methods used by developers in the field of computer science and engineering.

The review shows that the area of computer failure, reliability and integrity, and data loss and recover are emerging topic for researchers. The continually changing technology in these areas and the proliferation of computers in all sectors of society propel individuals to investigate means to attain high reliability and integrity and data safety.

CHAPTER III. DESIGN METHODS AND PROCEDURES

Questions of the Study

This study focused on the following major research questions:

1. Can a software system be developed that will provide recovery from system failure due to
    - a) power disruption
    - b) accidental re-booting
    - c) software errors?

2. What problems exist in achieving a software-only recovery system on the IBM PC-XT compatible machine?
    - a) Are the majority of applications software compatible to a software recovery system?
    - b) What resources are required for acceptable performance?
    - c) What situations such as software errors in application programs, are not recoverable?

3. What is the degradation in application program performance when utilizing a software recovery system in comparison to performance when the recovery system is not used?

    a) What is the degradation for numerically intensive applications?

    b) What is the degradation for disk intensive application?

    c) What is the degradation for keyboard intensive applications?

Hypothesis of the Study

Research hypotheses to be tested are as follows:

1. A software system recovery program can be developed which provides recovery from power interruptions, system resets and software system crashes.

2. Standard application software such as disk utilities, word-processing programs, spreadsheet programs and statistical analysis programs will operate successfully with a software recovery system.

3. System degradation as measured by the difference in seconds for standard comparison programs operating with and without the software recovery system will not be greater than 0.1 second.

Methodology of the Study

To provide answers to the questions and hypothesis above, the following methods were employed:

Hypothesis 1:

A systems approach was employed to develop an experimental software recovery system referred to henceforth as CRAM. This method included:

a. Identifying the desired operational characteristics of the software, i.e. operational goals and specification of user interface.

b. Analyzing the system resources available within which the system must perform. Included are the CPU, DOS, BIOS & RAM .

c. Classifying the major system components required to perform identified system functions including:

    - direct disk I/O
    - scanning system for pertinent systems operational variables
    - manipulating the FAT
    - creating disk file storage areas
    - protecting disk storage area
    - loading RAM segments from disk

- direct video output
- process interruption
- determining segments changed by software activity
- saving CPU registers
- copying RAM segments to disk
- initiating a recovery process
- restoring CPU registers from disk
- passing control from the recovery system to the recovered software system

d. Assemblying components into executable files.

e. Developing and testing program functions.

f. Testing and revising the system.

g. Evaluating the system operational characteristics.

Hypothesis 2:

To test the validity of the experimental recovery system, sample application programs were executed with the CRAM recovery software installed. The power was interrupted and the system was reset from the keyboard using the <CTRL>-<ALT>-<DEL> keys combination while the application programs were running. Results were classified as conflict, partial recovery and full recovery.

A conflict is when the executing program cannot coexist with CRAM, producing unpredictable behavior or causing the system to lock-up.

A partial recovery is when the restoration seems to occur but the system lock-up immediately or when attempt is made to enter information at the keyboard.

A full recovery is when the system is in the state it was before the interruption and the behavior is in all respect what was expected. Nine well known commercial programs were used in this test and their recovery state categorized into one of the above classifications.

The application programs used for testing were

- Norton Utility version 4.50
- WordStar release 3.31p
- WordStar 2000 release 1.01
- Turbo C+ + version 1.00
- Sideways version 2.01
- Lotus 123 version 2.01
- WindowDOS Capture Utility
- DOS resident print for DOS version 3.3
- WordPerfect version 5.1

Hypothesis 3:

The system was tested using the following method:

1. Generating sample time duration for execution of 8 benchmark programs described as follows:

   a) Sieve of Erotosthenes: Generating 16380 prime numbers.

   b) Disk I/O write : Writing 512 characters to a file on the hard disk.

   c) Disk I/O read : Reading 512 characters from a file on the hard disk.

   d) Keyboard interrupt : Scanning of the keyboard interrupt.

   e) Video display : Writing 512 characters to the video screen.

   f) DOS idle interrupt : Scanning the DOS idle interrupt.

   g) Clock interrupt : Scanning the clock interrupt.

   h) Random number : Generating 2000 random numbers.

2. Aggregation of the 100 samples into 20 groups for purposes of data analysis and plotting.

3. Use of the Students t-test for differences between independent means of samples generated with the recovery system installed and without the system

installed. Significance was established by testing for rejection of the null hypothesis of, no difference between means at the 0.05 level.

CHAPTER IV. RESULTS OF ANALYSIS AND HYPOTHESIS TESTING

The results of this study answered three questions reflected in the following three hypothesis:

Hypothesis 1

A software system recovery program can be developed which provides recovery from power interruptions, system resets and software system crashes.

The hypothesis was confirmed, the results below demonstrate the feasibility of the system's development.

The software CRAM was developed to answer the above question. The development of CRAM followed the modular and systems approach to software development. Modular in that each section was developed and tested separately and systems in that the project was divided into various tasks and development followed a define path. The study was divided into two major areas: Installation and Operation. Installation was further divided into System check, Disk scanning and Disk preparation. Operation was further divided into System start-up, Memory save, Memory restoration, and Data maintenance. The diagram in Figure 1 shows how CRAM's development was subdivided into various tasks and the goals for each task.

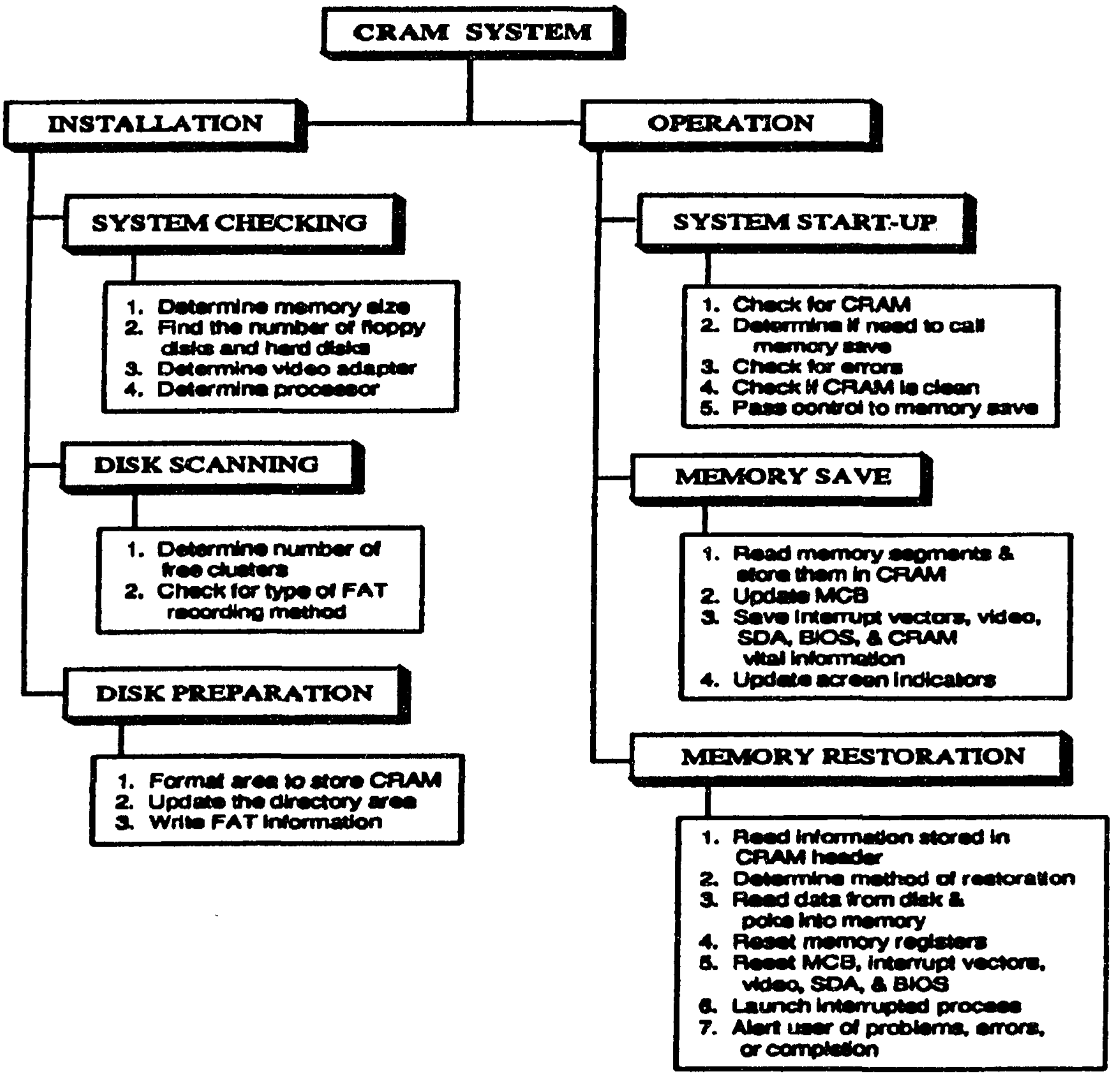

CRAM SYSTEM
INSTALLATION
SYSTEM CHECKING
1. Determine memory size
2. Find the number of floppy disks and hard disks
3. Determine video adapter
4. Determine processor
DISK SCANNING
1. Determine number of free clusters
2. Check for type of FAT recording method
DISK PREPARATION
1. Format area to store CRAM
2. Update the directory area
3. Write FAT information
OPERATION
SYSTEM START-UP
1. Check for CRAM
2. Determine if need to call memory save
3. Check for errors
4. Check if CRAM is clean
5. Pass control to memory save
MEMORY SAVE
1. Read memory segments & store them in CRAM
2. Update MCB
3. Save interrupt vectors, video, SDA, BIOS, & CRAM vital information
4. Update screen indicators
MEMORY RESTORATION
1. Read information stored in CRAM header
2. Determine method of restoration
3. Read data from disk & poke into memory
4. Reset memory registers
5. Reset MCB, interrupt vectors, video, SDA, & BIOS
6. Launch interrupted process
7. Alert user of problems, errors, or completion

Figure 1. CRAM's development showing the goals assigned to each task

An IBM-PC XT compatible computer from Cordata running at 10Mhz was used for both development and testing of the software. In order to gain low level interface to both the machine and the operating system most of the code was written in "C". Turbo C version 2.0 was chosen because it was available and provides a good integrated environment for software development. Some codes were done in assembly language to access low level interface to DOS that were either not available in "C" or provided better data access speed. Turbo Assembler version 2.0 was chosen because of availability.

CRAM is a unique software tool because it was designed to be a replacement for a standard UPS system. The power of this tool lies in the use of Undocumented DOS functions, memory management tool, disk management, context switching and the timely backup of the system to the hard disk.

Installation

The installation process involves checking the system and the preparation of the storage area on the disk. Files stored on the disk must be preserved, while a large enough space should be prepared for CRAM. CRAM must operate in the background and should minimize overhead on the system. System overhead, the time used by the executing program, will slow the system down. System degradation below an acceptable level is undesirable. There are two major methods of saving files to the disk:

Method 1. Using the DOS I/O functions.

Method 2. Using the system BIOS for direct I/O.

Method 1 would introduce an overhead on the system. It would also means that the file will be saved according to the format DOS uses. DOS save files in a best-fit-by-cluster way, which means sections of the file could be at different locations on the disk. This create what is commonly known as file fragmentation. An additional overhead would be added for the seek time for each section of the file.

Method 2 on the other hand would give CRAM complete control over where the file is located on the disk. This method allows CRAM to bypass the FAT in the event of a corrupted FAT of disk file table. The seek time overhead could be at a minimum if the file is stored in consecutive clusters, since only the address of the first cluster is needed.

The two methods are compared in terms of overhead below using the following general formula:

$$OH_{total} = \sum_{0}^{s} E_c t_t + t_f + K \tag{1}$$

E_c - seek time to a cluster

t_t - seek time to a track

t_f - time to execute DOS function call

K - a constant of seek time for disk that park their head after each I/O

s - the number of different areas where the file is stored.

Method 1:

$$OH_{total} = \sum_{0}^{s} E_c t_t + t_f + K \qquad (2)$$

If files are stored in consecutive clusters the formula for the total overhead would be:

$$OH_{total} = E_c t_t + t_f + K \qquad (3)$$

If t_d is small and negligible then both methods would yield the same value for total overhead.

Method 2:

$$OH_{total} = E_c t_t + K \qquad (4)$$

s = 1, since only one seek was needed.

K varies with the disk drives but will be the same for both methods.

Method 2 was chosen for this project since this method allows access to the disk even when the FAT is damaged, and will always yield lower or the same overhead. A special program was written to prepare the disk for CRAM and install CRAM on the hard disk. The program INSTALL.EXE was used to install CRAM on the hard disk. The diagram below in Figure 2 shows the layout of the various section of CRAM on the hard disk.

The program first checks each cluster on the disk for the CRAM identification marker and a special security code. The disk must be a non-removable hard disk. It can be either the bootable default disk or any other hard disk sub-system installed in the computer. The disk is next checked for a space large enough to hold the system's memory and all the software needed to run CRAM. If there is not enough space on the disk, CRAM will not be installed. The disk must have, in addition to the minimum space required for CRAM, two or more extra free clusters. One cluster in CRAM is used for storing vital information for the operation of CRAM including the CRAM identification marker and copyright notice.

Organizing the hard disk

In order to gain the fastest disk access and reliability of data the hard disk was first organized using a third party disk organizing software Norton Disk Doctor (1989). Any other disk organizing or disk optimization software could be used. The

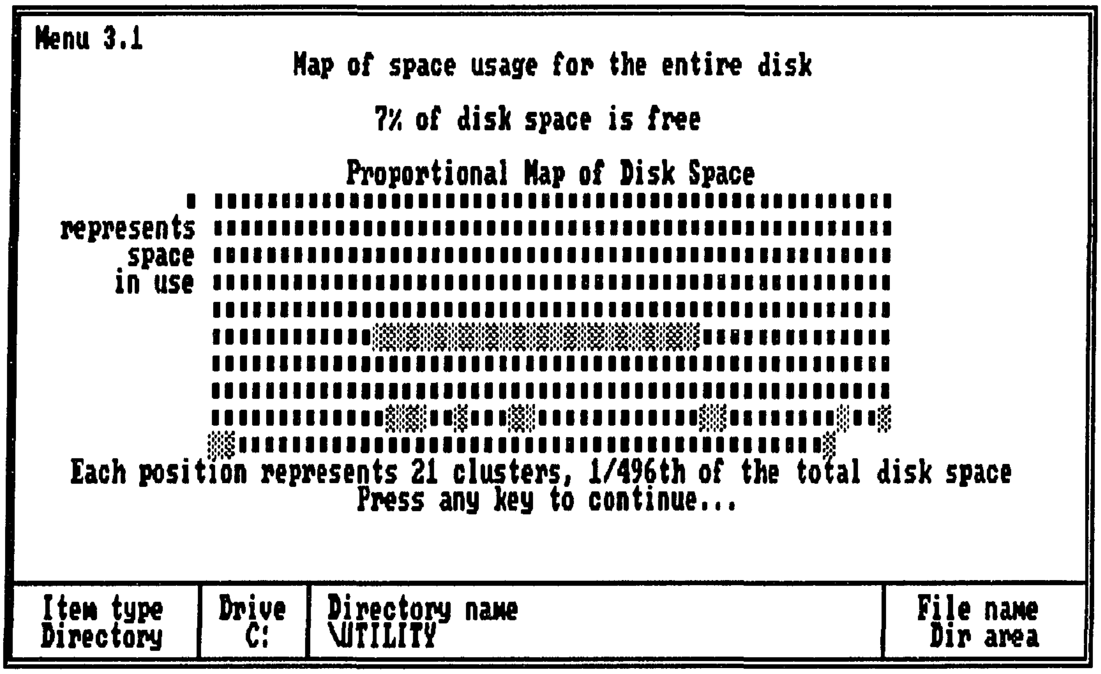
# of Sectors	
1	CRAM VITAL INFORMATION AREA (CVIA)
3	INTERRUPT VECTORS & DOS BIOS AREA
variable	MAIN MEMORY STORAGE AREA
32	VIDEO SCREEN MEMORY AREA
1	MEMORY CONTROL BLOCK (MCB)
4	SWAPPABLE DATA AREA (SDA)
varies	STACK MEMORY AREA

Figure 2. Layout of CRAM on the Hard Disk

disk will be analyzed by these software for fragmented space. Deleted files are only marked with a special code and the area is available for other files to use. However, the new file may need less space and thus leave some clusters unused. Norton Disk Doctor will arrange all the files on the disk to achieve the most efficient use of the disk. Figure 3 shows a fragmented disk and Figure 4 shows the same disk unfragmented by Norton Utility disk compression program.

Checking for free clusters

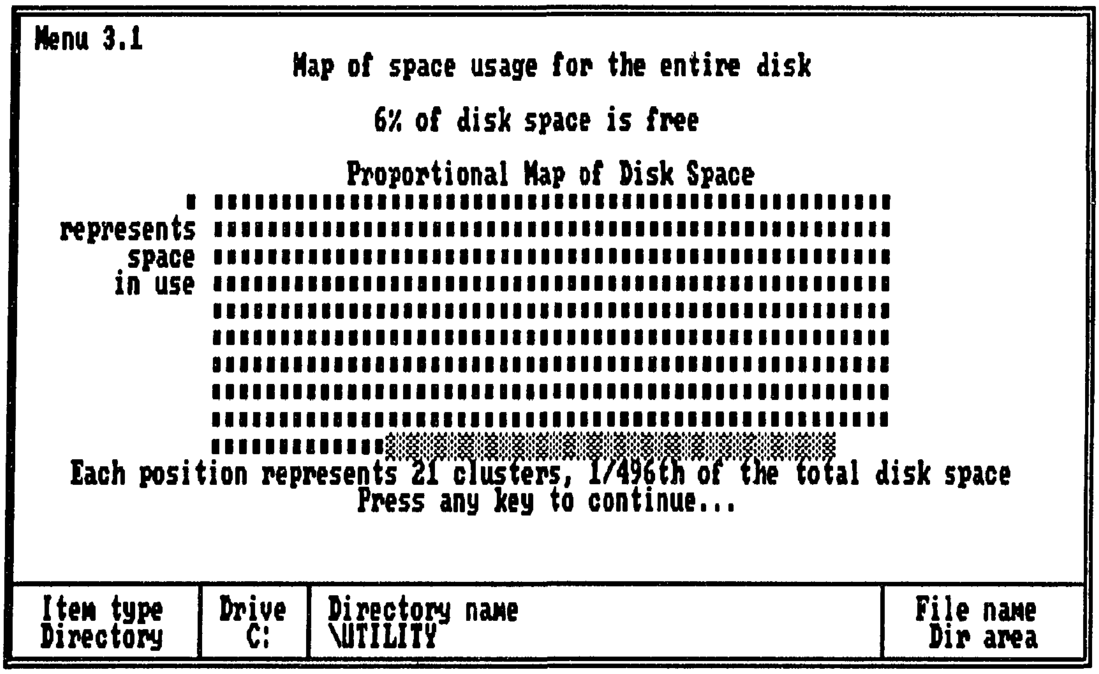
The entire disk was checked for the number of free consecutive clusters that could be used by CRAM. Clusters used by CRAM must be consecutive because no other checking will be done during memory save. The first cluster, or the first sector of the first cluster assigned to CRAM is the only information needed to instruct CRAM where to save memory on the disk. This method eliminates the time that would be used to check for correct sectors, and also, the seek time used during save was reduced to only one initial seek. Once enough space is found on the disk the system will be installed as illustrated by the flow diagram in Figure 5. Record of the clusters used by CRAM must be recorded in the File Allocation Table (FAT).



```
Menu 3.1
                    Map of space usage for the entire disk

                            7% of disk space is free

                          Proportional Map of Disk Space
     represents
        space
       in use

 Each position represents 21 clusters, 1/496th of the total disk space
                       Press any key to continue...
```

Item type	Drive	Directory name	File name
Directory	C:	\UTILITY	Dir area

Figure 3. Disk layout showing fragmented files



```
Menu 3.1
                    Map of space usage for the entire disk

                          6% of disk space is free

                        Proportional Map of Disk Space
         ▮
represents
     space
    in use
Each position represents 21 clusters, 1/496th of the total disk space
                        Press any key to continue...

Item type    Drive   Directory name                      File name
Directory     C:     \UTILITY                             Dir area
```

Figure 4. Disk layout after running Norton Utility disk organizing program (Speed Disk)

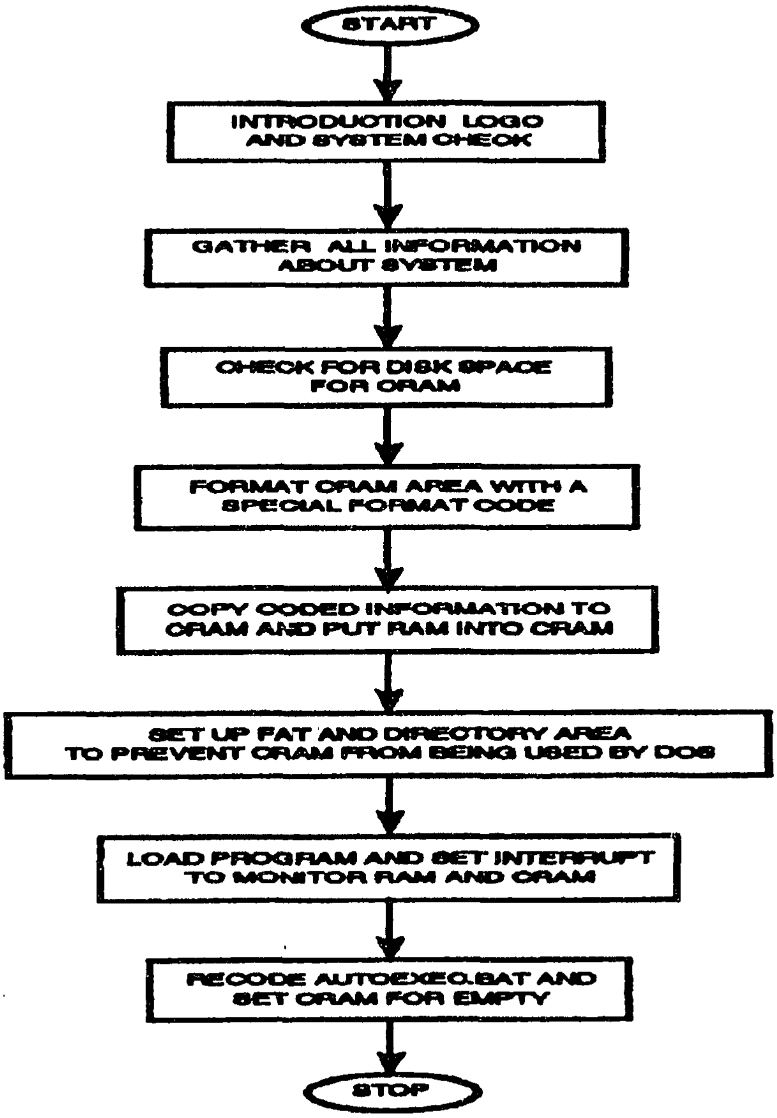

START
INTRODUCTION LOGO AND SYSTEM CHECK
GATHER ALL INFORMATION ABOUT SYSTEM
CHECK FOR DISK SPACE FOR CRAM
FORMAT CRAM AREA WITH A SPECIAL FORMAT CODE
COPY CODED INFORMATION TO CRAM AND PUT RAM INTO CRAM
SET UP FAT AND DIRECTORY AREA TO PREVENT CRAM FROM BEING USED BY DOS
LOAD PROGRAM AND SET INTERRUPT TO MONITOR RAM AND CRAM
RECODE AUTOEXEC.BAT AND SET CRAM FOR EMPTY
STOP

Figure 5. Installation Flow Chart

Writing the FAT

Writing information directly to the FAT is very important to CRAM but must be done carefully. Writing the incorrect information here can corrupt the disk file system and could crash the hard disk. The method used by CRAM was as follows:

1. Locate the first cluster to store CRAM.
2. Check for minimum sectors needed.
3. Calculate the number of sectors needed.
4. Calculate the sector number at the beginning cluster.
5. Set start sector to that sector number.
6. Mark FAT to include all clusters that make up CRAM area of the disk.
7. Put End of File Marker in the last sector of CRAM.

First the format of the FAT writing scheme was determined. There are two methods for writing the FAT: 12-bit and 16-bit. There may be other methods but these two were the only known methods at the time of development.

12-Bit FAT This format is the more common and more complicated of the two. Most floppy disks use this method for recording the FAT. The FAT is organized as a table of 4096 numbers that range from 0 to 4095 (0h through FFFh).

The number in each entry represents the status and use of the cluster that corresponds to that FAT. The number in each entry must not exceed three hexadecimal digit, which is a key element of how the 12-bit FAT entry is stored. The FAT entries are organized in pairs, where each pair occupies three bytes. To decode the information in the FAT use the following steps:

1. Multiply the FAT entry by 1½ bytes (multiply by 3, then divide by 2).
2. The result is the offset into the FAT, pointing to the entry that maps the cluster. That entry contains the next cluster occupied by the file.
3. The result has four hexadecimal digits but three digits are needed. Determine whether the FAT entry is odd or even.
4. If the entry is even, logical AND with 0FFFh. If it is odd, shift the result by 4.
5. If the resulting three digits falls between FF8h and FFFh this indicates the end of the file. Otherwise the digits represents the number of the next cluster occupied by the file.

16-Bit FAT This method of recording the FAT is associated with most hard disks, capable of storing more than 4086 clusters. This method works the same as the 12-bit one, but is much simpler. The entries are four bits larger than that of the 12-bit. The entries are therefore word values stored one after the other in the table. The values therefore range from 0000h to FFFFh (instead of 000h to FFFh), the only difference is the addition of the high-order hexadecimal F. To get the cluster values it is simply a matter of reading the word value of the entry that is being queried.

After the FAT recording method was determined, the next step was to check for consecutive free clusters. The minimum amount of clusters needed was determined using a formula based on the FAT recording method, the number of sectors per cluster and the number of sectors needed to store a DOS segment (HYDE, 1988) of memory. To determine the number of sectors needed per segment of memory each segment was considered to be 64K bytes wide and starting at absolute address zero (0000:0000). The formula used was:

$$Sectors per Segment = \frac{(Bytes per 64K Segment)}{(Bytes per sector)} + 1 \qquad (5)$$

In the case of CRAM

$$Sectors\ per\ Segment\ =\ (65535\ /\ 512)\ +\ 1\ =\ 128$$

After recording the clusters reserved for CRAM, the sectors were formatted to check for bad sectors and the integrity of data stored in those sectors.

Formatting CRAM

The disk area occupied by CRAM was first formatted to eliminate any bad sectors. If any bad sectors were found another area of the disk would be chosen. If no other area was available then the bad areas would be marked off to prevent CRAM from writing to them. This was important since direct disk I/O was used to save and retrieve data from CRAM. The following pseudo-code illustrates the format procedure:

```
   Search disk for minimum space
If space found > = minimum space
 {
   Check area for bad sectors
   if bad sectors found
     {
       check for another area on disk
       if no free chunk found
         {
           mark off bad sectors
           adjust size of space needed
         }
     }
 fill area of cram with a special character (ϵ)
   this is ascii character hex code EEh
 Enter the file in the Directory area
 Update the file size
 Update the File Allocation Table (FAT)
 Return the message that CRAM-DISK is ready

 }
```

The method used by CRAM to access data stored in these sectors make it imperative that CRAM's sectors be protected from DOS manipulation.

Protecting CRAM from DOS

CRAM needs to be protected from DOS operations such as DELETE, RENAME, COPY and ERASE. CRAM should also be protected from other commercial disk utilities programs. Unfortunately, there are some limitations to that. The following steps were taken to protect CRAM.

1. The filename that was used as place holder in the directory area of the disk was done in lower case letters. DOS does not recognize a lower case filename as a valid directory entry.
2. The file name has imbedded "null" characters that cannot normally be entered at the keyboard.
3. The file holder for CRAM was given the SYSTEM and READ ONLY file attribute. This is useful to prevent software like Norton Utilities or other Disk managers from moving the file clusters allotted to CRAM. This is extremely important since CRAM uses only a block of consecutive clusters.

The steps above was accomplished by inserting a special entry in the directory area of the hard disk containing CRAM reserved sectors.

Writing the directory area

The entry into the directory area is the key to keeping CRAM away from DOS. It can be seen that the file name "cram_xxx.xmd" was recorded in lower case, which DOS does not recognize as a legal file name. This entry is checked when CRAM is invoked for the first time to find the location of the first cluster reserved for CRAM. If this file exist and all necessary conditions are met for CRAM's operation the file entry is updated.

CRAM'S Operation

The main task of CRAM is to operate in the background of the computer performing timely backup of the computer system's memory to disk, with limited interruption of the foreground process. The system boot-up process and the subsequent loading of CRAM is demonstrated by the flow diagram in Figure 6. The operation of CRAM was visible by symbolic display on the screen. Figure 7 shows the symbols display by CRAM and the meaning of each. DOS has a built-in mechanism whereby developers can extend the operating system using interrupts. DOS stores a group of addresses between location 0000:0000h and 0000:0400h. Each address points to a routine in memory that performs a specific task. By replacing these routines with other codes the operation of that interrupt can be controlled.

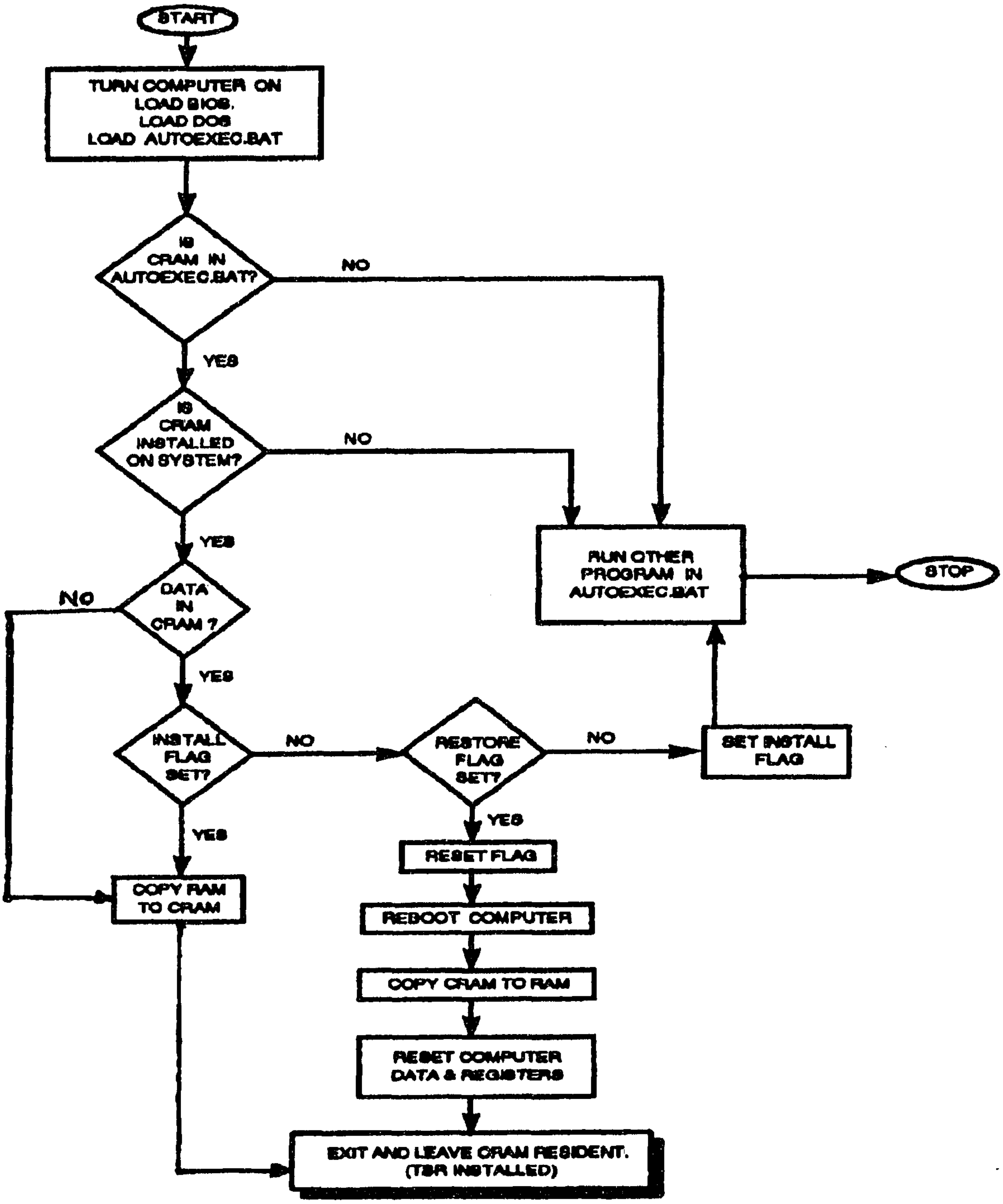


Figure 6. Flow diagram showing the start-up process for a DOS machine and the determination of CRAM's status

Process	Symbol	Error
Saving Registers to CRAM	Φ	
Getting Registers and vital CRAM statistics	∩	
Saving video memory to CRAM	none	
Resetting video memory	none	
Saving interrupt vector table to CRAM	φ	
Resetting interrupt vector table	Ω	
Normal CRAM operation in progress	ε	
Putting CRAM vital information in CRAM	∞	
Getting CRAM vital information from CRAM	≡	
Resetting memory	δ	
Loading CRAM	ç	
Saving DSA	Σ	
getting DSA from CRAM	Γ	
Restoring DSA	α	
Completion of DSA restoration	ß	

Figure 7. CRAM's process screen indicators

Some interrupts are activated by events in the system. Thus there are two types of interrupts hardware interrupts and software interrupts. This task was accomplished by utilizing DOS hardware and software interrupts and replacing some of these routines with new codes that do some operations that are specific to CRAM and also allow other programs to have access to the original routine. The major interrupts used by CRAM are the keyboard interrupt (9h), the clock interrupt (1Ch) and the DOS idle interrupt (28h). The flow Diagram in Figure 8 shows CRAM's operation and how the various interrupts are used.

Using hardware interrupt Hardware interrupt occurs when any peripheral device attached to the system requests the use of the CPU. This, in turn, will bar all other attachments from accessing the CPU until that device completes its use of the CPU. The hardware interrupt used by CRAM is interrupt number 09h, the keyboard service routine. The number of keystrokes are monitored to determined if the system needs to be transferred to CRAM. This interrupt was needed in cases where the program running in memory disable the other interrupts. The keyboard interrupt is always available but some programs that redirect inputs to receive from remote location, like a communication program using TTY (teletype protocol), thus under the control of another computer, may disable this interrupt.

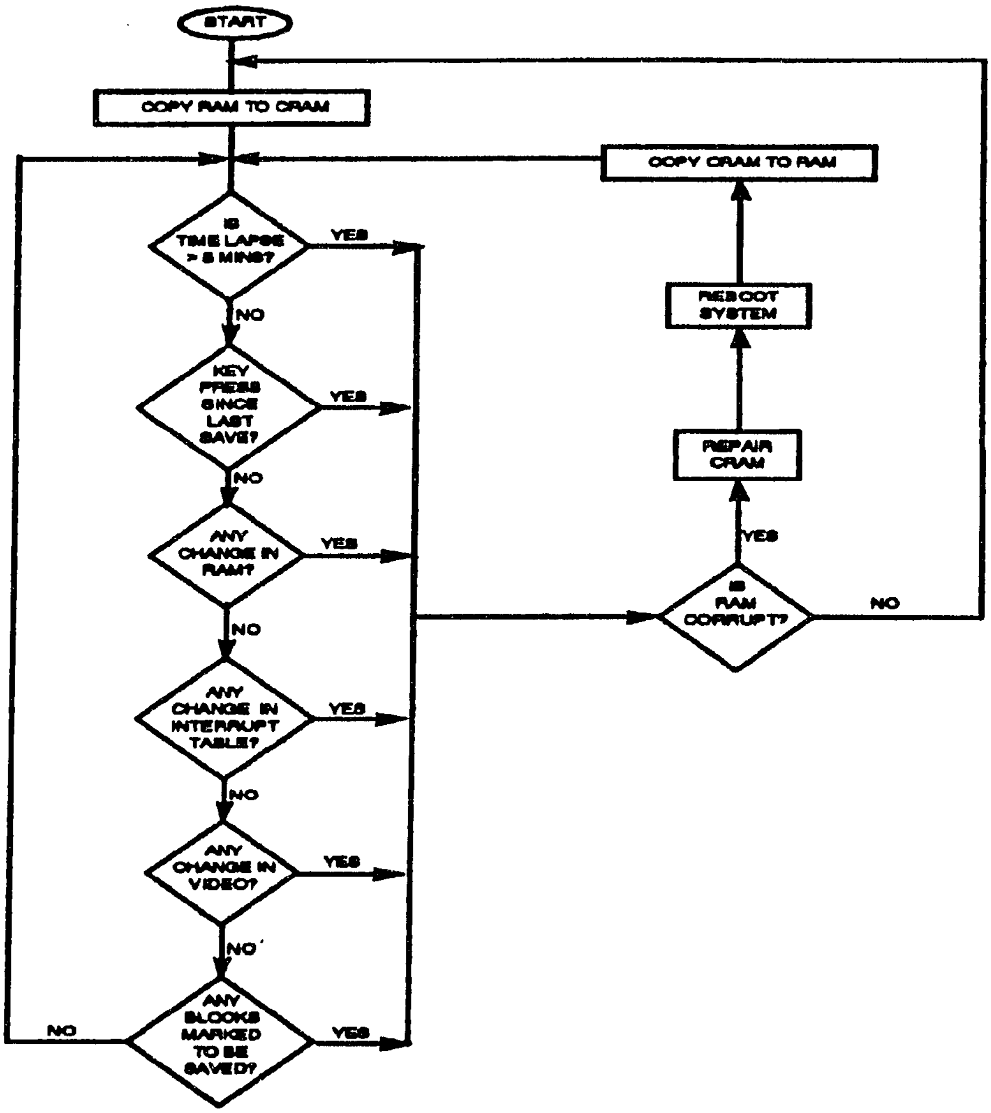


Figure 8. Flow Diagram showing CRAM's operations

Using software interrupt Software interrupts are routines provided by the system Basic Input-Output System (BIOS) that are activated by a predetermined event. The software interrupts used by CRAM are interrupts 1Ch, the timer interrupt service routine and interrupt 28h, the DOS idle interrupt routine. The timer routine is activated 18.2 times per second by the system clock while the DOS idle interrupt routine is activated whenever the system is waiting for input from the console (keyboard). These two interrupts are continuously monitor the system by providing the timing for critical events and scanning the keyboard. Some programs disable the DOS idle interrupt, thus the clock interrupt is left to monitor the system. The clock interrupt was observed to conflict with some programs when used by CRAM. All three interrupts were then included in this project to enable CRAM to operate under most conditions.

Using the keyboard interrupt This interrupt was used to record the number of keys entered at the console. Knowledge of the number of keys enabled CRAM to determine if memory needed to be saved. This interrupt was the only method of interface to the console when some program running in the foreground does not use the other interrupts. Many graphics programs and word processors like WordPerfect[1] do not use the DOS idle interrupt and therefore, would make CRAM become dormant for the duration of WordPerfect's existence in the system.

[1] WordPerfect is a trademark of WordPerfect Corporation

Using the clock interrupt The clock interrupt was used to monitor the systems RAM. One segment is checked in a round robin method every 18.2 seconds. At the completion of a cycle a flag is set for the idle interrupt or the keyboard to transfer RAM to CRAM.

Using the DOS idle interrupt This interrupt, when used by other programs running in the system or not hooked by any software, monitors the systems clock and various flags set by the clock or the keyboard interrupt. This interrupt routine will then transfer RAM to CRAM if the flags are set or the time since the last save exceeds the limit set by CRAM. CRAM must interrupt the system when any of the above interrupt is initiated. This interruption should only be done when the state of the system is in a safe mode and an interruption will not cause the system to "crash". The system must be constantly interrogated to know its state. The regular DOS functions does not provide a means to interrogate the system; a check of the undocumented DOS functions reveal some function that can provide this information.

Use of Undocumented DOS

The operation of CRAM requires access to all vital information about every programs running in the system and about CRAM itself. This information include knowing:

1. What state DOS is currently in
2. How to get the current DTA
3. How to set the DTA to a new location
4. The extended error information for the current process
5. How to get the address of the DOS information area

The above information is not fully documented but belong to a pool of information called Undocumented DOS.

This project would not be possible without the use of some undocumented DOS functions. These functions use the interrupt 21h service routine with the listed number as subroutines of this interrupt. Undocumented DOS functions used in CRAM were:

1.	34h	GET INDOS Flag Address
2.	50h	Set Active Process Data Block
3.	51h	Get Active Process Data Block
4.	5D06h	Get DOS Data Area Address
5.	5D0Ah	Set DOS Extended Error Information
6.	5D0B	Get DOS Data Areas

A very important area that worth further discussion is the DOS information area known as the DOS Swappable Data Area.

Using the DOS Swappable Data Area (SDA)

The SDA is accessed using the undocumented INT 21h function 5D06h for DOS 3.1 through 3.3, and function 5D0Bh for DOS 4.x. This is a block of data, typically about 73Ch bytes in size, that contains the current context of MS-DOS. The context of DOS includes the current PSP, and the three MS-DOS stacks. CRAM therefore, saves the SDA and later restores it after completing a task. This allows CRAM to pop up at any time without the danger of violating the non-reentrancy associated with DOS. This method does not require CRAM to wait until the DOS flags indicate it is safe. CRAM transfers this area of memory to disk each time a memory block is transferred to disk, which plays a significant role in restoring the system after an interruption. The number of times the memory is transferred to the disk depends on the refresh period.

Determination of CRAM refresh period

Updating information in CRAM was approached from a conservative assumption that power interruption is eminent at any time. Therefore, memory was checked for changes 18.2 times per second. This occurred when the system clock pulse is recorded by the hardware interrupt vectors 8(08h) and 28(1Ch). Each segment was checked using the checksum method. Checksum is the summing of the memory bytes. This number can be very large, therefore, since only the upper 2 bytes are significant, the result was shifted right by 8 to conserve memory and storage. The following formula was used:

$$\mathit{Checksum} = (\mathit{Totalmemorysegments}) << 8 \tag{6}$$

A flag is set for each segment to indicate areas of memory that has been changed. After all segments are checked the disk transfer will take place at a non-critical time. CRAM is also refreshed based on the number of keys pressed since the last memory save. The default setting is 50 keystrokes but can be changed at anytime. CRAM intercepts interrupt 09h, which is used to monitor the keyboard activities.

During each saving of memory to CRAM all the registers, the interrupt vectors, the SDA, and the video screen are also transferred to disk.

Saving Memory

Memory is saved periodically after the flag indicating that all segments have been checked for changes. Segments that were changed are the only ones saved. This process involves the storing of each segment of memory in the number of sectors calculated from equation 1. After CRAM is loaded the save routine is vectored to interrupt 28h. CRAM could also be forced to save by any other program running in the system that activates this interrupt, or by pressing the following key sequence

"<CTRL>-<ALT>-<5 on the keypad>". After checking the flags for saving the segments, the process of saving memory is done. The following steps shows how memory is saved:

1. Save the DSA.
2. Set the PSP to that of CRAM.
3. Set the DTA to that of CRAM.
4. Check the MCB and repair if needed.
5. Save the MCB.
6. Save interrupt vector table.
7. Save all flagged memory segments.
8. Save the video screen.
9. Save the stack.
10. Save all registers.
11. Update CRAM header information.

When power is interrupted using the reset button or unplugging the computer the system must be restored.

Memory Restoration

The memory restoration process involves a very rigid sequence of events. There can be no delays between events if interrupts are not disabled in all intervals between events. CRAM will re-boot the system when invoked for the first time. This will ensure the resetting of the original ROM BIOS and machine codes. After CRAM is loaded the restoration routine is vectored to interrupt FCh. CRAM could also be forced to restore by any other program running in the system that activates this interrupt, or by pressing the following key sequence "<CTRL>-<ALT>-<LEFT SHIFT>-<5 on the keypad>". Unlike the force-save key combination the force-restore key combination cannot be changed by the user.

Memory is restored in the following order:

1. Restore all segments of memory.
2. Reset the video memory area.
3. Reset all registers.
4. Set the Data Transfer Area to that of the interrupted program.
5. Set the Program's Segment Prefix to that of the interrupted program.
6. Reset the interrupt vector table.
7. Reset the DOS Swappable Data Area.
8. Reset or Rebuild the Memory Control Block.

9. Reset the TSR's stack.
10. Jump to the location where the TSR was prior to the systems interruption.

The sequence of events above is one method of context switching.

Context Switching

The hardest and most interesting part of the project was context switching during a restoration of the system's memory. Context switching during the saving of memory to CRAM was trivial compared with the restoration process. Context switching is a method of manipulating the computer registers and stacks to simulate a multi-user system and give different programs control of the system. This was accomplished by the use of the information stored in each application's Program Segment Prefix (PSP) and data in the DOS Swappable Data Area (SDA). Figure 9 shows the content of the PSP and Figure 10 shows the content of DOS SDA.

Another part of the restoration process that is very important and must be discussed when using context switching is the Memory Control Blocks (MCB).

Offset	
0000h	Int 20h
0002h	Segment, end of allocation block
0004h	Reserved
0005h	Long call to MS-DOS function dispatcher
000Ah	Previous contents of termination handler interrupt vector (int 22h)
000Eh	Previous contents of Ctrl-C interrupt vector (Int 23h)
0012h	Previous contents of critical-error handler interrupt vector (Int 24h)
0016h	Reserved
002Ch	Segment address of environment block
002Eh	Reserved
005Ch	Default file control block #1
006Ch	Default file control block #2
0080h	
00FFh	Command tail and default disk transfer area (buffer)

Figure 9. Layout of the Program Segment Prefix (PSP)

Offset	
00h	Critical error flag
01h	InDOS flag
02h	Drive on which critical error occurred or FFh
03h	Locus of last error
04h	Extended error code of last error
06h	Suggested action for last error
07h	Class of last error
08h	ES:DI pointer for last error
0Ch	Current DTA
10h	Current PSP
12h	Stores SP across an INT 23
14h	Return code from last process termination
16h	Current drive
17h	Extended break flag (1 BYTE)
1Eh	Available memory block (3 WORD size)
24h	Total memory installed (WORD)
26h	
to	DOS Stacks and other important data
73Ch	Some values are unknown to the public

Figure 10. Layout of DOS Swappable Data Area (SDA)

Working With Memory Control Blocks

After restoring memory, and sometimes during DOS operation, the memory control blocks can be damaged and thus need repair. The MCBs are like the FAT if corrupted the entire system will suffer "diarrhea" and crash. The disk system will not be affected but data in memory will be lost. A routine is called by CRAM to repair the MCB, if during its monitoring of the system, the MCB was found to be corrupted or unlinked.

The software was developed and tested in individual modules to facilitate easy debugging. The first objective was to device a method to safely prepare the hard disk to make the most efficient use of the space reserved for CRAM. The number of free clusters on the disk was determined using the code shown in Figure 11. Note the need to check for the method used to encode the File Allocation Table (FAT).

The entry into the directory area was done using the code found in Figure 12. Lower case characters were used for the file name. This entry will not be seen when the DIR command is entered at the DOS prompt. The FAT must be updated to mark off the clusters used by CRAM. The code segment in Figure 13 was used to update the FAT. After preparing the disk, routines were written to save the system's memory to the disk.

```
int get_fatfree(struct free_fat *freefat, WORD clust_req)
{
   long i, j, m, f_pos;
   DWORD f_begin, f_end, f_size=0, f_mark=0;
   WORD f_entry; WORD fatsize;  union FATS *f_ptr;

   DOSsec= 1; fatsize = bpb.nspf; nsects = 1; j = i = 0L;
   for (m=0; m<fatsize; m++)
     {
       if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
         return(-result);
       f_pos = i = 0L;
       while ((f_pos + sizeof(f_ptr) +1) <= (nsects * bpb.bps))
         {
          switch(dpt.sysid)
            {
              case D_FAT16:
                {      f_pos = ((i * 2) % bpb.bps);
                       f_ptr = (union FATS *) (&buffer[f_pos]);
                       f_entry = f_ptr->fat_16.fat16;
                       i++;j++;
                       break;
                }
              case D_FAT12:
              case     0:
                {
                       f_pos = (((i * 3) / 2) % bpb.bps);
                       f_ptr = (union FATS *) (&buffer[f_pos]);
                       if(i & 1) f_entry = f_ptr->fat_12_hi.fat12;
                       else    f_entry = f_ptr->fat_12_lo.fat12;
                       i++;j++;
                       break;
                }
       ... See Appendix C.
}
```

Figure 11. Code to find free clusters on the disk

```
int  put_f_name(struct free_fat f_free, DWORD data_sec)
 {
   BYTE update=0; WORD f_entry; WORD fat_entry[2]; struct dir_entry d_entry;
   struct dir_entry tmp_d_entry=
    {
     "cram_mem","xmd",0xC1," CRAM V1.1", 0x00, 0x00, 0x00, 0x00
    };
   long i, j, k, logical_sector; WORD start_cluster = 0;
   struct date ddate; struct time dtime;
   start_cluster = f_free.fbegin;
   getdate(&ddate); gettime(&dtime);
   tmp_d_entry.f_attrib=(ARCHIVE|R_O|SYSTEM|HIDDEN);
   tmp_d_entry.f_name[4] = 0xFF;
   tmp_d_entry.f_reserved[0] = 0xFF;
   tmp_d_entry.f_start_cluster = f_free.fbegin;
   tmp_d_entry.f_size =(long)((f_free.fsize+1) * bpb.bps * bpb.spc);
   DOSsec=(bpb.nspf * bpb.nfats) + 1;
   nsects = 1; i = 0L; j  = (long)((bpb.nroot_dir * 32) / bpb.bps);
   for (k=0; k<j; k+ +)
     {
       i = 0L;
       if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer )) != 0)
           return(-result);
       while( i < (bpb.bps) )
         {
           memcpy(&d_entry, &buffer[i], 32);
           if (memcmp(tmp_d_entry.f_name, d_entry.f_name, 8) = = 0x00) update = 80;

   ... See Appendix C.
   return(0);
}
```

Figure 12. Code to make a directory entry that protect CRAM from DOS

```
int put_fat(struct free_fat freefat, DWORD data_sec)
{
   DWORD i, j, k, m, f_pos; DWORD f_begin, f_end, f_size=0, f_mark=0;
   WORD f_entry; WORD fatsize, cps; union FATS *f_ptr;
   cps = dpt.sysid == D_FAT12 ? (bpb.bps * 2 / 3)+1 : (bpb.bps / 2 );
   for (k=0; k<bpb.nfats; k++)
   {
   DOSsec= ((k*bpb.nspf)+1+(f_free.fbegin / cps));
   fatsize = (f_free.fsize / cps) + 1; nsects = 1; j = ((f_free.fbegin / cps) * cps); i = 0L;
    for (m=0; m<fatsize+1; m++)
     {
       if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
          return(result);
       f_pos = i = 0L;
       while ( (f_pos + sizeof(f_ptr) +1) <= (nsects * bpb.bps))
         {
          switch(dpt.sysid)
            {
              case D_FAT16:
               {     f_pos = ((i * 2) % bpb.bps);
                     f_ptr = (union FATS *) (&buffer[f_pos]);
                     f_entry = f_ptr->fat_16.fat16;
                     if (f_entry == 0 && j >= f_free.fbegin && j <= f_free.fend)
                       {
                         if(j == f_free.fend)
                           {
                            f_entry = 0xFFFF;
                           }
                         else f_entry = j +1;
                         f_ptr->fat_16.fat16 = f_entry;
                       }
              }
   ... See Appendix C.


  return(0);
}
```

Figure 13. Code to write to the FAT

Saving CRAM Data

The function *put_memory* shown in Figure 14 was used to save the system's memory to the hard disk. The Norton Utility package was used to view the disk to see if the data on disk is the same as was in the corresponding location in memory. The success of this stage led to the development of routines to restore the memory from data stored on disk.

Restoring Memory

This stage requires routines to get the data from the disk and routines to "poke" the data into memory. The code segment in Figure 15 was used to restore the memory. During the development of this area some problems developed which is described below. The next stage is the development of a method to scan memory and to determine, if any, segments changed since an earlier check of memory. The segments that were changed is transferred to the disk on the next clock cycle. The routine in Figure 16 was used to scan memory segments. This function is initiated by either the DOS idle interrupt, the keyboard interrupt, or the clock interrupt. The code in Figure 17 shows how to calculate the checksum for each segment of memory, tagging any segment that have been changed since the last checksum calculation on that memory segment.

```
  int put_memory(void)
  {
  ...
   for (seg=START_SEG; seg < maxmem; seg+=4096)
     {
      i = 0L;
      scr[0] [0].s_char = 0x00D8;
      scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4)
          + (scr[0] [0].s_attr << 4)) & 0x77;
      if (start_flag == 0)
        {
         offs = START_OFF;
         DOSsec = (c_header.mem_sec + ((WORD)((seg - sav_start) >> 12)
             * (0x10000/0x200)));
        }
      else
        {
        offs = 0;
        DOSsec = (long)(c_header.mem_sec + (((seg - sav_start) >> 12)
            * (0x10000/0x200)) - ((0x10000/0x200) - sec_seg1_end));
        }
      /* calculate the number of sectors per 64k segment */
      SEC_SEG  = (WORD)((0x10000  - offs) / (bpb.bps));
      if (DOSsec < reserved_sec) return(-1);
      for (j=0; j<SEC_SEG; j++)
        {
         movedata(seg, (offs+i), FP_SEG(buffer), FP_OFF(buffer), (bpb.bps *nsects));
         stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
         if ((result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
hst.SECTOR, nsects, buffer)) != 0)
                return(result);
         DOSsec ++;
         i += bpb.bps;
  ...
}
```

Figure 14. Code showing the main memory saving routine

```
   int reset_memory(void)
   { ...
    for (j=0; j<SEC_SEG; j++)
     {
      scr[0] [0].s_char = 0x0087;
      scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
      conv_mem_ptr = MK_FP(seg, offs+i);
      stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
      if ((result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
           hst.SECTOR, nsects, buffer)) != 0) return(result);
      if(filler == 0)
        {
          movedata(FP_SEG(&buffer[c_header.offs_filler]),
           FP_OFF(&buffer[c_header.offs_filler]),seg, (offs+i+c_header.offs_filler),
          ((bpb.bps * nsects)-c_header.offs_filler));
            conv_mem_ptr = MK_FP(seg, offs+i+c_header.offs_filler);
          j=0L; filler = 1;
        } else
        {
          if (seg == 0x2000 && DEBUG && ((offs+i) > 0x7E00));
          else
           movedata(FP_SEG(&buffer[0]), FP_OFF(&buffer[0]),seg, (offs+i),
              (bpb.bps * nsects));
        }
            DOSsec++; i += bpb.bps;
     }
/*\\\\\\\\\\\\\\\\\\\\\\\\\\ RESET REGISTERS AND DSA///////////////*/
    reset_video();  DOSsec += bpb.nspt;
    while(*diskette != 0x00);  /* wait for disk drive to stop spinning */
    disable();
    _AX = c_header.AX; _BX = c_header.BX; _CX = c_header.CX; _DX = c_header.DX;
    _ES = c_header.ES;  _DS = c_header.DS;  _CS = c_header.CS;  _SS = c_header.SS;
    _SP = c_header.SP;  _DI = c_header.DI;  _SI = c_header.SI;  _BP = c_header.BP;
    _FLAGS = c_header.FLAGS;
    enable();
   ... See Appendix C.
  }
```

Figure 15. Code showing the method used to restore memory

```
void mem_save(void)
 {...
  if (savetime >= timelag && !reset_mem)
   {
    save_DSA(); SaveDosSwap(); SetPSP(c_header.cram_psp);
    regs.h.ah = SET_DTA; sregs.es=c_header.cram_dta_seg;
    regs.x.dx = c_header.cram_dta_off;
    intdosx(&regs, &regs, &sregs); GetExtErr(&c_header.ErrInfo);
    if( ! mcb_chk(get_mcb()))
     {
      putstr(" <<-oo->>===== ERROR in MCB Chain ===<<-oo->>");
       result = set_MCB(0); scr[0] [79].s_char = result + 0x30;
     }
    else
     {
       result = save_MCB(); scr[0] [79].s_char = result + 0x30;
     }
    get_time_date();
    DOSsec = c_header.curr_mem_sec;
    if ( (START_SEG >= sav_end) || (maxmem > sav_end))
      {
        START_SEG = sav_start;
        maxmem    = sav_start + 0x1000;
        DOSsec    = c_header.mem_sec;
        c_header.curr_mem_sec = DOSsec;
        start_flag  = 0;
        sec_seg1_end = 0;
      }
    capture_mem();
    put_header();
    save_interrupt();
 ... See Appendix C.
```

Figure 16. Code for the main memory loop

```
  long mem_checksum(WORD seg_start, WORD offs_start, DWORD mem_size)
  {
    long sum=0x0000L, cnt=0, k, l, m, n;
    long i, seg, offs;
    WORD far *s_mem;
    ldiv_t   cal;
    cal = ldiv(mem_size, 0x10000);
    k = cal.quot;
    l = cal.rem;
    for (seg=seg_start; seg< =(k*0x1000); seg+ =0x1000)
     {
       cnt =0;
       if (k != 0)
         {
           n = 0x10000;
           offs = 0;
         }
       else
         {
           n = mem_size;
           offs = offs_start;
         }
       s_mem = MK_FP(seg, offs);
       while(cnt < n)
        {
           sum + = *(s_mem+ +);
           cnt+ =sizeof(s_mem);
        }
     }
     cnt = 0; offs = 0;
     if (k != 0)
           sum + = *(s_mem+ +);
           put_hex(*(s_mem));
           cnt+ =sizeof(s_mem);
        }
   sum = (sum > > 2);
   return((long)sum);
}
```

Figure 17. Code to show the calculation of memory checksum

This module was tested by observation of the codes displayed on the screen. This type of software, because of its unique operation, must be tested using observational evaluation.

The software was developed and installed on an IBM-PC XT compatible. One major requirement of CRAM is that it must be the first program to be in memory after the command interpreter, DOS COMMAND.COM. The code in Figure 18 was used to check for the presence of CRAM in memory each time the program begins execution. If other programs were installed then CRAM will abort the process and issue an error message. Figure 19 and 20 show the layout of memory before CRAM was installed and after installation respectively. It should be noted from Figure 20 that CRAM TSR is located between the resident portion of the command shell and the Transient Program Area (TPA), the area where all other users programs are executed.

The program developed above showed that a system can be developed that can provide recovery after a failure. This was demonstrated by unplugging the computer, re-booting the system, or using the reset button to reset the system. The test of the software answered the next hypothesis.

```
MCB far *IS_CRAM(MCB far *mcb)
{
...
    buf[0] = '\0';
    mcb = get_cmd_mcb(mcb);
    tmpmcb = mcb;
    tmpowner = mcb->owner;
    for (;;)
        switch (mcb->type)
        {
             case 'M' : /* Mark : belongs to MCB chain */
               {
                  mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
                  if(tmpowner == mcb->owner)
                  tmpmcb = mcb; break;
               }
             case 'Z' : /* Zbikowski : end of MCB chain */
                  {
                    s = progname_fm_psp(FP_SEG(tmpmcb) + 1);
                    while((s) && (i <= 128))
                       buf[i++] = *s++;
                    if ( strstr(buf,"CRAM") != NULL)
                     {
                      return(tmpmcb);
                     }
                    else
                     {
                      printf("\b\b");
                      put_str("\n\r CRAM cannot continue :: need to be the first");
                      put_str("\n\r Program to load in your Autoexec.bat file");
                      put_str("\n\r the file {");
                      put_str(buf);put_str("} was found instead\n\r");
                      return((MCB far *)NULL);
...
```

Figure 18. Code to check for CRAM in memory to make sure it is the first program in memory after the DOS shell (COMMAND.COM)

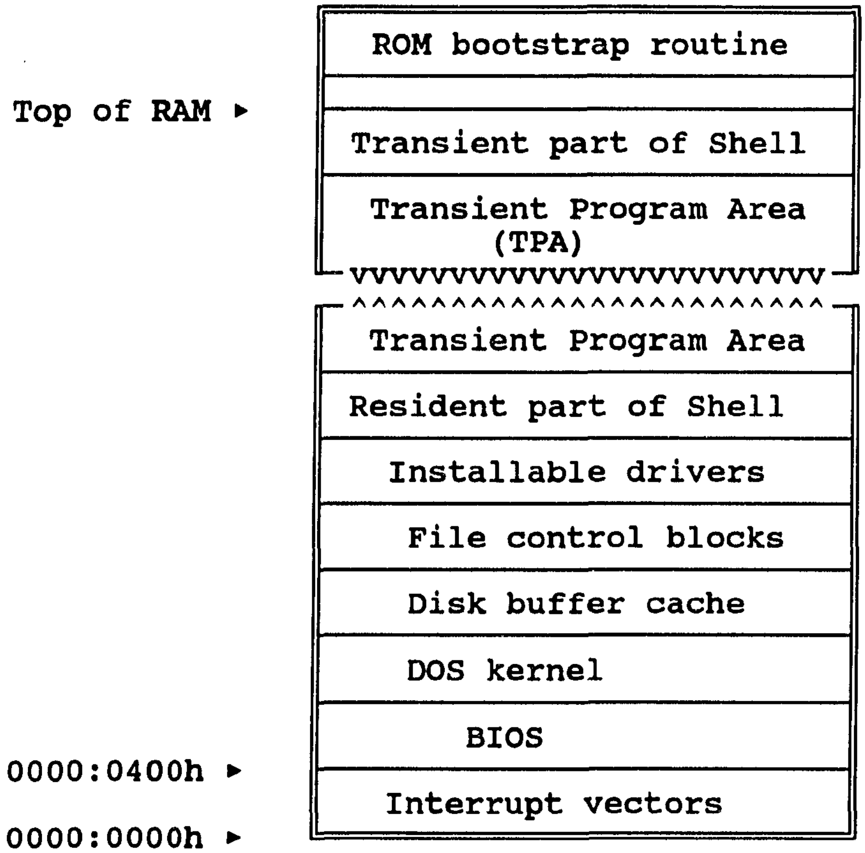
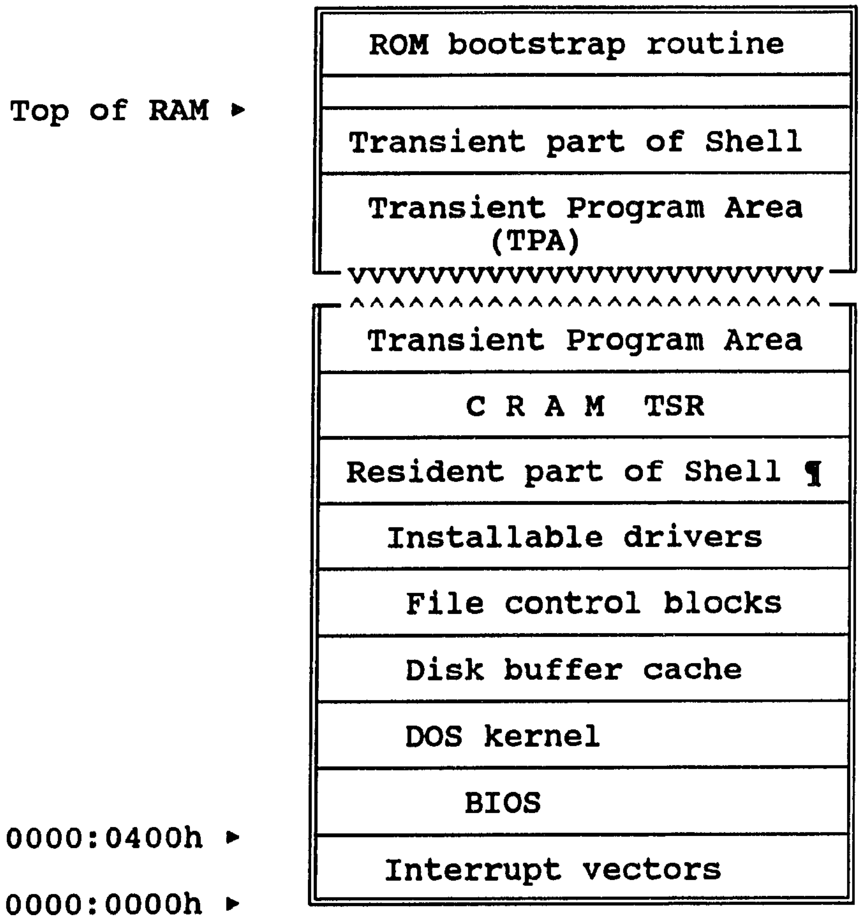


Figure 19. MS-DOS environment after the start-up process



¶ The default is COMMAND.COM

Figure 20. Layout of memory after CRAM was loaded

Hypothesis 2

Standard application software such as disk utilities, word-processing programs, spreadsheet programs and statistical analysis programs will operate successfully with a software recovery system.

This hypothesis was partially confirmed as the results demonstrate.

Results of software testing

Nine commercial software packages, shown in Figure 21, were used to analyze CRAM to determine if the system could be restored. Each software was tested separately and the system was either partially restored, fully restored to its original state before the interruption or there was a conflict. A partial restoration is defined as when the system appear to restore but not fully operational or error symbols indicate unpredictable events. A conflict is defined as when the presence of CRAM caused the executing program to lockup the system or gives unpredictable result.

Recovering from a power failure: Software conflicts

The system was tested for performance after a power failure. This area was the major premise on which the project was developed. The power switch and the

Commercial	Number of Conflicts	Number of Restorations after 5 executions with 5 sec. lag time	
Software	with CRAM	Partial	Full
Norton Utilities	0	1	4
WordStar	2	1	2
WordStar 2000	0	4	1
Turbo C	0	5	0
Sideways	0	1	4
Lotus 123	5	0	0
Capture (TSR)	3	2	0
DOS Print (TSR)	0	1	4
WordPerfect	1	4	0

Figure 21. Software conflicts and the number of partial and full restoration of the system by CRAM.

reset button on the computer were used to simulate power failure. Each program was executed five times with CRAM resident in memory and five times with CRAM not in memory. Figure 21 shows the list of programs that were tested and the associated conflicts. It was noted that about 75% of the time the system was restored. However, only 33% were full restoration. This could be due to the way in which both WordStar 2000, WordPerfect and Turbo C used the timer interrupt. It was observed that the display error symbol indicated, in most cases of partial restoration, that the timer interrupt was active.

Further analysis was done based on recovering from power failure: software conflicts and the time delay of other programs brought on by CRAM's presence in the system. problems.

Hypothesis 3

System degradation as measured by the difference in seconds for standard comparison programs operating with and without the software recovery system will not be greater than 0.1 second.

This hypothesis was partialy confirmed and rejected for two of the benchmarks. The results below will show the benchmarks that indicate significance.

The program code in Figure 22 was used to measure time differences between eight processes. Observation were made first without CRAM loaded and then with CRAM loaded and running in the background.

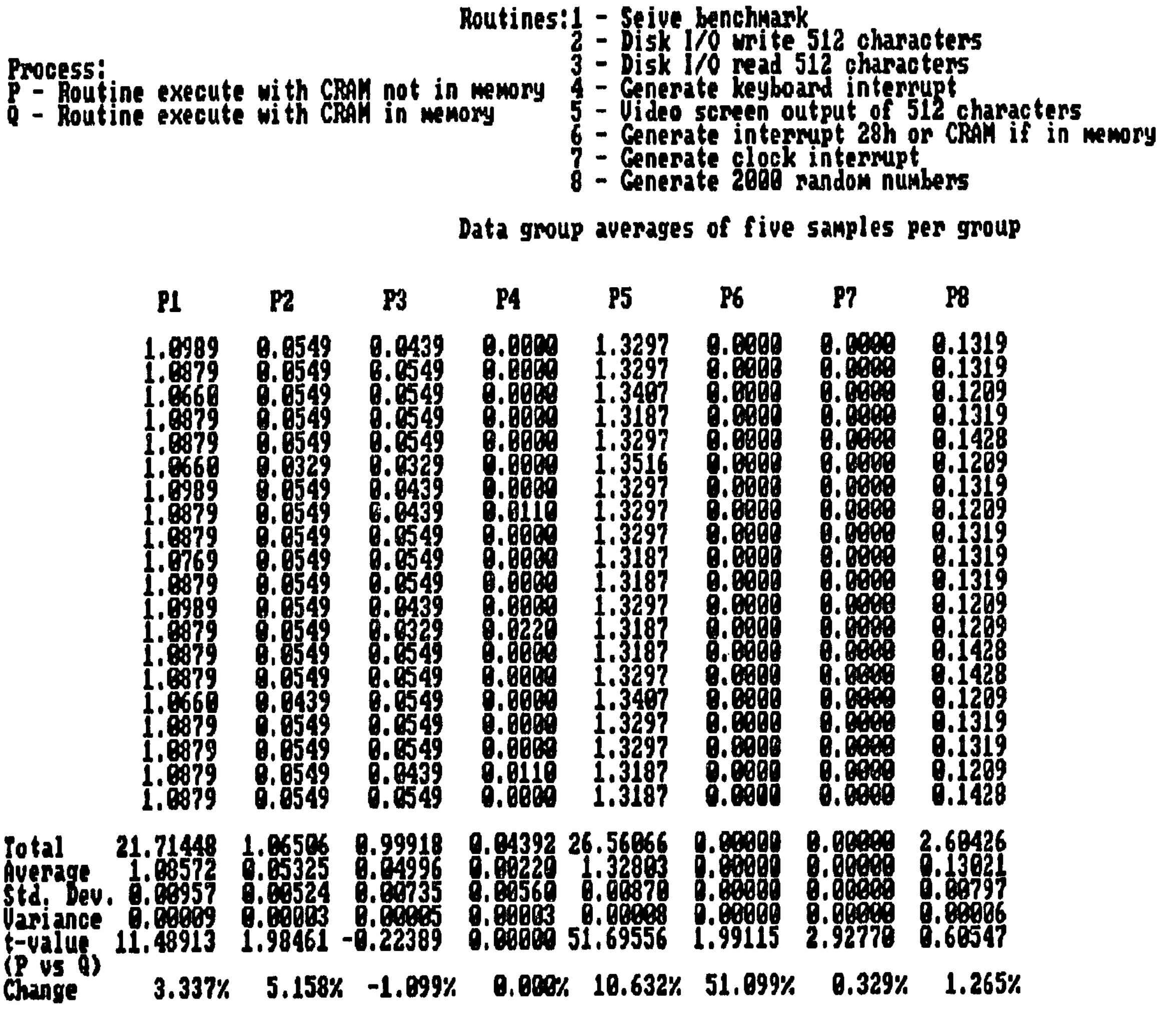
Time delay

During the saving of memory all other processes that were in operation paused for a brief moment. This delay was noticeable because of the speed of the IBM-PC XT. This time will vary with the speed of the micro-processor. The delay time also depends on the number of changed segments in memory. The type of disk used, that is the size and access time, also contributed to the length of the delay.

There is also a time delay due to CRAM operating in the background. Figure 23 and Figure 24 show the difference in execution time for five processes when CRAM is in memory and when not in memory.

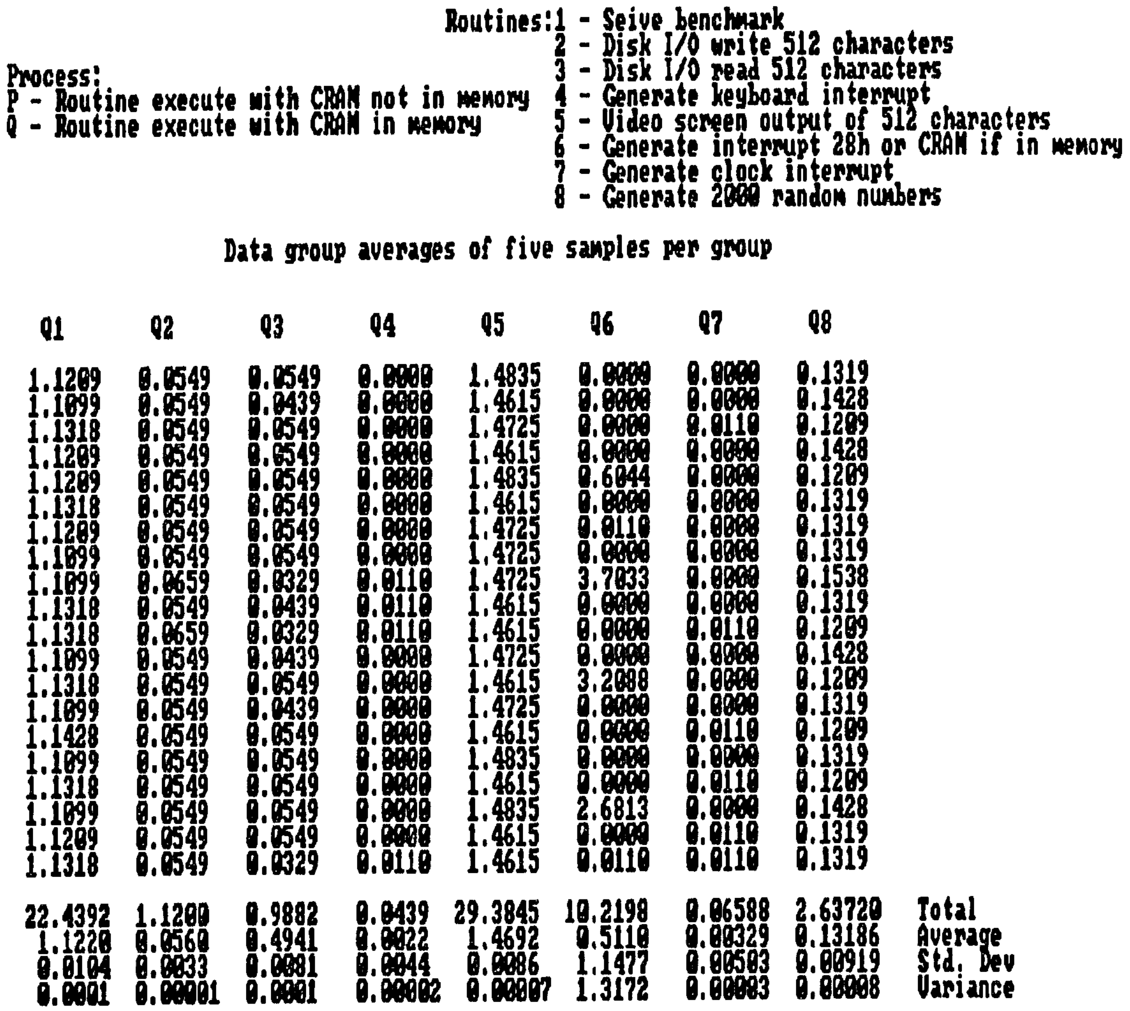
```
#include <stdio.h>
#include <bios.h>
#define true 1
#define false 0
#define size 8190
#define sizep1 8191
char flags[sizep1];

int main(void)
{
  register int iter;
  register int i, k, prime, count;
  char tempstr[10];
  int t1, t2;
  printf("Enter the number of iterations desired: ");
  gets(tempstr);
  iter = atoi(tempstr);
  while( iter--)
  {
    printf("\nIteration %2d", iter);
    count = 0;
    t1 = biostime (0,0);
    for  (i = 0; i < = size; i+ +) flags[i] = true ;
    for ( i = 0; i< = size; i+ +)
      {
        if (flags[i])
          {
            prime = i + i + 3;
            k = i + prime;
            while ( k < = size )
             {
               flags[k] = false;
               k + = prime;
             }
            count+ +;
          }
      }
 t2 = biostime (0, 0);
```

Figure 22. Benchmark program code



```
                                         Routines:1 - Seive benchmark
                                                  2 - Disk I/O write 512 characters
Process:                                          3 - Disk I/O read 512 characters
P - Routine execute with CRAM not in memory       4 - Generate keyboard interrupt
Q - Routine execute with CRAM in memory           5 - Video screen output of 512 characters
                                                  6 - Generate interrupt 28h or CRAM if in memory
                                                  7 - Generate clock interrupt
                                                  8 - Generate 2000 random numbers

                                         Data group averages of five samples per group

             P1        P2        P3        P4        P5        P6        P7        P8

          1.0989    0.0549    0.0439    0.0000    1.3297    0.0000    0.0000    0.1319
          1.0879    0.0549    0.0549    0.0000    1.3297    0.0000    0.0000    0.1319
          1.0660    0.0549    0.0549    0.0000    1.3407    0.0000    0.0000    0.1209
          1.0879    0.0549    0.0549    0.0000    1.3187    0.0000    0.0000    0.1319
          1.0879    0.0549    0.0549    0.0000    1.3297    0.0000    0.0000    0.1428
          1.0660    0.0329    0.0329    0.0000    1.3516    0.0000    0.0000    0.1209
          1.0989    0.0549    0.0439    0.0000    1.3297    0.0000    0.0000    0.1319
          1.0879    0.0549    6.0439    0.0110    1.3297    0.0000    0.0000    0.1209
          1.0879    0.0549    0.0549    0.0000    1.3297    0.0000    0.0000    0.1319
          1.0769    0.0549    0.0549    0.0000    1.3187    0.0000    0.0000    0.1319
          1.0879    0.0549    0.0549    0.0000    1.3187    0.0000    0.0000    0.1319
          1.0989    0.0549    0.0439    0.0000    1.3297    0.0000    0.0000    0.1209
          1.0879    0.0549    0.0329    0.0220    1.3187    0.0000    0.0000    0.1209
          1.0879    0.0549    0.0549    0.0000    1.3187    0.0000    0.0000    0.1428
          1.0879    0.0549    0.0549    0.0000    1.3297    0.0000    0.0000    0.1428
          1.0660    0.0439    0.0549    0.0000    1.3407    0.0000    0.0000    0.1209
          1.0879    0.0549    0.0549    0.0000    1.3297    0.0000    0.0000    0.1319
          1.0879    0.0549    0.0549    0.0000    1.3297    0.0000    0.0000    0.1319
          1.0879    0.0549    0.0439    0.0110    1.3187    0.0000    0.0000    0.1209
          1.0879    0.0549    0.0549    0.0000    1.3187    0.0000    0.0000    0.1428

Total     21.71448   1.06506   0.99918   0.04392 26.56066   0.00000   0.00000   2.60426
Average    1.08572   0.05325   0.04996   0.00220  1.32803   0.00000   0.00000   0.13021
Std. Dev.  0.00957   0.00524   0.00735   0.00560  0.00870   0.00000   0.00000   0.00797
Variance   0.00009   0.00003   0.00005   0.00003  0.00008   0.00000   0.00000   0.00006
t-value   11.48913   1.98461  -0.22389   0.00000 51.69556   1.99115   2.92770   0.60547
(P vs Q)
Change      3.337%    5.158%  -1.099%    0.000%  10.632%   51.099%    0.329%    1.265%
```

Figure 23. Delay times for some benchmark functions



Routines:1 - Seive benchmark
2 - Disk I/O write 512 characters
3 - Disk I/O read 512 characters
4 - Generate keyboard interrupt
5 - Video screen output of 512 characters
6 - Generate interrupt 28h or CRAM if in memory
7 - Generate clock interrupt
8 - Generate 2000 random numbers

Process:
P - Routine execute with CRAM not in memory
Q - Routine execute with CRAM in memory

Data group averages of five samples per group

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	
1.1209	0.0549	0.0549	0.0000	1.4835	0.0000	0.0000	0.1319	
1.1099	0.0549	0.0439	0.0000	1.4615	0.0000	0.0000	0.1428	
1.1318	0.0549	0.0549	0.0000	1.4725	0.0000	0.0110	0.1209	
1.1209	0.0549	0.0549	0.0000	1.4615	0.0000	0.0000	0.1428	
1.1209	0.0549	0.0549	0.0000	1.4835	0.6044	0.0000	0.1209	
1.1318	0.0549	0.0549	0.0000	1.4615	0.0000	0.0000	0.1319	
1.1209	0.0549	0.0549	0.0000	1.4725	0.0110	0.0000	0.1319	
1.1099	0.0549	0.0549	0.0000	1.4725	0.0000	0.0000	0.1319	
1.1099	0.0659	0.0329	0.0110	1.4725	3.7033	0.0000	0.1538	
1.1318	0.0549	0.0439	0.0110	1.4615	0.0000	0.0000	0.1319	
1.1318	0.0659	0.0329	0.0110	1.4615	0.0000	0.0110	0.1209	
1.1099	0.0549	0.0439	0.0000	1.4725	0.0000	0.0000	0.1428	
1.1318	0.0549	0.0549	0.0000	1.4615	3.2088	0.0000	0.1209	
1.1099	0.0549	0.0439	0.0000	1.4725	0.0000	0.0000	0.1319	
1.1428	0.0549	0.0549	0.0000	1.4615	0.0000	0.0110	0.1209	
1.1099	0.0549	0.0549	0.0000	1.4835	0.0000	0.0000	0.1319	
1.1318	0.0549	0.0549	0.0000	1.4615	0.0000	0.0110	0.1209	
1.1099	0.0549	0.0549	0.0000	1.4835	2.6813	0.0000	0.1428	
1.1209	0.0549	0.0549	0.0000	1.4615	0.0000	0.0110	0.1319	
1.1318	0.0549	0.0329	0.0110	1.4615	0.0110	0.0110	0.1319	
22.4392	1.1200	0.9882	0.0439	29.3845	10.2198	0.06588	2.63720	Total
1.1220	0.0560	0.4941	0.0022	1.4692	0.5110	0.00329	0.13186	Average
0.0104	0.0033	0.0081	0.0044	0.0086	1.1477	0.00503	0.00919	Std. Dev
0.0001	0.00001	0.0001	0.00002	0.00007	1.3172	0.00003	0.00008	Variance

Figure 24. Delay times for some benchmark functions contd.

The graphs in Figures 25 thru 27 give a graphical picture of the differences. The processes are:

1 Numerical calculation using the well known sieve routine.

2 Disk I/O write of 512 bytes.

3 Disk I/O read of 512 bytes.

4 Scan of the keyboard.

5 Video out of 512 bytes.

6 Scan of interrupt 28h (a call to CRAM if installed) .

7 Generate clock interrupt

8 Generate 2000 random numbers

Two measurements were taken of the time to execute each process, one while CRAM was installed (Q), and one when CRAM was not installed (P).

A degradation of 3.23% for the sieve numerical calculation, 1.25% for the random number generation, and 9.6% for the video display operation was observed. Very little effect was noted for most of the other operation except for times when CRAM's presence may delay the clock interrupt by 0.05 seconds. The data collected from the random number generating process show some overlapping which indicate a need to test the means for differences. A t-test was done on this data and the t-values are shown in Figure 23. There was no significant difference between the mean delay time for generating random numbers whether CRAM was running or not. Thus it

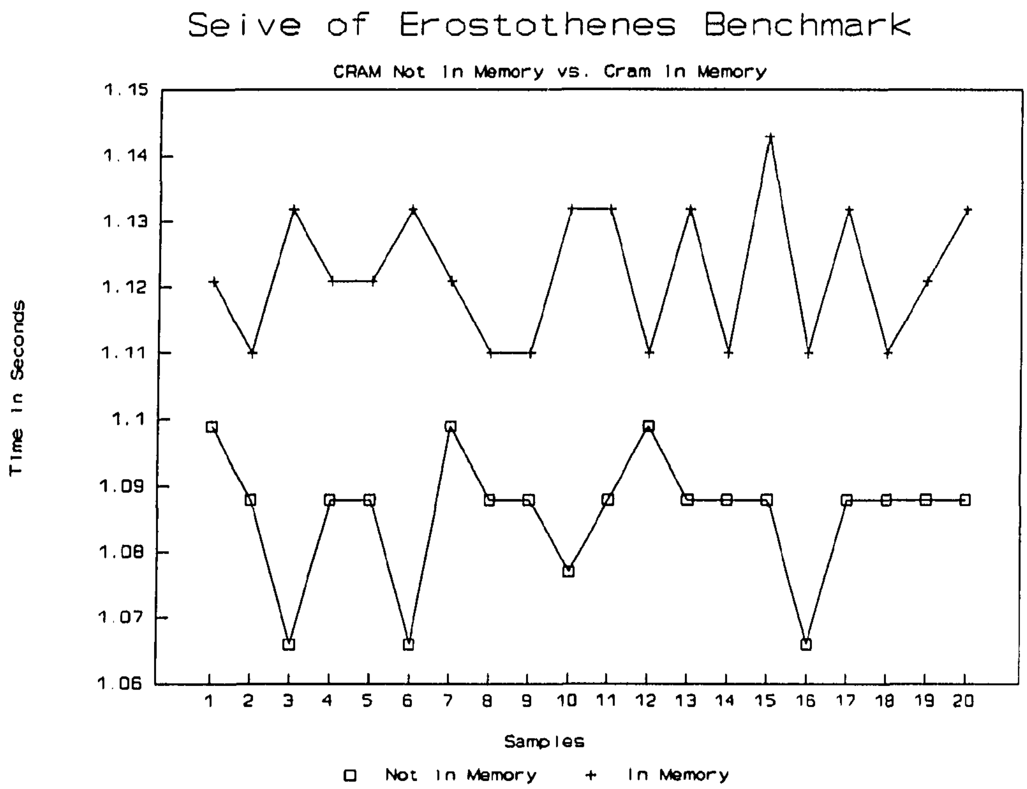


Figure 25. Graph of sieve numerical test

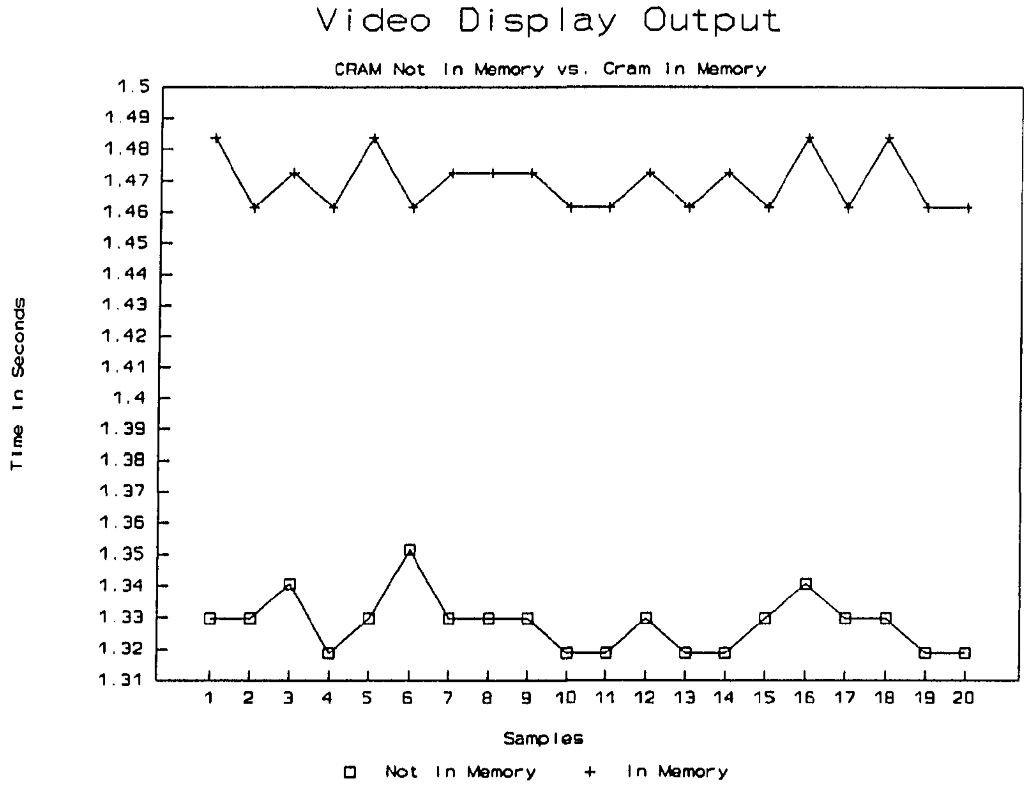


Figure 26. Graph of Video display test

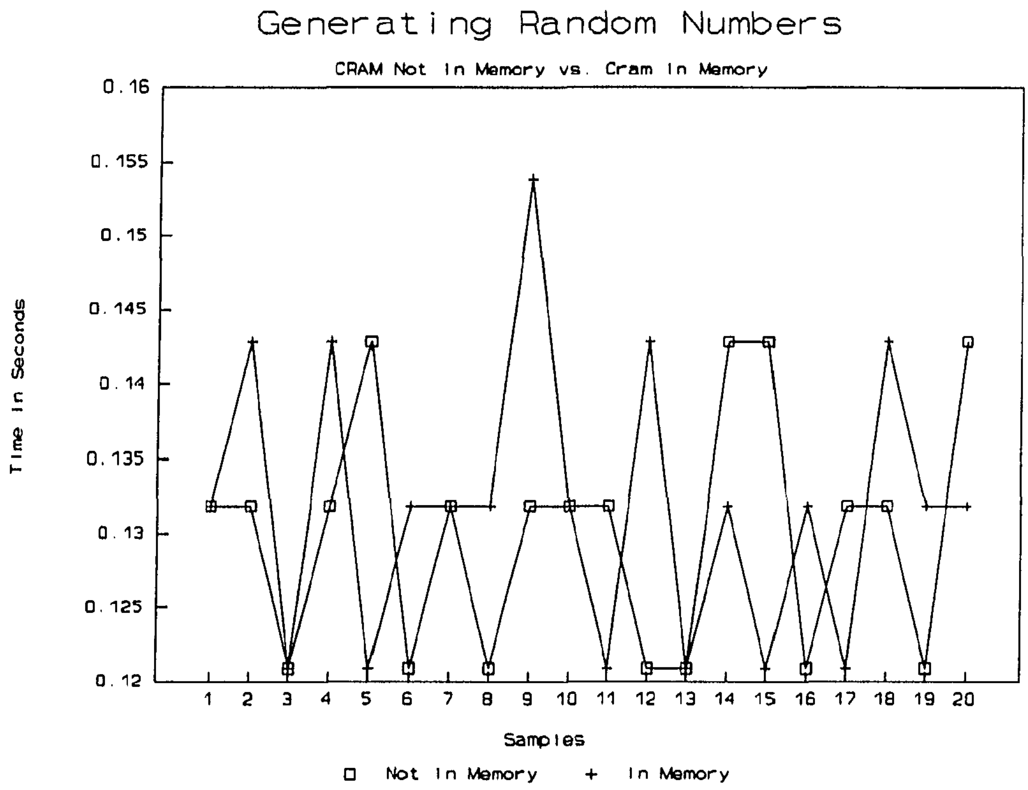


Figure 27. Graph of random numbers generation test

can be resolved that CRAM does not affect program execution that involves random number generation.

In summary, the results of this study demonstrated that

a) a method can be used to save data to disk for easy access,

b) software can be used to implement a hardware solution, although some hardware and software problems need to be resolved or circumvented, and

c) the system suffers a negligible speed degradation of less than 0.01 second.

CHAPTER V. SUMMARY AND RECOMMENDATIONS

The intent of this study was to develop a software based system to restore the computer's memory after a catastrophic failure, and provide a system to replace the existing hardware systems such as Uninterruptible Power Supplies. The researcher was faced with three fundamental questions discussed below:

Discussion to Question 1

Can a software system be developed that will provide recovery from a system failure?

A software program was developed that constantly monitors the computer's Random Access Memory (RAM). Memory is saved to disk for future retrieval if a change was detected. The software developed is called Constant Random Access Memory (CRAM). CRAM was designed to be expandable to include the use of extended memory, expanded memory and the DOS high memory block. Although this project did not include these areas in its development, it was designed to incorporate these techniques in future development. One question that was posed by this method of saving memory is "what would result from a power failure in the middle of a memory save process?" The present method will not be affected to any great extent by such occurrence because memory, where most of the program's critical data and information is located, is saved first. The program's code can be reloaded from disk. This question could, however, be answered in a more practical way by recommending future development to this project to include the following procedure: memory could be swapped to a work area of the disk and later swapped into CRAM. This would provide a complete safety buffer which means that memory could always be restored free of error, in one period required to save all segments of memory that were changed. This method was not included in this project at this time because this implementation was for the IBM PC-XT, which is a relatively slow machine. The introduction of too many disk accesses would slow the system down considerably. The faster machines such as the IBM PC-AT or models 386 and 486 could be tested for degradation in speed if such a procedure were introduced into the overall processing.

Implications for future research

This software needs much improvement in its design to include the newer model machines. Although attempt was made not to write code that is machine dependent this was violated in some instances to speedup development. The checksum method used in this study could be substituted with the tagged memory technique suggested by Adams (1991). This would reduce the memory scanning time. The system is design with redundancy checking that could be eliminated to decrease the overall scan time.

CRAM could be interfaced with many other application and DOS itself. DOS could be merged with codes that are implemented in the windows environment. Another growing concern among industries is data safety in local area networks. CRAM could be enhanced to operate at each work-station. Local area network with a star bus configuration rely heavily on the server for data reliability and availability. CRAM could be used to improve reliability and availability.

Since most workable codes compiled to an object file can be transferred to firmware and the code for CRAM is not machine dependent, it is the author's belief that future development of DOS or the ROM BIOS could include CRAM as part of its operating system code. Melear (1986), noted that the improve technology in EEPROM will play an important roll in the development of firmware.

CRAM could also be used as an addition security to existing UPS system, with proper interfacing.

This study could be extended to other systems such as Unix and OS/2 .

Discussion to Question 2

What problems exist in achieving a software-only recovery system?

Development problems

The use of DOS, a single user system, to simulate a multi-user system using TSR programs, poses some interesting and challenging problems. It is difficult to restore a system using a TSR program since the TSR program needs to be running at the same time. This problem was overcome by first loading CRAM at the same relative position for both saving and restoring memory. CRAM was loaded immediately after the command interpreter. This enabled the restoration to only start at memory location after the interpreter. Memory locations before the interpreter were not saved or restored. However, some important areas like the Bios area and the interrupt vector table, stored in lower memory, were saved and restored.

Secondly, the restore routine was executed as a separate interrupt. This allowed the routine to execute after the TSR was fully loaded into the system.

Thirdly, all areas of memory are constantly monitored and transferred to CRAM if memory was changed by software running in the system.

Another problem faced was the context switching after a restoration. Although the Program's Segment Prefix for the TSR replaced the PSP of the interrupted process, the system would hang when attempting to load a new program. The problem was discovered in the way DOS updates the MCB. There was no information available on the method used by DOS to update the MCB. This problem was alleviated by the constant monitoring of the MCB and the updating of the MCB at the time of restoration. The routine in Figure 28 and Figure 29 were used to repair the MCB.

The third problem was the unusual and unexpected failure of the interrupted process to release the memory it occupied when terminated. This occurred after a system restoration. This problem was solved by monitoring the terminating address stored in the programs PSP. This address was updated if changed unexpectedly.

Turbo C version 2.0, used in the development of this project also introduced some problems that may be caused from bugs in the compiler itself. It was discovered that the use of function STRSTR and STRUPR to check for the character "R" in the command line variable ARGV gives unpredictable side effects. It was later determined that this phenomena contributed to the trashing of the MCB and the program's inadvertent lack of releasing memory when terminated. This could be as catastrophic as a power outage without a backup or equivalent to the 1989 disruption of telephone service because of a problem in the telephone software code.

```
set_mcb_chain(BYTE *buffer)
{
  MCB            far  *mcb;
  struct MCB_TBL far  *tblptr;
  int                 i = 0, j=0;

  j = atoi(&buffer[511]);
  for (;;)
     {
     tblptr = (struct MCB_TBL far *)(&buffer[i]);
     mcb = MK_FP(tblptr->addr,0);
     mcb->type  = tblptr->mcb.type;
     mcb->owner = tblptr->mcb.owner;
     mcb->size  = tblptr->mcb.size;
     if (mcb->type == 'M' && j > 0)
       {
        i+=sizeof(struct MCB_TBL);
        j--;
       }
      /* mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0); */
     else
        return (tblptr->mcb.type == 'Z');
     }
}
```

Figure 28. Code used to set MCB

```
int get_mcb_chain(MCB far * mcb, BYTE  *buffer)
{
   MCB_TBL far *tblptr;
   int i = 0, j=0;

   for (;;)
     {
      tblptr = (MCB_TBL far *)(&buffer[i]);

      tblptr->mcb.type = mcb->type;
      tblptr->mcb.owner = mcb->owner;
      tblptr->mcb.size = mcb->size;

      tblptr->addr = (WORD) FP_SEG(mcb);

      if (mcb->type == 'M')
        {
         mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
         i+=sizeof(struct MCB_TBL);
         j++;
        }
      else
        {
         itoa(j, &buffer[511],16);             /*Store the number of MCBs */
         return (j);        /*Return the number of MCBs present */
        }
     }
}
```

Figure 29. Code to get MCB

Implementation problems

Nine commercial programs were tested for restoration using CRAM. It was noted that about 75% of the time the system was restored. However, only 33% were full restoration. This could be due to the way in which both WordStar 2000, WordPerfect and Turbo C used the timer interrupt. It was observed that when the display error symbol indicated, in most cases of partial restoration, that the timer interrupt was active. It may be That the timer interrupt was used improperly or should not be interrupted.

Implications for future research

The development of this system for multi-user multi-task machine would eliminate the problems associated with using a TSR program. The location of CRAM in memory would be still important, since all of memory must be restored including the area occupied by CRAM. CRAM must not be over-written during the restoration process.

The second area that may affect future development is the version of Turbo C compiler that is used. The problem in the compiler that caused the executable code to trash the Memory Control Block (MCB) should be fixed.

Discussion to Question 3

What is the degradation in application program performance when utilizing a software recovery system?

The time delay due to CRAM operating in the background, based on the difference in execution time for five processes when CRAM is in memory and when not in memory was analyzed.

The processes are:

1 Numerical calculation using the well known sieve routine.

2 Disk I/O write of 512 bytes.

3 Disk I/O read of 512 bytes.

4 Scan of the keyboard.

5 Video out of 512 bytes.

6 Scan of interrupt 28h (a call to CRAM if installed) .

7 Generate clock interrupt

8 Generate 2000 random numbers

Two measurements were taken of the time to execute each process, one while CRAM was installed, and one when CRAM was not installed.

A degradation of 3.3% for the sieve numerical calculation, 1.3% for the random number generation, 5.2% for the disk I/O write operation, and 10.6% for the video display operation was observed. Less than 1% change was noted for most of the other operation except for times when CRAM's presence may delay the clock interrupt by 0.05 seconds. A t-test was done on the data collected from the random number generating process which show some overlapping which indicate a possibility of no differences in the means. There was no significant difference between the mean delay time for generating random numbers whether CRAM was running or not. Therefore, it can be concluded that CRAM does not affect program execution that involves random number generation.

One unresolved finding is the increase in disk read operation when CRAM is in memory. This may be due to the continuous activation of the timer interrupt by CRAM. This interrupt provides a counter for the disk read/write head settling time.

Summary

A great percentage of the time for this study was spent writing the code for the software. This system when implemented could safe-guard an IBM-PC XT computer, under some circumstances, from data loss due to a power failure. This implementation is completely software driven and requires no interaction from the user once installed. This system is transparent to the user and is only visible by symbols displayed on the video screen. For some applications the degradation in speed is negligible but for applications requiring many disk access the operating speed may be noticeable.

REFERENCES

Adams, S. J., Simms, T. (1991). A Tagged Memory Technique for Recovery From Transient Errors in Fault Tolerant Systems. Proceedings - 1990 Real-Time Systems Symposium. by IEEE, p 312-321.

Adams, P. M., Tondo, C. L. (1990). Writing DOS Device Drivers in C. Englewood Cliffs, New Jersey: Prentice Hall, Inc.

Angermeyer, J., Jaeger, K. (1986). The Waite Group's MS-DOS Developer's Guide. Indianapolis, Indiana: Howard W. Sams & Company.

Bacon, D. F. (1991). Transparent Recovery in Distributed Systems Position Paper. Operating Systems Review (ACM), 25 (2), 91-94.

Bailey, R. W. (1983). Human Error in Computer Systems. Englewood Cliffs, New Jersey: Prentice Hall, Inc.

Barkakati, N. (1989). Turbo C Bible. Indianapolis Indiana. The Waite Group, Inc. Howard W. Sams & Company.

Barnes, C. C., Coleman, A., Showalter, J. M., Walker, M. L. (1991). VM/ESA Support for Coordinated Recovery of Files. IBM Systems Journal, 30 (1), 107-125.

Bassiouni, M. A. (1986). System and Program Models of Storage Allocation For Reducing Seek Delay. The Computer Journal. 29 (1), 47-51.

Beaudin, T. (1992). Unattended Backup Keeps IS Staff Levels Low. Systems 3X/400, January p 53-54.

Beizer, B. (1988). The Frozen Keyboard. Blue Ridge Summit, Pennsylvania: TAB Professional and Reference Books.

Belli, F., Jedrzejowicz, P. (1991). An Approach to the Reliability Optimization of Software with Redundancy. IEEE Transactions on Software Engineering, 17 (3), 310-312.

Bellin, D., Suchman, S. (1990). The Structured Systems Development Manual. Englewood Cliffs, New Jersey: Prentice Hall, Inc.

Bennett, R. B., Bitner, W. J., Musa, M. A., Ainsworth, M. K. (1991). Systems Management for Coordinated Resource Recovery. IBM Systems Journal, 30 (1), 90-106.

Beynon-Davies, P. (1989). Information Systems Development. Houndmills, Basingstoke, Hampshire and London: Macmillan Education Ltd.

Boling, D. (1992). Strategies and Techniques for Writing State-of-the-Art TSRs that Exploit MS-DOS 5. Microsoft Systems Journal, 7 (1), 41-59.

Bondavalli, A., Simoncini, L. (1990). Failure Classification With Respect to Detection. Proceedings of the 2nd IEEE Workshop on Future Trends of Distributed Computing Systems. by IEEE, p 47-53.

Borland, (1988). Turbo C Version 2.0 (TC). Scotts Valley, California: Borland International, Inc.

Borland, (1988). Turbo Assembler Version 2.0 (TASM). Scotts Valley, California: Borland International, Inc.

Boyle, J., Butler, R., Disz, T., Glickfeld, B., Lusk, E., Overbeek, R., Patterson, J., Stevens, R., (1987). Portable Programs for Parallel Processors. New York, New York: Holt, Rinehart and Winston, Inc.

Brown, R., Kyle, J. (1991). PC Interrupts: A Programmer's Reference to BIOS, DOS, and Third-Party Calls. Addison-Wesley Publishing Company, Inc.

Chantico Publishing Company, Inc. (1991). Disaster Recovery Handbook. Blue Ridge Summit Pennsylvania: Tab Professional and Reference Books.

Chu, J. L., Torabi, H. R., Towler, F. J. (1991). A 128kb CMOS Static Random-Access Memory. IBM Journal of Research and Development, 35 (3), 321-329.

Collins, G., Blay, G. (1982). Structured Systems Development Techniques: Strategic Planning to System Testing. Marshfield, Massachusetts: Pitman Publishing Inc.

Colvin, N. J. (1989). System Bios for IBM PC/XT/AT Computers and Compatibles. Phoenix Technologies Ltd. Addison-Wesley Publishing Company, Inc.

Crecine, J. P. (1986). The Next Generation of Personal Computers. EDUCOM Bulletin, Spring, 2-10.

Daniels, B. K. (1987). Achieving Safety and Reliability with Computer Systems New York, New York: Elsevier Science Publishing Co., Inc.

De Peyster, D (1989). On The Wild Side. PC Resource, May, 6.

Dettmann, T. R. (1988). DOS Programmer's Reference. Carmel, Indiana: Que Corporation.

Dettmann, T. R. (1989). DOS Programmers's Reference. 2nd Edition. Carmel, Indiana: Que Corporation.

DeYoung, B. (1984). Resource Sharing of Micro Software, or, What Ever Happened to All That CP/M Compatibility? Microcomputers for Information Management. 1 (4), 313-324.

Duncan, R. (1988). Advanced MSDOS Programming, Second Edition. Redmond, Washington: Microsoft Press, A Division of Microsoft Corporation.

Ely. D. P. (1990). Trends and Issues in Educational Technology. Tech Trends, 35 (4), 9-11.

Falk, H. (1987). New Tools Help Exterminate Software Bugs. Computer Design, October, 52-58.

Friedman, A. L., Cornford, D. S. (1989). Computer Systems Development: History, Organization and Implementation. New York, New York: John Wiley & Sons, Inc.

Garcia-Molina, H., Polyzois, C. A. (1990). Issues in Disaster Recovery. Digest of Papers Thirty Fifth IEEE Computer Society International Conference COMPCON 89. by IEEE, p 573-577.

Garland. V. E. (1990). Computers in 2001: Preparing Educational Administrators. Tech Trends, 35 (3), 17-22.

Gibbons, T. (1976). Integrity and Recovery in Computer Systems. Rochelle Park, New Jersey: Hayden Book Company.

Grossman, C. P. (1985). Cache-DASD Storage Design for Improving System Performance. IBM Systems Journal. 24 (3-4), 316-334.

Han, S. H., Malek, M. (1986). Two-Dimensional Multiple-Access Testing for Random-Access Memories. IEEE, p 248-251.

Hetzel, W. (1984). The Complete Guide to Software Testing Wellesley, MA: Information Sciences, Inc.

Hisano, T. (1986). Transparent Memory: A Hardware Solution to the Memory Conflict Problem. Systems and Computers in Japan, 17 (11), 100-108.

Hyde, R. L. (1988). Overview of Memory Management. Byte, April, p 219-225.

Johnson, D. B., Zwaenepoel, W. (1991). Transparent Optimistic Rollback Recovery. Operating Systems Review (ACM), 25 (2), 99-102.

Kaczeus, S. (1990). Disk Reliability is a Function of Design as Well as Manufacture. Computer Technology Review, 10 (9), 59-60, 62-63.

King, R. P., Halim, N., Garcia-Molina, H. Polyzois, C. A. (1990). Overview of Disaster Recovery for Transaction Processing Systems. Proceedings - International Conference on Distributed Computing Systems, by IEEE, p 286-293.

King, R. A. (1987). The MS-DOS Handbook, Alameda, CA: SYBEX Inc.

Kinoshita, K., Saluja, K. K. (1986). Built-in Testing of Memory Using An On-chip Compact Testing Scheme. IEEE Transactions on Computers, C-35 (10) 862-870.

Klopp, C. (1990). Vulnerability Awareness Improves Contingency Planning. Computers & Security, 9 (4), 309-311.

Koren I., Koren, Z., Su, S. Y. H. (1986). Analysis of a Class of Recovery Procedures. IEEE Transactions on Computers, C-35 (8), 703-711.

Kyle, J. (1992). Electronic mail communication, Compuserve, 76703,762.

Lengefeld, H. C. (1990). Battery-Free Rotary Options Are Available For Today's UPS. Computer Technology Review, Special Winter Issue, 132-135.

Levary, R. R., Edwards, W. D. (1986). Analyzing The Impact of Adding a New Software System on Main Memory Usage. Computer Journal, 29 (6), 522-526.

Levy, E. (1991). Incremental Restart. Proceedings - International Conference on Data Engineering, by IEEE, p 640-648.

Levy, E., Silberschatz, A. (1990). Log-Driven Backups: A Recovery Scheme for Large Memory Database Systems. Proceedings of the 5th Jerusalem Conference on Information Technology, by IEEE, p 99-109.

Lion, K. (1990). Digital Audio Tape as a Backup for the AS/400. Computer Technology Review, 10 (16), 83-84, 88.

Littlewood, B. (1987). Software Reliability Boston Massachusetts: Blackwell Scientific Publications.

Lopez, A. M. Jr. (1992). Mass Storage and Communication: The Big Byte in The 90s. Interface, 14 (1), 47-54.

Lua, K. T. (1990). Failure of Instruction Prefetching of 8088/286/386 Microprocessors in XT/AT systems. Microprocessing and Microprogramming, 29 (2), 97-106.

Lucente, M. A., Harris, C. H., Muir, R. M. (1991). Memory System Reliability Improvement Through Associative Cache Redundancy. IEEE Journal of Solid-State Circuits, 26 (3), 404-409.

Mace, P. (1988). The Paul Mace Guide to Data Recovery. New York, New York: Simon & Schuster, Inc.

MacKenzie, F. B. (1987). Automated Secondary Storage Management. Annually History of Computer. 9 (1), 29-35.

Maslak, B. A., Showalter, J. M., Szczygielski, T. J. (1991). Coordinated Resource Recovery in VM/ESA. IBM Systems Journal, 30 (1), 72-89.

Mason, R. M. (1984). Current and Future Microcomputer Capabilities: Selecting the Hardware. Microcomputers for Information Management. 1 (1), 1-13.

Matick. R. E. (1986). Impact of Memory Systems on Computer Architecture and System Organization. IBM Systems Journal. 25 (3-4), 274-305.

Melear, C. (1986). Applications For Microcomputers With E**2PROM. Conference Record - Electronic. p 12.

Miller, A. R. (1986). Memory Manipulations. Part 1: Arabic Versus Roman. Part 2: Adjusting Memory Size. Byte, 11 (11) 232-234, 236, 238-245.

Mueller, S. (1991). Guide To Data Recovery. Carmel, Indiana: Que Corporation.

Newman, I. A., Stallard, R. P., Woodward, M. C. (1987). Hybrid Multiple Processor Garbage Collection Algorithm. Computer Journal, 30 (2), 119-127.

Newport, D. F., Alley, G. T., Bryan, W. L., Eason, R. O., Bouldin, D. W. (1986). IEEE, p 578-581.

Norton, P. (1989). The Norton Utilities Software Package.

Norton, P. (1985). The Peter Norton Programmer's Guide to the IBM PC. Redmond, Washington: Microsoft Press, A division of Microsoft Corporation.

Phillips, D. (1986). COMPLEAT Multiprocessor System. Electronic Products, 29 (8), 75-81.

Purdum, J. (1989). C Programmer's Toolkit. Carmel, Indiana: Que Corporation.

Ramsey, H. (1990). Telephone conversation about the PowerSave at IIT Power Systems.

Rich, M. (1986). Method of Flexible Catch RAM Display For Memory Testing. Digest of Papers - International Test Conference, by IEEE, p 222.

Sando, S. (1985). Achieving Nanosecond Cache Performance With GAAS. Wescon Conference Record, A paper Distributed by Western Periodicals Co. for IEEE, 6 pages.

Sargent III, M., Shoemaker, R. (1986). The IBM PC from the Inside Out, Revised Edition. Addison-Wesley Publishing Company, Inc.

Schildt, H. (1988). C: Power User's Guide. Berkeley, California: McGraw-Hill, Inc.

Schulman, A., Michels, R. J., Kyle, J., Paterson, T., Maxey, D., & Brown R., (1990). Undocumented DOS: A Programmer's Guide to Reserved MS-DOS Functions and Data Structures. New York, New York: Addison-Wesley Publishing Company, Inc.

Schustack, S. (1989). Variation in C. Redmond, Washington: Microsoft Press, A division of Microsoft Corporation.

Sinutko, M. (1987). Memory Access to Multiple-Sensitivity Information. Proceedings of the Seventeenth International Symposium on Multiple-Valued Logic. by IEEE. p 109-116.

Smith, D. D., Bulgren, W. G. (1987). Memory Management Algorithms for Buffer Pool Systems. IEEE Computer Society, p 83-89.

Spector, A. Z. (1984). Computer Software for Process Control. Scientific American, 25 1 (3), 174-186.

Tam, V., Hsu, M. (1990). Fast recovery in Distributed Shared Virtual Memory Systems. Proceedings of the 10th International Conference on Distributed Computing Systems. by IEEE, p 38-45.

Tammaru, T. (1985). Memory Addressing Arrangement. Technical Digest AT&T Technology. 77, p 47.

Tasch, U., Sheridan, T. B. (1990). On-Line Model Based Topographic Search of System Failures. Journal of the Franklin Institute, 327 (2), 251-258.

Upadhyaya, S. J. (1990). Rollback Recovery in Real-Time Systems with Dynamic Constraints. Proceedings - IEEE Computer Society's International Computer Software & Applications Conference. by IEEE, p 524-529.

Wadlow, Thomas A. (1987). Memory Resident Programming on the IBM PC. Addison-Wesley Publishing Company, Inc.

Waite, M. (1988). The Waite Group's MS-DOS Developers and Power Users. Indianapolis, Indiana. Howard W, Sams & Company.

Wallace, R. H., Stockenberg, J. E., Charette, R. N. (1987). A Unified Methodology for Developing Systems. New York, New York: Intertext Publications, Inc.

Yanney, R. M., Hayes, J. P. (1986). Distributed Recovery in Fault-Tolerant Multiprocessor Networks. IEEE Transactions on Computers. C-35 (10), 871-878.

ACKNOWLEDGMENTS

I would like to sincerely thank the many people who helped me throughout my doctoral program. Especially, I would like to thank my major professor, Dr. William Miller, for his advice and timely suggestions which contributed to the successful completion of this dissertation. His encouragement and guidance were gratefully appreciated.

I also want to thank the Department Chair, Dr. John Dugger, along with the other members of my committee. In addition, I am grateful to the office staff in the Department of Industrial Education and Technology for their assistance while I was away from campus.

Finally, I am indebted to my lovely wife, Dr. Carol A. S. Brevett, for her contribution in editing this manuscript. She showed continual patience, perseverance and care while the computer diverted my thoughts and dreams at night and held me hostage during the day.

'APPENDIX A. INSTALLATION GUIDE

Introduction

CRAM is a memory resident software tool designed to replace a standard UPS system. This software, when installed on any IBM-PC XT or compatible computer, will monitor the computer's main memory and save it to disk when changes are detected. This software uses no additional hardware and a small amount of memory overhead. To start CRAM you must first have a PC-XT equipped with a hard disk system. CRAM can use a floppy disk but disk access speed will decrease dramatically. The floppy disk system can be used for transferring information to other systems as will be seen later in this manual. CRAM must be installed on the system using the installation program. This version was developed for the XT and thus may give unpredictable results when running on any other system. The video drivers supported are MDA, CGA and EGA. Programs running in graphic mode may experience problems in the restoration of the video screen. Although the program may be totally restored, the screen images may not be completely restored.

System Preparation Prior to CRAM Installation

The system must be prepared before CRAM is installed in order to get the best performance and also to prevent CRAM from failing to install because the free clusters on the disk are not next to each other but scattered throughout the disk (fragmented).

First obtain a copy of either the Norton disk optimizing utility or any other disk organizing utility. It is a good idea to run either the Norton Disk Doctor utility first to analyze the disk or DOS check disk utility (CHKDSK). Now run the optimization program to organize the disk using the full optimization option according to the software you choose. This process may take awhile depending on the size of the hard disk.

CRAM Installation

After the optimization process is completed, the Install program which accompanies the CRAM software should be executed. Place the CRAM installation disk in one of the drives. Enter **INSTALL**. This process will take a while again depending on the size of the hard disk. The install program will check the disk to make sure CRAM was not installed before. It will also check for an area of the disk to install the software and the area to save memory. If no errors were encountered the program will show that CRAM was installed successfully and that the installation process is complete. The program will pause for about one second, then it will re-boot the system to load CRAM into the system.

Invoking CRAM

To start CRAM enter **CRAM** at the DOS prompt. CRAM must be supplied with two or more of the switches found under "Switches used with CRAM" below. Once installed feedback from CRAM is evident by a symbolic display character in the upper left corner of the screen and other indicators on the top row of the screen. See Figure 1A in the appendix for the meaning of these symbols. If CRAM was not installed, follow the above directions found under "CRAM installation" before invoking CRAM.

Switches Used With CRAM

CRAM must be supplied with a combination of two or more of the following switches:

-h *key1 key2* hot keys used to force CRAM to save memory

key1 is a code for special keys:

1 = Right Shift

2 = Left Shift

4 = CTRL

8 = ALT

Keys can be combined like 12 for CTRL-ALT

key2 is any other non-special key (regular ASCII characters)

-i to start-up by first saving all of memory and setting some vital parameters

-k to start-up but no initial saving of memory or setting of vital parameters

-l *drive* drive that CRAM will use to save memory

-u to start-up then restore memory to the process stored in CRAM

-x to extract CRAM from memory and reset the system to run without CRAM

-xx to extract CRAM from both memory and the entire disk system

System Requirements

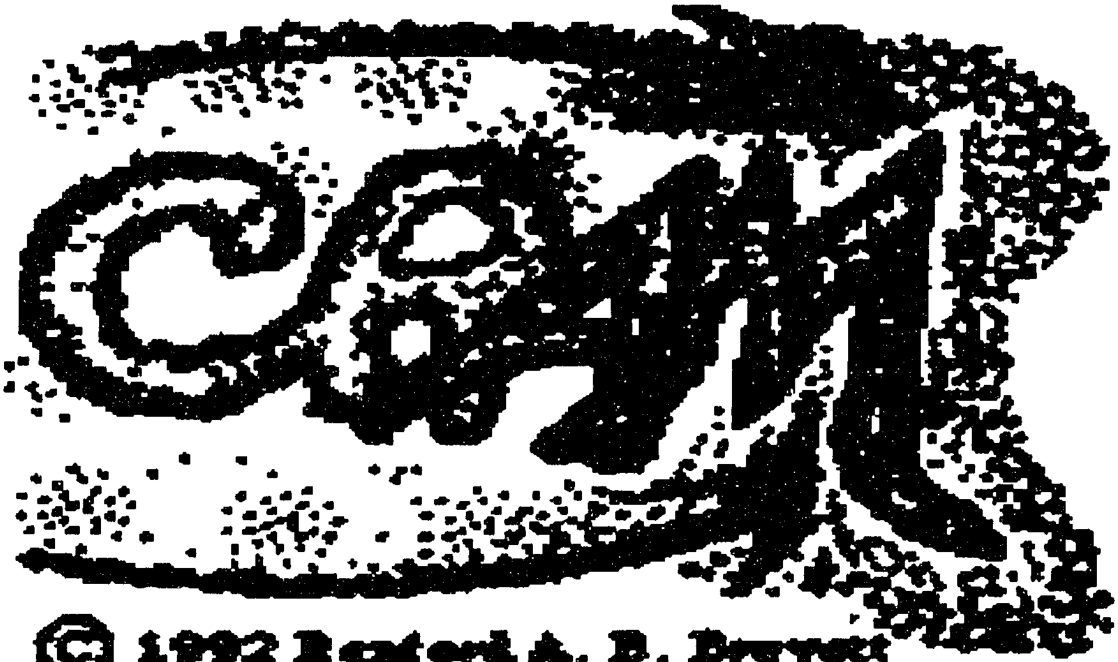
The following are the minimum requirements needed for CRAM to install and operate.

1. An IBM-PC XT or compatible
2. Hard disk with at least 700K bytes of free space
3. At least 512K memory (only about 64k is required for CRAM's operation but most software needs at least 512k to run)

4. MDA, CGA or EGA video adapter

5. At least one floppy drive (3½ or 5¼)

APPENDIX B. USER MANUAL


CAM

Introduction

Guidelines for the use of CRAM do not warrant a users manual, however, the information provided here will help other developers understand the inner working of CRAM. CRAM, unlike other software, is not a utility or a user processing program but a tool that requires almost no user interaction after installation. CRAM uses a Real-time Symbolic Feedback System (RSFS). This is the constant display of the systems processes and error conditions by way of indicators on the screen. There are four sets of indicators. The symbols are shown in the Table below. The first is a one character symbol that tells what area of CRAM is being processed. The second set is a group of flags and a counter. The third is the lower byte of the DOS flag register. The fourth set indicates the areas of memory that are being changed by a process, flagged to be saved or already saved by CRAM.

Some DOS Error Codes Returned by CRAM

ERROR 03	Disk is write protected.
ERROR 04	Could not find disk sector.
ERROR 10	Error in data transfer.
ERROR 20	Diskette controller failed.
ERROR 80	Disk timed out error.

Error Codes Specific to CRAM

CRAM cannot continue

Another program other than CRAM was found to be loaded in memory after the command interpreter. CRAM must be the first file loaded after the command interpreter.

DISK ERROR saving memory

The disk is bad or CRAM is corrupted.

Drive is not a CRAM disk

A drive that was specified in the -L switch does not have a copy of CRAM installed on it. You need to run INSTALL and use the -L switch to indicate the drive to use to install CRAM.

ERROR with SWAP_SAVE

Maybe not enough memory to save information in the DOS Swappable Data Area.

ERROR formatting track

An invalid track number was specified.

ERROR reading sector

The sector may be damaged.

ERROR *errnum* reading Disk Parameter Table (DPT)

The drive specified either has a damaged DPT or does not have one. The drive specified could be invalid or the hard disk is damaged. *Errnum* is the DOS error number that caused this error.

ERROR *errnum* Reading BIOS Parameter Block (BPB)

An attempt was made to read a disk with damaged BPB. The drive specified could be invalid. This error message also reports the drive, head and sector of the last disk access. *Errnum* is the DOS error number that caused this error.

ERROR in Memory Control Block (MCB)

The memory control block has invalid entries. CRAM will fix this error so there is no need to be alarmed.

ERROR writing sector

The sector may be damaged.

CRAM not Installed on disk

CRAM was unable to find the CRAM ID on the disk. It is likely that CRAM was not installed on the disk.

ERROR initializing DOS Swappable Area (SDA)

CRAM could not use the DOS SDA. There may be insufficient memory or CRAM is corrupted.

ERROR saving memory::save_mem

CRAM encounters a problem with the hard disk or disk used to save memory. Either the disk is bad or CRAM was corrupted.

***** See Figure 1A for the symbolic error codes. *****

Re-installing CRAM

If CRAM needs to be reinstalled after de-installation or installed to a newly reformatted disk, the following procedures must be followed:

1. You must be the registered owner of the software.
2. The disk must be the in the same machine.

Process	Symbol	Error
Saving Registers to CRAM	Φ	
Getting Registers and vital CRAM statistics	∩	
Saving video memory to CRAM	none	
Resetting video memory	none	
Saving interrupt vector table to CRAM	φ	
Resetting interrupt vector table	Ω	
Normal CRAM operation in progress	ϵ	
Putting CRAM vital information in CRAM	∞	
Getting CRAM vital information from CRAM	≡	
Resetting memory	δ	
Loading CRAM	ç	
Saving DSA	Σ	
getting DSA from CRAM	Γ	
Restoring DSA	α	
Completion of DSA restoration	ß	

Figure 1A. CRAM's process screen indicators

3. The system must be in its original state when first installed (i.e. has all the peripherals that were present at the time of first installation).

4. If you are the registered owner of the software but 2 and/or 3 above cannot be satisfied call or write:

   Renford A. B. Brevett
   109 Newton Drive
   Bear, DE. 19701
   (302) 325-0876

   You will be given instructions or a new copy of the software.

5. If all of the above was satisfied then you only need to run the installation program (INSTALL) and CRAM will be installed on the hard disk and will re-install itself without any further intervention.

Getting Information About CRAM

The program CRAMINFO.EXE on the CRAM disk allows access to information about CRAM. Below is a sample of the output from CRAMINFO. The information in CRAM can be accessed at anytime.

Editing CRAM

The program CRAMEDIT.EXE on the CRAM disk allows the editing of data in CRAM. This program should only be used by experienced programmers. This tool is useful in editing CRAM which can then be restored with the edited information. If the system was in a critical error state and was saved by CRAM in that state without correction then CRAMEDIT could be used to edit memory before attempting to restore CRAM. Note that CRAM should not be running in memory while editing CRAM.

Transferring CRAM To a Different Disk and/or Computer

CRAM can be used to transfer information between two systems. This feature is useful if you are working on a project at school or work and want to continue at home without losing the data or the place you were at in your project. This is done by using a 3½ floppy disk to transport the system. The following procedure is used to transfer CRAM to another system:

1. Start the system without CRAM running
2. Run the installation program using the drive switches to indicate the new drive.

ex. INSTALL -i -L B

CRAM will be installed on the B drive

3. CRAM will now save memory to drive B.

4. After turning off the computer remove the floppy disk.

5. Insert the floppy disk in the drive of the other system.

6. Start the system without CRAM loaded.

7. At the DOS prompt start CRAM by entering:

   CRAM -u -L B or A {A or B for the floppy drive with CRAM }

8. After the restoration you can continue where you were at the office.

Removing CRAM From the Hard Disk and System

CRAM was designed to use a hard disk for normal operations. Another feature of CRAM is transferring an entire operation from one computer to another or one disk to another. To do this the system must have a 3½ floppy disk drive. To transfer CRAM use the following procedure:

1. Start the system without CRAM running.

2. At the DOS prompt enter CRAM /t *source-drive destination drive*

   ex. CRAM /t C B

3. CRAM will request the original disk

4. After transferring CRAM to the original disk CRAM will be removed from the system and disk.

APPENDIX C. CRAM SOURCE CODE

TSR.H TSR macros, variables and functions declaration.

```
/* TSR Prototype file and common variables */

#define INTERRUPT void interrupt far

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


/* Prototypes for functions in CRAMUTIL.C */
int        DosBusy(void);
int        Int28DosBusy(void);
void       InitInDos(void);
unsigned   GetPSP(void);
void       SetPSP(unsigned segPSP);
int        InitDosSwap(void);

/* Prototypes for functions in TSRUTIL.ASM */
int  far  deinstall(void);
void far  idle_int_chain(void);
void far  init_intr(void);

void interrupt far   new_int10(void);
void interrupt far   new_int13(void);
void interrupt far   new_int25(void);
void interrupt far   new_int26(void);

void far  timer_int_chain(void);

/* Prototypes for functions in STACK.ASM */
void far  set_stack(void);
void far  restore_stack(void);

/* Common variables */
extern char far *  indos_ptr;
extern char far *  crit_err_ptr;
```

CRAM.H Macros, definitions and functions declarations not found in TSR.H.

```
/****            CRAM.H
CRAM declarations
Copyright for Iowa State University by Renford A. B. Brevett.

****/
/* Declaration of constants to be used by Biosdisk function.          */

#define  RESET   0x00        /* Resets disk system                 */
#define  STATUS  0x01     /* Return the status of last disk operation */
#define  READ    0x02     /* Reads one or more disk sectors        */
#define  WRITE   0x03     /* Writes one or more disk sectors       */
#define  VERIFY  0x04     /* Verifies one or more disk sectors     */
#define  FORMAT  0x05      /* Formats a track                      */
#define  MAX_SEG  0x10     /* Maximnm segments supported by CRAM amount*/
                              /* in paragraphs of 64K (0x10 * 64k = 1Meg  */
#define  MAXPART    4     /* Maximun partition on a Fix Disk         */

#define  D_UNUSED   0  /* System ID Unused Partition */
#define  D_FAT12    1  /*  12 Bit fats              */
#define  D_FAT16    4  /*  16 Bit fats        */
#define  D_EXTPAR   5  /*  Extended DOS Partition   */
#define  D_40PAR    6  /*  4.0 >32Mb Partition      */

#define  R_O        0x01 /* Read Only File Attribute */
#define  HIDDEN     0x02 /* Hidden File Attribute    */
#define  SYSTEM     0x04 /* System File Attribute    */
#define  VOLUME     0x08 /* Volume Entry Attribute   */
#define  SUBDIR     0x10 /* Subdirectory Entry Attribute */
#define  ARCHIVE    0x20 /* Archive Entry Attribute  */
#define  UNUSED     0x00 /* Directory Entry never used */
#define  ERASED     0xE5 /* Erased Directory Entry   */
#define  DIRRECTORY 0x2E /* Directory Entry Attribute */

#define SIG   ("Copyright(c) 1991 Renford A. B. Brevett at Iowa State University ")
#define LINEFEED        0x0D
#define SPACE           " "
#define COLON            ":"
#define RDGT            " >>"
#define LDGT            "<< "
#define PARAGRAPHS(x)    ((FP_OFF(x) + 15)  >> 4)

#define KEYBOARD_PORT     0x60 /* KEYBOARD Data Port */
#define KEYBRD_SHIFTSTATUS 0x02 /* CHECK KEYBOARD SHIFT STATUS */
#define KEYBRD_READY       0x01 /* Check for Character in KEYBOARD buffer */
#define KEYBRD_READ        0x00 /* Read Character from KEYBOARD buffer    */
#define RIGHT_SHIFT       0x01
#define LEFT_SHIFT        0x02
#define CTRL_KEY          0x04
#define ALT_KEY          0x08
#define FIVEKEY          0x4C
#define DELKEY           0x53
#define INSTALL_CHECK      0x00
#define INSTALLED         0xFF
```

```
 #define DEINSTALL          0x01
 #define KEY_MAX            0x14  /* Number of keystrokes before next save */

 #define GET_PSP_DOS2       0x51
 #define GET_PSP_DOS3       0x62
 #define SET_PSP            0x50
 #define PSP_TERMINATE      0x0A  /* Termination addr. in our PSP */
 #define PSP_PARENT_PSP     0x16  /* Parent's PSP from our PSP */
 #define PSP_ENV_ADDR       0x2C   /* environment address from PSP */

 #define STDERR             fileno(stdout)
 #define MAX_WID            12
 #define GET_DOSSWAP3       0x5d06
 #define GET_DOSSWAP4       0x5d0b
 #define SWAP_LIST_LIMIT    20

 #define GET_EXTERR         0x59
 #define SET_EXTERR         0x5d0a
 #define GET_INDOS          0x34
 #define GET_CRIT_ERR       0x5D06

 #define SET_DTA            0x1A  /* SET Disk Transfer Address */
 #define GET_DTA            0x2F  /* GET Disk Transfer Address */
 #define DOS_EXIT           0x4C  /* DOS terminate (exit) */

 #ifdef __TURBOC__
 #define GETVECT(x)          getvect(x)
 #else
 #define GETVECT(x)          _dos_getvect(x)
 #endif

typedef void far *FP;


#define MK_S(addr)          ((FP) ( FP_SEG(addr< <8)))
#define MK_O(addr1) ((FP)(    ((DWORD)(addr1)-((DWORD)(addr1)< <4)) ))

#define MCB_FM_SEG(seg)       ((seg) - 1)
#define IS_PSP(mcb)           (FP_SEG(mcb) + 1 = = (mcb)->owner)
#define ENV_FM_PSP(psp_seg)   (*((WORD far *) MK_FP(psp_seg, 0x2c)))
#define TERM_FM_PSP(psp_seg)  MK_FP(psp_seg, 0x0A)

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

/* Some Common Variables  */

extern int      DEBUG;

/*  Declaration of TYPES used in CRAM  */

typedef void (interrupt far *INTVECT)();
typedef unsigned char BYTE;  /* Redefine unsigned char as BYTE          */
typedef unsigned      WORD;  /* Redefine unsigned as WORD               */
typedef unsigned long DWORD;  /* Redefine unsigned long as Double WORD    */
typedef enum { FALSE, TRUE } BOOL;

typedef struct {
   BYTE type;          /* 'M'=in chain; 'Z'=at end */
```

```
    WORD owner;         /* PSP of the owner */
    WORD size;          /* in 16-byte paragraphs */
    BYTE unused[3];
    BYTE dos4[8];
    } MCB;

 typedef struct MCB_TBL
  {
    WORD  addr;
    MCB   mcb;
    DWORD term_addr;
  }MCB_TBL;

 /* Definitions and functions for video control  */

 typedef struct SCR_LOC
    {
          char    s_char, s_attr;
    } SCR_LOC;                         /* One screen location    */

 typedef SCR_LOC        SCRLINE [80];    /* One screen line        */

 struct bits
  {
   unsigned bit0 : 1;
   unsigned bit1 : 1;
   unsigned bit2 : 1;
   unsigned bit3 : 1;
   unsigned bit4 : 1;
   unsigned bit5 : 1;
   unsigned bit6 : 1;
   unsigned bit7 : 1;
  };

 struct bits_16
  {
    struct bits lobits;
    struct bits hibits;
  };


 struct CRAM_ver
   {
    unsigned  major : 3;
    unsigned  minor : 3;
    unsigned  beta  : 1;
    unsigned  test  : 1;
   };

/*  Declaration of STRUCTURES used in CRAM   */

struct DPT                /* A structure to hold information for the  */
         {                 /* Disk Parameter Table                     */
           BYTE   bootid;
           BYTE   starthead;
           BYTE   startsec;
           BYTE   starttrack;
           BYTE   sysid;
           BYTE   endhead;
          BYTE    endsec;
          WORD    endtrack;
          DWORD   firstsec_in_part;
          DWORD   numsecs_in_part;
          };

struct BPB              /* A structure to hold information for the  */
          {             /* Bios Parameter Block                     */
          unsigned jmpcode;
          BYTE    jmpaddr;
          BYTE     sysid[8];
          WORD    bps;
          BYTE     spc;
          WORD    ressec;
          BYTE    nfats;
          WORD    nroot_dir;
          WORD    ndsksect;
          BYTE    fmtid;
          WORD    nspf;
          WORD    nspt;
          WORD    nsides;
          WORD    nres_sec;
          DWORD    vol_s_32;
          WORD    endbp;
          };

struct HST                  /* A Structure to hold information for the  */
     {                      /* Head, Sector and Track when converting   */
          unsigned char LETTER;  /* from absolute DOS sector                */
          unsigned char DRIVE_NUM;
          unsigned HEAD;
          unsigned SECTOR;
          long     TRACK;
     };

struct  ExtErr
{
  unsigned int errax;
  unsigned int errbx;
  unsigned int errcx;
};


struct dir_entry
 {
  BYTE    f_name[8];
  BYTE    f_ext[3];
  BYTE    f_attrib;
  BYTE    f_reserved[10];
  WORD     f_time;
  WORD     f_date;
  WORD     f_start_cluster;
  DWORD    f_size;
 };

struct CRAM_HEADER
  {
    DWORD ID;
    WORD   CS, DS, ES, SS;
```

```
        WORD  AX, BX, CX, DX;
        WORD  BP, DI, SI, SP;
        WORD  IP, PSP, FLAGS;
        WORD  date;
        WORD  time;
        BYTE  f_access[64];
        WORD  checksum;
        DWORD clusters;
        DWORD start_cluster;
        BYTE  password[16];
        struct ExtErr  ErrInfo;
        WORD  start_addr;
        DWORD f_size;
        WORD  int_checksum;
        WORD  offs_filler;
        WORD  sec_seg1_end;
        WORD  video_sec;
        WORD  dsa_sec;
        WORD  mcb_sec;
        WORD  stack_sec;
        WORD  int_sec;
        WORD  mem_sec;
        WORD  curr_mem_sec;
        WORD  data_sec;
        int   dsa_size;
        WORD  dta_seg;
        WORD  dta_off;
        DWORD terminate_addr;
        WORD  cram_dta_seg;
        WORD  cram_dta_off;
        WORD  cram_psp;
        BYTE  save_seg_flag[MAX_SEG];
        WORD  seg_checksum[MAX_SEG];
        WORD  sector_in_cram[MAX_SEG];
        WORD  size_of_seg[MAX_SEG];
        BYTE  reserve_for_seg[MAX_SEG];
  };

struct PSP_INFO
{
  WORD  resev1;
  WORD  sys_mem;          /* in 16 bytes blocks */
  WORD  resev2;
  BYTE  DOScall[5];
  WORD  bytes_in_seg;
  DWORD term_addr;       /*  IP, CS       */
  DWORD cntrl_c_addr;    /*  IP, CS       */
  DWORD hard_err_addr;   /*  IP, CS       */
  BYTE  resev3[22];
  WORD  env_addr;
  BYTE  resev4[34];
  BYTE  DOScall2[12];
  BYTE  FCB1[16];
  BYTE  FCB2[16];
  DWORD resev5;
  BYTE  DTA[128];
};

struct  F_HEADER
```

```
  {
    WORD    filetype;
    WORD    bytes_in_last_page;
    WORD    f_size;              /*  in 512 bytes page   */
    WORD    relo_entries;
    WORD    h_size;
    WORD    minalloc;             /* minimum memory required */
    WORD    maxalloc;             /* maximum memory required */
    WORD    SS;                  /* SS relative to start of program */
    WORD    SP;
    WORD    checksum;
    WORD    IP;
    WORD    CS;
    WORD    off_2_RT;
    WORD    ovrl_link;
  };

 struct partition
  {
   BYTE    code[446];
   struct  DPT  DPT_tbl[MAXPART];
   WORD     DOSsig;
  };

 union FATS
 {
 /* use as FATS.fat_16.f16  = ?
    or    FATS.fat_12_lo.fat12 =  ?
    or    FATS.fat_12_hi.fat12 =  ?
 */
   struct
     {
          unsigned int fat16 : 16;
     } fat_16;
   struct
     {
          unsigned int fat12 : 12;
          unsigned int xxx    : 4;
     } fat_12_lo;
   struct
     {
          unsigned int  xxx : 4;
          unsigned int  fat12:12;
     } fat_12_hi;
};

struct free_fat
 {
   DWORD  fbegin;
   DWORD  fend;
   DWORD  fsize;
 };

/* Prototypes in GEN_UTIL.c   */

extern unsigned put_chr(int c);
extern unsigned put_str(char far *s);
extern unsigned put_num(unsigned long u, unsigned wid, unsigned radix);
#define         put_hex(u)       put_num(u, 4, 16)
```

```
#define      put_bit(u)     put_num(u, 2, 16)
#define      put_long(ul)   put_num(ul, 9, 10)
#define      putstr(s)    { put_str(s); put_str("\r\n"); }
extern unsigned fstrlen(const char far *s);

extern int       LPT(char *s);
extern void      gotoXY(int x, int y);
extern void      curr_cursor(int *x, int *y);
extern void      set_cursor_type(int t);
extern void      clear_screen(char ch);
extern void      clear_win(int x1, int y1, int x2, int y2, char ch, BYTE attrib);
extern int       vmode();
extern int       scroll_lock();
extern int       get_char();
extern void      clrEol(void);
extern void      (*helpfunc)(void);

/* Prototypes in CRAMTOOLS.c    */

extern struct    CRAM_ver c_ver;
extern int       getfat_info(struct free_fat *freefat, WORD clust_req);
extern long      mem_checksum(WORD seg_start, WORD offs_start, DWORD mem_size);
extern int       get_fatfree(struct free_fat *freefat, WORD clust_req);
extern int       put_fat(struct free_fat freefat, DWORD data_sec);
extern int       put_f_name(struct free_fat f_free, DWORD data_sec);
extern long      get_f_name(void);
extern char      pause(int err_num);
extern int       show_disk(void);
extern long      get_data_sec(void);
extern int       ckeckdisk(void);

/* Prototypes in CRAMMEM.c  */

extern           mcb_chk(MCB far *mcb);
extern MCB far   *get_mcb (void);
extern char far *progname_fm_psp(unsigned psp);
extern void      display_progname(MCB far *mcb);
extern MCB far   *get_cmd_mcb(MCB far *mcb);
extern MCB far   *IS_CRAM(MCB far *mcb);
extern BOOL      belongs(void far *vec, unsigned start, unsigned size);
extern void      display(MCB far *mcb);
extern char far *env(MCB far *mcb);
extern void      display_cmdline(MCB far *mcb);
extern void      display_vectors(MCB far *mcb);
extern WORD      low_mem(MCB far *mcb);
extern int       get_mcb_chain(MCB far * mcb, BYTE  *buffer);
extern int       set_mcb_chain(BYTE  *buffer, WORD psp);
extern           showMCB(char  *buffer);

/* Prototypes  in CRAMUTIL.c*/

extern int       color_adpt(void);
extern int       SaveDosSwap(void);
extern void      RestoreDosSwap(void);
extern int       save_DSA(void);
extern int       reset_DSA(void);
extern int       get_drive_info(char *drv);
extern void      GetExtErr(struct ExtErr *ErrInfo);
extern void      SetExtErr(struct ExtErr *ErrInfo);
extern DWORD    sector2cluster(DWORD sector);
extern DWORD    cluster2sector(DWORD cluster);

/* Prototypes in CRAMINT.C */

extern void     tsr_exit(void);
extern void     usage(char *);
extern int      UnlinkVect(int Vect, INTVECT NewInt, INTVECT OldInt);
extern void     parse_cmd_line(int argc, char *argv[]);

/* Prototypes in DISK.C  */

extern int      stohst(int drive, long track, int head, long *DOSsec,
                        struct BPB *bpb, struct DPT *dpt, struct HST  *hst);
extern int      getDPT(unsigned drive, struct DPT  *dpt);
extern int      getBPB(int drive, int head, long track, int sector, struct BPB *bpb);
extern int      getsector(int drive, int nsects, long *DOSsec, struct BPB *bpb,
                        struct DPT *dpt,  BYTE *buffer);
extern int      putsector(int drive, int nsects, long *DOSsec, struct BPB *bpb,
                        struct DPT *dpt,  BYTE far *buffer);
extern int      fmttrack(int drive, int nsects, long track, int head, long *DOSsec,
                        struct BPB *bpb, struct DPT *dpt);
extern int      printBPB(struct BPB bpb);
extern int      printDPT(struct DPT  dpt);  /* Prints the Disk Parameter Table  */
extern int      printsector(BYTE *buffer);
```

DISK.C Functions used most disk I/O.

```
/****          DISK.C
File of declaration and some general disk functions
Copyright for Iowa State University by Renford A. B. Brevett.

****/

#include <bios.h>
#include <stdio.h>
#include <dos.h>
#include <dir.h>
#include <stdlib.h>
#include <mem.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include "cram.h"


/*==============================================================
=====*/
int printDPT(struct DPT  dpt)  /* Prints the Disk Parameter Table  */
 {
   printf("\r\n\tBoot ID             %Xh"
          "\r\n\tStart Head          %d"
          "\r\n\tStart sector        %d"
          "\r\n\tStart Track         %d"
          "\r\n\tFAT System ID       %d"
          "\r\n\tEnd Head            %d"
          "\r\n\tEnd Sector          %d"
          "\r\n\tEnd Track           %d"
          "\r\n\tFirst Partition sec.  %ld"
          "\r\n\tSectors in Partition  %ld\r\n",
   dpt.bootid, dpt.starthead, dpt.startsec, dpt.starttrack,
   dpt.sysid, dpt.endhead, dpt.endsec, dpt.endtrack,
   dpt.firstsec_in_part, dpt.numsecs_in_part);
   return(0);
 }

int getDPT(unsigned drive, struct DPT  *dpt)

/*
 Gets the Disk Parameter Table for the drive given by the parameter
 drive.  The parameter dpt is a pointer to the structure that hold
 the result.
*/
 {
   unsigned int  offset = 0x01BE;
   int result;
   BYTE  s, t, r;
   BYTE buffer[512];

   result = biosdisk(2,drive,0,0,1,1,buffer);
   if (result != 0)
          {
```

```
                            result = biosdisk(2,drive,0,0,1,1,buffer);
                            if (result != 0)
                             {
                              printf("Error %d reading Disk Parameter Table (DPT)", result);
                              return(result);
                             }
                  }
     memcpy(&dpt->bootid,          &buffer[offset++], 1);
     memcpy(&dpt->starthead,       &buffer[offset++], 1);
     memcpy(&dpt->startsec,        &buffer[offset++], 1);
     memcpy(&dpt->starttrack,      &buffer[offset++], 1);
     memcpy(&dpt->sysid,           &buffer[offset++], 1);
     memcpy(&dpt->endhead,         &buffer[offset++], 1);
     memcpy(&dpt->endsec,          &buffer[offset++], 1);
     memcpy(&dpt->endtrack,        &buffer[offset++], 1);
     memcpy(&dpt->firstsec_in_part, &buffer[offset],   4);
     memcpy(&dpt->numsecs_in_part,  &buffer[offset+4], 4);
     s = dpt->endsec;
     dpt->endsec =(s & 0x3F);
     t = dpt->endtrack;
     r = (s & 0xC0);
     dpt->endtrack = (t | (r << 2));
     return(result);
  }

int getBPB(int drive, int head, long track, int sector, struct BPB *bpb)
/*
  Gets the Bios Parameter Block of drive. The BPB is found on the drive
  using the head, track and sector passed to the function. A pointer
  to the BPB structure bpb hold the result.
*/
   {
    int result;
    BYTE buffer[512];

    result = biosdisk(2,drive,head,track,sector,1,buffer);
    if (result != 0)
      {
             result = biosdisk(2,drive,head,track,sector,1,buffer);
            if (result != 0)
               {
                printf("\r\n ERROR %d Reading Disk Parameter Block (BPB)", result);
                printf("\r\n drive:%-2.2d head:%d sector:%d \r\n",
                            drive, head, sector);
                return(result);
               }
      }
   memcpy(&bpb->jmpcode,   &buffer[0],  2);
   memcpy(&bpb->jmpaddr,   &buffer[2],  1);
   memcpy(&bpb->sysid,     &buffer[3],  8);
   memcpy(&bpb->bps,            &buffer[11], 2);
   memcpy(&bpb->spc,       &buffer[13], 1);
   memcpy(&bpb->ressec,    &buffer[14], 2);
   memcpy(&bpb->nfats,     &buffer[16], 1);
   memcpy(&bpb->nroot_dir, &buffer[17], 2);
   memcpy(&bpb->ndsksect,  &buffer[19], 2);
   memcpy(&bpb->fmtid,     &buffer[21], 1);
   memcpy(&bpb->nspf,      &buffer[22], 2);
   memcpy(&bpb->nspt,      &buffer[24], 2);
```

```
    memcpy(&bpb->nsides,    &buffer[26], 2);
    memcpy(&bpb->nres_sec,  &buffer[28], 4);
    memcpy(&bpb->vol_s_32,  &buffer[32], 4);
    memcpy(&bpb->endbp, &buffer[510], 2);
    return(result);
   }

 int printBPB(struct BPB bpb)
 /*
  Prints the Bios Parameter Block.  The pointer bpb must point to a structure
  after a previous getBPB.
 */
 {
   printf("\r\n=============== Boot Record =============================\r\n");

   printf("\r\n\t Jump Code                  %4X \r\n\t"
                         " Jump Address              %2X \r\n\t"
                         " System ID                 %.*s \r\n\t"
                         " Bytes per Sector          %d \r\n\t"
                         " Sectors per Cluster       %d \r\n\t"
                         " Reserve Sectors           %d \r\n\t"
                         " Number of Fats             %d \r\n\t"
                         " Number of Root Dir.         %d \r\n\t"
                         " Number of Disk Sectors      %ld \r\n\t"
                         " Media Format ID.          %2X \r\n\t"
                         " Number of Sectors per FAT   %d \r\n\t"
                         " Number of Sectors per Track %d \r\n\t"
                         " Number of Sides            %d \r\n\t"
                         " Number of Reserve Sectors   %d \r\n\t"
                         " Voulme 32 Bit value        %lu \r\n\t"
                         "\r\n=============== Word at End of Boot Record ===== %4X
====================\r\n",
            bpb.jmpcode,bpb.jmpaddr,8,bpb.sysid,bpb.bps,bpb.spc,bpb.ressec,bpb.nfats,bpb.nroot_dir,
            (long)bpb.ndsksect,bpb.fmtid,bpb.nspf,bpb.nspt,bpb.nsides,bpb.nres_sec,(DWORD)bpb.vol_s_32,bpb.endbp);

  return(0);
}

int printsector(BYTE *buffer)
/*
 Prints the sector pointed to by buffer.  Buffer must be alloted enough
 memory and passed after a previous call to getsector.
*/
 {
  int i=0, j=1, x = 52;
  printf("\r\n");
  gotoxy(1, wherey());
           for(i=0; i<512; i++)
            {
             if (j >= 48)
              {
                    printf("\r\n");
                    j = 1;
                    x = 52;
                    gotoxy(1,wherey());
              }
             gotoxy(j+2,wherey());printf("%-2.2X ",buffer[i]);
             gotoxy(x,wherey());
             if (buffer[i] == 0x07 || buffer[i] == 0x0A || buffer[i] == 0x00)
```

```
                    {
                     putc('.', stdout);
                    }
                else
                    putc(buffer[i], stdout);
          j+ =3;
               x+ +;
              }
   return(0);
  }

int stohst(int drive, long track, int head, long *DOSsec,
            struct BPB *bpb, struct DPT *dpt, struct HST  *hst)

/*
  stohst convert a absolute DOS sector to its head, track, sector number
  needed for bios calls.  To reverse the process DOSsec must first
  be assigned the value -1 before a call to stohst.
*/
 {
 int       result=0;
 ldiv_t    head_cal;
 ldiv_t    track_cal;
 ldiv_t    sector_cal;
 int       sector = 1;
 int       drv;
 long       maxsides   = 2L;
 int       skiptrack;
 long       spt;

 if (drive > 0x79)
  {
   drv = ((drive - 0x80) + 2);
   maxsides   = dpt->endhead;
   skiptrack = bpb->nspt;
  }
 else
  {
   drv = drive;
   maxsides   = 1L;
   skiptrack = 0;
  }
  spt           = bpb->nspt;
  if(*DOSsec = = -1)
   {
         *DOSsec = (sector-1) + (head * spt)
                         + (track * spt * (maxsides+1));
   }
  else
   {
         head_cal       = ldiv(*DOSsec+skiptrack, (long)spt);
         track_cal      = ldiv(*DOSsec+skiptrack, (long)(spt * (maxsides+1)));
         track          = (long)track_cal.quot;
         head           = (int)(head_cal.quot % (maxsides + 1));
         sector_cal     = ldiv(*DOSsec+skiptrack, (long) spt);
         sector         = 1 + (int) sector_cal.rem;
   }

 hst->HEAD      = head;
```

```
   hst->SECTOR    = sector;
   hst->TRACK     = track;
   hst->DRIVE_NUM = drive;
   hst->LETTER    = 'A'+drv;
   return(result);
  }


int getsector(int drive, int nsects, long *DOSsec, struct BPB *bpb,
                         struct DPT *dpt, BYTE *buffer)
/*
   Gets an absolute sector on drive and buffer will point to the content
   of the sector.
*/
 {
   int result;

   static struct HST hst;
   stohst(drive, hst.TRACK, hst.HEAD, (long *)DOSsec, bpb, dpt, &hst);
   result = biosdisk(2, hst.DRIVE_NUM, hst.HEAD, hst.TRACK, hst.SECTOR,
                nsects,buffer);
   if (result != 0)
     {
           result = biosdisk(2, hst.DRIVE_NUM, hst.HEAD, hst.TRACK, hst.SECTOR,
                        nsects,buffer);
           if (result != 0)
            {
             printf("\r\n Error %d Reading Sector %ld", result, *DOSsec);
             printf("\r\nDrive %c  T:%ld S:%d H:%d [Rel. Sector:%ld]\r\n",
                        hst.LETTER, hst.TRACK, hst.SECTOR,hst.HEAD, *DOSsec);
             return(result);
            }
     }
  return(result);
}


int putsector(int drive, int nsects, long *DOSsec, struct BPB *bpb,
    struct DPT *dpt, BYTE far *buffer)
/*
   Writes nsects sectors to drive starting at absolute sector DOSsec.
   The pointer buffer must point to the information to write.
*/
{
  int result;
  static struct HST hst;
  char *pbuf;

  stohst(drive, hst.TRACK, hst.HEAD, DOSsec, bpb, dpt, &hst);
  pbuf = (char *)buffer;
  result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK, hst.SECTOR,
              nsects, &pbuf[0]);
  if (result != 0)
    {
          printf("\r\n Error %d Writing Sector %ld", result, *DOSsec);
          printf("\r\nDrive %c  T:%ld S:%d H:%d [Rel. Sector:%ld]\r\n",
              hst.LETTER, hst.TRACK, hst.SECTOR,hst.HEAD, *DOSsec);
          return(result);
    }
   return(result);
  }

int fmttrack(int drive, int nsects, long track, int head, long *DOSsec,
      struct BPB *bpb, struct DPT *dpt)
/*
  Format a track on drive.
*/
{
  int result, i;
  BYTE  *buf;
  int  sizecode;

  struct HST hst;

   stohst(drive, track, head, DOSsec, bpb, dpt, &hst);
   buf = malloc (bpb->nspt*4);
   gotoxy(1, wherey());
   printf("Formating Disk ....Drive:<%c:> with %d Sectors Per Track ",
    hst.LETTER, bpb->nspt);
    switch (bpb->bps)
     {
           case 128 : sizecode = 0;break;
           case 256 : sizecode = 1;break;
           case 512 : sizecode = 2;break;
           case 1024: sizecode = 3;break;
           case 2048: sizecode = 4;break;
     }
            for (i = 0; i<9; i++)
            {
                      buf[i * 4]      = hst.TRACK;
                      buf[i * 4 +1]   = hst.HEAD;
                      buf[i * 4 +2] = i + 1;
                      buf[i * 4 +3] = sizecode;
            }

  result = biosdisk(FORMAT, hst.DRIVE_NUM, hst.HEAD, hst.TRACK, hst.SECTOR,
               nsects, &buf);
               gotoxy(wherex(), wherey());
               printf(" H:%d T:%ld S:%d", head, track, hst.SECTOR);
  if (result != 0)
   {
        printf(" \r\nError %d Formating Sector %ld", result, *DOSsec);
        printf("\r\nDrive %c \r\nT:%ld S:%d H:%d [Rel. Sector:%ld]\r\n",
            hstLETTER, track, hst.SECTOR, head, *DOSsec);
        goto end;
   }
 end:
 free(buf);
 return(result);
}
```

CRAMUTIL.C Most functions used to manipulate CRAM.

```
 /*                  CRAMUTIL.C                  */
/***
 Utilities used with CRAM to control interrupts and memory
 Copyright by Renford A. B. Brevett at Iowa State University.


***/
#include <stdlib.h>
#include <dos.h>
#include <stdio.h>
#include <mem.h>
#include <string.h>
#include <stdarg.h>
#include <bios.h>
#include <io.h>
#include "tsr.h"
#include "cram.h"

extern SCRLINE     far *scr;
extern int         drive;
extern long        DOSsec;
extern struct DPT   dpt;
extern struct BPB   bpb;
extern struct HST   hst;
extern long         result;
extern long         maxsectors;
extern long         maxsides;
extern long         maxtracks;
extern int          skiptrack;
extern struct       CRAM_HEADER   c_header;
extern BYTE           buffer[];
extern BYTE           swap_save_buff[];
extern DWORD           data_sec, reserved_sec;

struct  swap_list   /* format of DOS 4+ SDA list */
{
   void   far*   swap_ptr;
          int    swap_size;
};
/* variables for 3.x swap work */

char  far      * swap_ptr;  /* pointer to dos swap area */
char  far      * swap_save; /* pointer to our local save area */
static  int     swap_size_indos;
static  int     swap_size_always;
static  int     size;

/* variables for 4.x swap work */

static int        swap_count;      /* count of swappable areas */
static struct     swap_list swp_list[SWAP_LIST_LIMIT]; /*list of swap areas*/
static char far    *swp_save[SWAP_LIST_LIMIT];  /* out save area */
static int        swp_flag[SWAP_LIST_LIMIT];  /* flags if has been swapped */
static int        dos_level;  /* for level dependent code */
int               dos_critical = 0;     /* in critical section, can't swap */
char far          *indos_ptr=0;
char far          *crit_err_ptr=0;
extern BYTE far   *diskette;
static union  REGS  regs;
static union  REGS  rg;

/* Prototypes */

void              get_time_date(void);
DWORD                 cluster2sector(DWORD cluster);
DWORD                 sector2cluster(DWORD sector);
int               SaveDosSwap(void);
void              RestoreDosSwap(void);
int               save_DSA(void);
int               reset_DSA(void);
void              GetExtErr(struct ExtErr *ErrInfo);
void              SetExtErr(struct ExtErr *ErrInfo);
int               get_drive_info(char *drv);
/*=================================================================*/
/*     Functions to manage DOS flags   */

/*****
Function: Init InDos Pointers
Initialize pointers to InDos Flags
*****/
void InitInDos(void)
{
   union  REGS regs;
   struct SREGS segregs;

   regs.h.ah = GET_INDOS;
   intdosx(&regs,&regs,&segregs);
   /* pointer to flag is returned in ES:BX */
   FP_SEG(indos_ptr) = segregs.es;
   FP_OFF(indos_ptr) = regs.x.bx;

   if (_osmajor < 3)  /* flag is one byte after InDos */
      crit_err_ptr = indos_ptr + 1;
   else if (_osmajor= =3 && _osminor = = 0) /* flag is one byte before */
      crit_err_ptr = indos_ptr - 1;
   else
   {
      regs.x.ax = GET_CRIT_ERR;
      intdosx(&regs,&regs,&segregs);
      /* pointer to flag is returned in DS:SI */
      FP_SEG(crit_err_ptr) = segregs.ds;
      FP_OFF(crit_err_ptr) = regs.x.si;
   }
}

/*****
Function: DosBusy
This function will non-zero if DOS is busy
*****/
int DosBusy(void)
{
   if (indos_ptr && crit_err_ptr)
```

```
      return (*crit_err_ptr || *indos_ptr);
   else
      return 0xFFFF;  /* return dos busy if flags are not set */
}

/*****
Function: Int28DosBusy
This function will return non-zero if the InDos flag is > 1 or
the critical error flag is non zero. To be used inside of an
INT 28 loop. Note that inside INT 28, InDOS == 1 is normal, and
indicates DOS is *not* busy; InDOS > 1 inside INT 28 means it is.
*****/
int Int28DosBusy(void)
{
   if (indos_ptr && crit_err_ptr)
     return (*crit_err_ptr || (*indos_ptr > 1));
   else
     return 0xFFFF;  /* return dos busy if flags are not set */
}


/*  Functions to manage DOS swap areas       */



/*****
Function: InitDosSwap
Initialize pointers and sizes of DOS swap area. Return zero if success
*****/
int InitDosSwap(void)
{
   union  REGS regs;
   struct SREGS segregs;

   if ((_osmajor == 3) && (_osminor >= 10))
           dos_level = 3;
   else if (_osmajor >= 4)
       dos_level = 4;
   else
       dos_level = 0;
   if (dos_level == 3)    /* use 215D06 */
   {
       regs.x.ax = GET_DOSSWAP3;
       intdosx(&regs,&regs,&segregs);
       /* pointer to swap area is returned in DS:SI */
       FP_SEG(swap_ptr) = segregs.ds;
       FP_OFF(swap_ptr) = regs.x.si;

       swap_size_indos = regs.x.cx;
       swap_size_always = regs.x.dx;

       size = 0;  /* initialize for later */
            return((swap_save = (char far *)swap_save_buf) == 0);
   }
  else if (dos_level == 4)  /* use 5d0b */
  {
       struct swap_list far *ptr;
       int far *iptr;
       int i;
      regs.x.ax = GET_DOSSWAP4;
      intdosx(&regs,&regs,&segregs);
      /* pointer to swap list is returned in DS:SI */
      FP_SEG(iptr) = segregs.ds;
      FP_OFF(iptr) = regs.x.si;
      swap_count = *iptr;               /* get size of list */
      iptr++;
      ptr = (struct swap_list far *) iptr;  /* create point to list */

      if (swap_count > SWAP_LIST_LIMIT)    /* too many data areas */
         return 2;

      /* get pointers and sizes of data areas */
      for (i = 0; i < swap_count; i++)
      {
         swp_list[i].swap_ptr = ptr->swap_ptr;
         swp_list[i].swap_size= ptr->swap_size;
             if (! (swp_save[i] = malloc(swp_list[i].swap_size & 0x7fff)))
/*               if (! (swp_save[i] = (char far *)&buffer[(swp_list[i].swap_size & 0x7fff)]))*/
            return 3;   /* out of memory */
         swp_flag[i] = 0;
         ptr++;   /* point to next entry in the list */
      }
          return 10;
   }
   else
          return 9;   /* 9 = unsupported DOS */
}

/*****
Function: SaveDosSwap
This function will save the dos swap area to a local buffer
It returns zero on success, non-zero meaning can't swap
*****/
int SaveDosSwap(void)
{
   if (dos_level == 3)
   {
      if (swap_ptr && !dos_critical)
      {
         /* if INDOS flag is zero, use smaller swap size */
         size = (*indos_ptr) ? swap_size_indos : swap_size_always;

         movedata(FP_SEG(swap_ptr), FP_OFF(swap_ptr),
               FP_SEG(swap_save), FP_OFF(swap_save),
               size);
      }
      else     /* can't swap it */
         return 1;
   }
   else if (dos_level == 4)
   {
      /* loop through pointer list and swap appropriate items */
      int i;
      for (i = 0; i < swap_count; i++)
      {
         if (swp_list[i].swap_size & 0x8000)  /* swap always */
         {
            movedata(FP_SEG(swp_list[i].swap_ptr),
                FP_OFF(swp_list[i].swap_ptr),
                FP_SEG(swp_save[i]),
                FP_OFF(swp_save[i]),
                swp_list[i].swap_size & 0x7fff);
        }
        else if (*indos_ptr)    /* swap only if dos busy */
        {
           movedata(FP_SEG(swp_list[i].swap_ptr),
                FP_OFF(swp_list[i].swap_ptr),
                FP_SEG(swp_save[i]),
                FP_OFF(swp_save[i]),
                swp_list[i].swap_size);
        }
      }
   }
   else
     return 1;

   return 0;
}

/*****
Function: RestoreDosSwap
This function will restore a previously swapped dos data area
*****/
void RestoreDosSwap(void)
{
   if (dos_level = = 3)
   {
    int i;
        /* make sure its already saved and we have a good ptr */
        if (size && swap_ptr)
        {
           disable();
           movedata(FP_SEG(swap_save), FP_OFF(swap_save),
                    FP_SEG(swap_ptr),  FP_OFF(swap_ptr), size);
           enable();
           size = 0;
        }
   }
   else if (dos_level = = 4)
   {
        int i;
        scr[0] [0].s_char = 0x0034;
        for (i = 0; i < swap_count; i++)
        {
           movedata(FP_SEG(swp_save[i]),
                    FP_OFF(swp_save[i]),
                    FP_SEG(swp_list[i].swap_ptr),
                    FP_OFF(swp_list[i].swap_ptr),
                    swp_list[i].swap_size);
          swp_flag[i] = 0;  /* clear flag */
        }
   }
}

/*     extended error saving and restoring */

int save_DSA(void)
```

```
 {
  scr[0] [0].s_char = 0x00E4;
  scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
  DOSsec = c_header.dsa_sec;
  if((SaveDosSwap()) == 0)
   {
        c_header.dsa_size = size;
        if (dos_level == 3)
         {
             /* make sure its already saved and we have a good ptr */
           if (size && swap_ptr)
            {
                 movedata(FP_SEG(swap_save), FP_OFF(swap_save),
                          FP_SEG(buffer), FP_OFF(buffer), size);
                 if(DOSsec <= reserved_sec) return (-1);
                 stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
                 if ((result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
                      hst.SECTOR, 4, buffer)) != 0)
                   return(result);
                 size = 0;
            }
         }
        else if (dos_level == 4)
         {
           int i;
           for (i = 0; i < swap_count; i++)
            {
                 movedata(FP_SEG(swp_save[i]),
                          FP_OFF(swp_save[i]),
                          0xB800,
                          i * 0x100,
                          swp_list[i].swap_size);
                          swp_flag[i] = 0;  /* clear flag */
            }
         }

   }
  return(0);
 }

int reset_DSA(void)
 {
  scr[0] [0].s_char = 0x00E2;
  scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
  size      = c_header.dsa_size;
  DOSsec = c_header.dsa_sec;
  stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
  if ((result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
           hst.SECTOR, 4, buffer)) != 0)
        {
           put_str(" [result=");put_hex(result);put_str("] ");
           put_str(" <TRACK=");put_hex(hst.TRACK);put_str("> ");
           return(result);
        }
  while(*diskette != 0x00); /* wait for disk drive to stop spinning */
  if((SaveDosSwap()) == 0)
   {
        size      = c_header.dsa_size;
        movedata(FP_SEG(buffer), FP_OFF(buffer),
              FP_SEG(swap_save), FP_OFF(swap_save),size);
       }
     else
       {
        scr[0] [0].s_char = 0x00B0;
       }
     if(swap_save)
       {
             scr[0] [0].s_char = 0x00E0;
             scr[0] [0].s_attr = ((scr[0] [0].s_attr > > 4) + (scr[0] [0].s_attr < < 4)) & 0x77;
             RestoreDosSwap();
             scr[0] [0].s_char = 0x00E1;
             scr[0] [0].s_attr = ((scr[0] [0].s_attr > > 4) + (scr[0] [0].s_attr < < 4)) & 0x77;
       }
     else
       {
             put_str("[ERROR with SWAP_SAVE]");
             put_str("\n\r");
             put_str(swap_save);
             return(-1);
       }
    return(0);
  }

/*****
Function: GetExtErr
get extended error information
*****/
void GetExtErr(struct ExtErr *ErrInfo)
{
  union  REGS regs;

  if (_osmajor > = 3)   /* only for DOS 3 and above */
  {
    regs.h.ah = GET_EXTERR;
    regs.x.bx = 0; /* must be zero */
    intdos(&regs,&regs);
    ErrInfo- > errax = regs.x.ax;
    ErrInfo- > errbx = regs.x.bx;
    ErrInfo- > errcx = regs.x.cx;
  }
}

/*****
Function: SetExtErr
set extended error information
*****/
void SetExtErr(struct ExtErr *ErrInfo)
{
  union  REGS regs;
  struct SREGS segregs;

  if (_osmajor > = 3)   /* only for DOS  3 and above */
  {
    regs.x.ax = SET_EXTERR;
    regs.x.bx = 0;        /* must be zero */
    segread(&segregs);  /* put address of err info in DS:DX */
    regs.x.dx = (int) ErrInfo;
    intdosx(&regs,&regs,&segregs);
    }
  }
  /*****
  Function: GetPSP - returns current PSP
  *****/
  unsigned GetPSP(void)
  {
     if (_osmajor == 2)
     {
        if (! crit_err_ptr) /* must not have called InitInDos */
           return 0;

        *crit_err_ptr = 0xFF;  /* force use of proper stack */
        regs.h.ah = GET_PSP_DOS2;
        intdos(&regs,&regs);
        *crit_err_ptr = 0;
     }
     else
     {
        regs.h.ah = GET_PSP_DOS3;
        intdos(&regs,&regs);
     }
     return regs.x.bx;
  }

  /*****
  Function: SetPSP - sets current PSP
  *****/
  void SetPSP(unsigned segPSP)
  {
     if (!crit_err_ptr)          /* must not have called InitInDos */
        return;
     *crit_err_ptr = 0xFF;       /* force use of correct stack    */
     regs.h.ah = SET_PSP;
     regs.x.bx = segPSP;         /* pass segment value to set     */

     intdos(&regs,&regs);
     *crit_err_ptr = 0;
  }

  int get_drive_info(char *drv)
  {
   char  *drv_letter="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   int r;

   if (_argc > 2)
    {
     drv[0] = toupper(_argv[2][0]);
     drv[1] = 0x00;
     drive = strcspn(drv_letter,drv);
    }
   else
    {
     drive    = 0x00;
     drv[0]   = drv_letter[drive];
     drv[1]   = 0x00;
    }
   if (drv[0] >= 'C')
    {
      drive + = 0x7E;
      r = getDPT(drive, &dpt);
      if ( r > -1 )
             {
             getBPB(drive, dpt.starthead, dpt.starttrack, dpt.startsec, &bpb);
             if(DEBUG)
               {
                 printDPT(dpt);
                 printBPB(bpb);
               }
             }
      maxsides  = dpt.endhead;
      skiptrack = bpb.nspt;
    }
  else
    {
      maxsectors = 8L;
      maxsides   = 1L;
      maxtracks  = 79L;
      r = getBPB(drive,0,0,1,&bpb);
      if(DEBUG)
             printBPB(bpb);
      skiptrack = 0;
    }
  if (r > -1)
   return (drive);
   else return (r);
}

void get_time_date(void)
{
 struct date ddate;
 struct time dtime;
 getdate(&ddate);
 gettime(&dtime);
 c_header.time = ((dtime.ti_hour << 0x0B) + (dtime.ti_min << 0x05)
                          + (dtime.ti_sec >> 1));
 c_header.date = ( ( (ddate.da_year-1980) << 0x09)
                          + (ddate.da_mon << 0x05) + ddate.da_day);
}

DWORD cluster2sector(DWORD cluster)
 {
 return (DWORD)( ((cluster-2) * bpb.spc)
      +  (bpb.ressec + (bpb.nfats*bpb.nspf)
      +((bpb.nroot_dir*32)/bpb.bps) ));
 }

DWORD sector2cluster(DWORD sector)
 {
   return(DWORD)((sector+1)/bpb.spc);
 }
```

CRAMINT.C Functions to replace Interrupts and Utilities associated with interrupts.

```
/*
                                    CRAMINT.C

        Most of the interrupt routines are declared here.
*/

#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <bios.h>
#include "tsr.h"
#include "CRAM.H"

extern unsigned         ss_save; /* slot for stack segment register */
extern unsigned         sp_save; /* slot for stack pointer register */

extern int            tsr_already_active;   /* true if TSR active */
extern int            popup_while_dos_busy; /* true if hot key hit while dos busy */
extern int            int_28_in_progress;   /* true if INT 28 in progress */
extern int            unsafe_flag;          /* true if INT 13 in progress */
unsigned              keycode;
char                  buf[20];              /* work buffer */
unsigned long         TerminateAddr;  /* used during de-install */
union      REGS       regs;           /* register work structures */
struct SREGS          sregs;
struct ExtErr         ErrInfo;        /* save area for extended error info */
int                   hot_key;        /* keycode for activation */
int                   shift_key;      /* shift status bits (alt, ctrl..) */
int                   user_key_set = 0;

/* Save areas for old interrupt pointers */
INTVECT                 old_int8, old_int9, old_int10, old_int13,
                               old_int1b, old_int23, old_int24, old_int2a;

extern int            dos_critical;   /* used by DOSSWAP.C */

void interrupt far new_int2a(INTERRUPT_REGS);
extern  void far        set_stack(void);
extern  void far        restore_stack(void);

/* PROTOTYPES FOR THIS MODULE */

extern void mem_save();
extern void interrupt far new_int13(void); /* in TSRUTIL.ASM */
extern void interrupt far old_int28(void);
extern void interrupt far new_int8();
extern void interrupt far new_int9();
extern void interrupt far new_int28();
```

```
void interrupt far new_int1b();
void interrupt far new_int23();
void interrupt far new_int24(INTERRUPT_REGS r);

void      tsr_exit(void);
void      usage(char *);
int       UnlinkVect(int Vect, INTVECT NewInt, INTVECT OldInt);
void      parse_cmd_line(int argc, char *argv[]);

extern int          int_1C_in_progress;
extern int          keys_punched;

/*********
 * CTRL-BREAK INTERRUPT HANDLER
 *********/
void interrupt far new_int1b()
{
   /* do nothing */
}

/**********
 * CTRL-C INTERRUPT HANDLER
 **********/
void interrupt far new_int23()
{
   /* do nothing */
}

/**********
 * CRTITICAL ERROR INTERRUPT HANDLER
 **********/
void interrupt far new_int24(INTERRUPT_REGS r)
{
   if (_osmajor > = 3)
      r.ax = 3;  /* fail dos function */
   else
         r.ax = 0;
   put_str("\n\rCRITICAL *** ERROR *** IN SYSTEM \n\r");
   put_str("\n\rCRAM will try to Recover or SHUT system Down \n\r");
   put_str("\n\r     SEE SCREEN DUMP  \n\r");
   geninterrupt(0x05);
   tsr_exit();
}

/*********
 * DOS INTERNAL INTERRUPT HANDLER
 *********/
void interrupt far new_int2a(INTERRUPT_REGS r)
{
  switch (r.ax & 0xff00)
  {
    case 0x8000:   /* start critical section */
      dos_critical++;
      break;
    case 0x8100:   /* end critical section */
    case 0x8200:   /* end critical section */
      if (dos_critical)    /* don't go negative */
        dos_critical--;
      break;
```

```
        default:
          break;
      }
      (* old_int2a)();
  }

  // only restores OldInt if someone hasn't grabbed away Vect
  int UnlinkVect(int Vect, INTVECT NewInt, INTVECT OldInt)
  {
      if (NewInt == getvect(Vect))
      {
          setvect(Vect, OldInt);
          return 0;
      }
      return 1;
  }

  void tsr_exit(void)
  {
      set_stack();
      /* put interrupts back the way they were, if possible */

      if (!(UnlinkVect(8, new_int8, old_int8)      |
            UnlinkVect(9, new_int9, old_int9)      |  // Do not use ||, we
                UnlinkVect(0x28, new_int28, old_int28) |  // don't want early out
                UnlinkVect(0x13, new_int13, old_int13) |
                UnlinkVect(0x2a, new_int2a, old_int2a)    ))
      {
              // Set parent PSP, stored in PSP of TSR, to the current PSP
              *(int far *)(((long)_psp << 16) + PSP_PARENT_PSP) = GetPSP();

              // Set terminate address in PSP of TSR
              *(long far *)(((long)_psp << 16) + PSP_TERMINATE) = TerminateAddr;

              /* set psp to be that of TSR */
          SetPSP(_psp);

          /* exit program */
          bdos(DOS_EXIT, 0, 0);
      }
      restore_stack();
}

void usage(char *progname)
{
    fputs("Usage: ", stdout);
    puts(progname);
    puts(" [-d to deinstall] [-k keycode shift-status] [-f multiplex id]");
    puts(" Valid multiplex id");
    puts("    00 through 15 specifies a unique INT 2F ID");
    puts(" Valid shift-status is any combination of:");
    puts("    1 = Right Shift");
    puts("    2 = Left Shift");
    puts("    4 = CTRL");
    puts("    8 = ALT");
  /* exit(1);*/
}

void do_deinstall(char *progname)
```

```
{
   fputs(progname, stdout);
   switch (deinstall())
   {
      case 1:
         puts(" was not installed");
         break;
      case 2:
         puts(" deinstalled");
         break;
      default:
         puts(" deactivated but not removed");
         break;
   }
   exit(0);
}

int set_shift_key(unsigned sh)
{
   /* figure out, report on shift statuses */
   /* make sure shift key < 0x10 and non-zero */
   if (((shift_key = sh) < 0x10) && shift_key)
   {
      printf("Activation: %s%s%s%sSCAN=%d\n",
      shift_key & RIGHT_SHIFT ? "RIGHT " : "",
      shift_key & LEFT_SHIFT ? "LEFT " : "",
      shift_key & CTRL_KEY ? "CTRL " : "",
      shift_key & ALT_KEY ? "ALT " : "",
        hot_key);
      return 1;
   }
   else /* error, bad param */
   {
    puts("Invalid Shift-Status");
    return 0;
   }
}

void parse_cmd_line(int argc, char *argv[])
{
   int i;
   int tmp;

   for (i = 1; i < argc; i++)   /* for each cmdline arg */
      if ((argv[i][0] == '-') || (argv[i][0] == '/'))
         switch(toupper(argv[i][1]))
         {
            case 'D':
               do_deinstall(argv[0]);
               break;
            case 'K':    /* set pop-up key sequence */
               user_key_set = 1;
               i++;     /* bump to next argument */
               if ((hot_key = atoi(argv[i])) != 0)
               {
                  i++;  /* bump to next argument */
                  if (! set_shift_key(atoi(argv[i])))
                     usage(argv[0]);
               }
```

```
          else
             usage(argv[0]);
          break;
               default:   /* invalid argument */
          usage(argv[0]);
      }  /* end switch */
    else
      usage(argv[0]);
}
```

CRAMMEM.C Functions to manipulate the memory arenas or MCB.

```
 /*                          CRAMMEM.C

  functions to use  DOS MCB chain(s)
 */

 #include <stdlib.h>
 #include <stdio.h>
 #include <ctype.h>
 #include <string.h>
 #include <mem.h>
 #include <dos.h>
 #include <conio.h>
 #include "cram.h"

 #ifndef MK_FP
 #define MK_FP(seg,ofs)  ((FP)(((DWORD)(seg) << 16) | (ofs)))
 #endif

int       get_mcb_chain(MCB far * mcb, BYTE  *buffer);
int       set_mcb_chain(BYTE *buffer, WORD psp);
                showMCB(char  *buffer);
char far  *progname_fm_psp(unsigned psp);
void      display_progname(MCB far *mcb);
MCB far   *get_cmd_mcb(MCB far *mcb);
MCB far   *IS_CRAM(MCB far *mcb);
BOOL       belongs(void far *vec, unsigned start, unsigned size);
void      display(MCB far *mcb);
char far  *env(MCB far *mcb);
void      display_progname(MCB far *mcb);
void      display_cmdline(MCB far *mcb);
void      display_vectors(MCB far *mcb);
WORD        low_mem(MCB far *mcb);


MCB far *get_mcb (void)
{
 union REGS reg;
 struct SREGS seg;
 unsigned far *tmpp;

 segread(&seg);
 reg.h.ah = 0x52;
 intdosx( &reg, &reg, &seg);
 tmpp = (unsigned far *) MK_FP( seg.es, reg.x.bx - 2 );
 return ((MCB far *) MK_FP( *tmpp, 0));
}

set_mcb_chain(BYTE *buffer, WORD psp)
{
   MCB            far  *mcb;
   struct MCB_TBL far  *tblptr;
   int                  i = 0, j=0;
```

```
      j = atoi(&buffer[511]);
      for (;;)
         {
             tblptr = (struct MCB_TBL far *)(&buffer[i]);
             if(tblptr->mcb.owner == psp || psp == 0)
               {
                 DWORD far *addr = TERM_FM_PSP(mcb->owner);
                  mcb = MK_FP(tblptr->addr,0);
                  mcb->type  = tblptr->mcb.type;
                  mcb->owner = tblptr->mcb.owner;
                  mcb->size  = tblptr->mcb.size;
                  *addr = *(&tblptr->term_addr);
               }
             if (tblptr->mcb.type == 'M' && j > 0)
               {
                  i+=sizeof(struct MCB_TBL);
                  j--;
               }
             else
                 return (tblptr->mcb.type == 'Z');
             }
 }

 int get_mcb_chain(MCB far * mcb, BYTE  *buffer)
 {
     MCB_TBL far *tblptr;
     int i = 0, j=0;

     for (;;)
        {
            DWORD far *addr = TERM_FM_PSP(mcb->owner);
            tblptr = (MCB_TBL far *)(&buffer[i]);

            tblptr->mcb.type = mcb->type;
            tblptr->mcb.owner = mcb->owner;
            tblptr->mcb.size = mcb->size;
            tblptr->term_addr = *addr;
            tblptr->addr = (WORD) FP_SEG(mcb);

            if (mcb->type == 'M')
              {
                mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
                i+=sizeof(struct MCB_TBL);
                j++;
              }
            else
              {
                itoa(j, &buffer[511],16);              /*Store the number of MCBs */
                return (j);         /*Return the number of MCBs present */
              }
            }
}


showMCB(char  *buffer)
 {
   int i=0, j=0, k=0;
   MCB_TBL far *mcb_tbl;
```

```
    k = atoi(&buffer[511]);

    clear_screen(0xB0);
  /*  gotoxy(1,1);*/
   /* k = get_mcb_chain(first_mcb, buffer);*/
    putstr("\n\r = = = = = Begin MCB table values  = = = = = \n\r");
    while (i> =0 && k > = 0)
      {
            textcolor(15);
            textbackground(1);
            gotoxy(5,((2*j)+3));clreol();
            mcb_tbl = (struct MCB_TBL far *)(&buffer[i+ =sizeof(MCB_TBL)]);

            putstr(" ");put_hex(mcb_tbl->addr);put_str(" ");put_chr(mcb_tbl->mcb.type);
            put_hex(mcb_tbl->mcb.owner);put_str(" ");put_hex(mcb_tbl->mcb.size);put_str(" ");
            put_long((long)(mcb_tbl->mcb.size < < 4));putstr(" ");

  /*        cprintf("\n\r %.4X   %c    %.4X    %.4X  (%lu) \n\r",
            mcb_tbl->addr,
            mcb_tbl->mcb->type,
            mcb_tbl->mcb->owner,
            mcb_tbl->mcb->size,
            (long)(mcb_tbl->mcb->size < < 4)
             );*/

            k--;
            if (strstr(strupr(_argv[1]), "+")) display((MCB far *)(&mcb_tbl->mcb));
            if (mcb_tbl->mcb.type = = 'Z')
             {
              i = -1;
              break;
             }
            j+ +;
            if(j> =10)
             {
               j=0;
               textcolor(14);
               textbackground(3);
               put_str("Press any key to continue ... ");
               getch();
             }
    }
    putstr("\n\r = = = = = End MCB table values  = = = = = \n\r");

    put_str("Press any key to continue ... ");
    getch();
    return(0);
 }


mcb_chk(MCB far *mcb)
{
   for (;;)
      if (mcb->type = = 'M')
         mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
      else
         return (mcb->type = = 'Z');
}
```

```
 MCB far *IS_CRAM(MCB far *mcb)
 {
   static MCB far * tmpmcb;
   static WORD tmpowner;
   char far *s;
   char buf[128];
   unsigned char i=0;

   buf[0] = '\0';
   mcb = get_cmd_mcb(mcb);
   tmpmcb = mcb;
   tmpowner = mcb->owner;
/*   mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0); */
   for (;;)
      switch (mcb->type)
      {
            case 'M' : /* Mark : belongs to MCB chain */
             {
                  mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
                  if(tmpowner == mcb->owner)
                  tmpmcb = mcb;
                  break;
             }
            case 'Z' : /* Zbikowski : end of MCB chain */
                  {
                    s = progname_fm_psp(FP_SEG(tmpmcb) + 1);
                    while((s) && (i <= 128))
                       buf[i++] = *s++;
                    if ( strstr(buf,"CRAM") != NULL)
                     {
                     put_str("\n\rPrograms can be restored at address ");
                     put_hex(FP_SEG(tmpmcb) + tmpmcb->size + 1);
                     put_str(":");put_hex(FP_OFF(tmpmcb));put_str("\n\r");
                     return(tmpmcb);
                     }
                    else
                     {
                      printf("\b\b");
                      put_str("\n\r CRAM cannot continue :: need to be the first");
                      put_str("\n\r Program to load in your Autoexec.bat file");
                      put_str("\n\r the file {");
                      put_str(buf);put_str("} was found instead\n\r");
                      return((MCB far *)NULL);
                     }
                  }
           default :
                  {
                   put_str("Error in MCB chain when checking for CRAM \r\n");
                   exit(-2);
                  }
        }
}

WORD low_mem(MCB far *mcb)
{
  if(mcb)
   {
    mcb = IS_CRAM(mcb);
    if(mcb == NULL)
```

```
            return (0);
        else
            return(FP_SEG(mcb) + mcb->size + 1);
      }
      put_str("Invalid MCB ptr ");
      return(0x0000);
  }

 MCB far *get_cmd_mcb(MCB far *mcb)
 {
    static void far *vect_2e = (void far *) 0;
    static MCB far * tmpmcb;
    static WORD tmpowner;

    if(! vect_2e)
      vect_2e = GETVECT(0x2E);
    for (;;)
       switch (mcb->type)
       {
              case 'M' : /* Mark : belongs to MCB chain */
                    if (mcb->owner == 0x0000)
                      {
                        mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
                        break; /* Skip over free blocks between current owner*/
                      }

                    if(belongs(vect_2e, FP_SEG(mcb), mcb->size))
                      {
                        mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
                        tmpmcb = mcb;
                        tmpowner = mcb->owner;
                      }
                    else
                     mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
                    if(tmpowner == tmpmcb->owner)
                     tmpmcb = mcb;
                    break;

          case 'Z' : /* Zbikowski : end of MCB chain */
                    if(tmpmcb)
                    return(tmpmcb);
                    else
                    return((MCB far *)NULL);
              defasult :
                {
                    put_str("ERROR in MCB chain \r\n");
                    exit(-5);
                }
      }
}

BOOL belongs(void far *vec, unsigned start, unsigned size)
{
   unsigned seg = FP_SEG(vec) + (FP_OFF(vec) >> 4); /* normalize */
   return (seg >= start) && (seg <= (start + size));
}


void walk(MCB far *mcb)
```

```
 {
   printf("\n\rSeg    Owner   Size\n\r");
   for (;;)
     switch (mcb->type)
     {
       case 'M' : /* Mark : belongs to MCB chain */
         display(mcb);
         mcb = MK_FP(FP_SEG(mcb) + mcb->size + 1, 0);
         break;
       case 'Z' : /* Zbikowski : end of MCB chain */
         display(mcb);
         return;
          default :
                   {
                    put_str("ERROR in walking MCB chain \r\n");
                    exit(-1);
                   }
     }
 }

void display(MCB far *mcb)
{
  static void far *vect_2e = (void far *) 0;
  unsigned env_seg=0;

  printf("%04X  %04X   %04X (%6lu)  ",
        FP_SEG(mcb), mcb->owner, mcb->size, (long) mcb->size << 4);

  if (IS_PSP(mcb))
  {
        void far *e = env(mcb);

        if ((env_seg == FP_SEG(e)))
         printf("%04X    ", env_seg);
        else
         printf("        ");

    display_progname(mcb);
  }

  if (! vect_2e)
    vect_2e = GETVECT(0x2e);  /* do just once */
  if (! mcb->owner)
    printf("free ");
  /* 0008 is not really a PSP; belongs to CONFIG.SYS */
  else if (mcb->owner == 8)
        printf("config.sys ");
  /* INT 2Eh belongs to master COMMAND.COM (or other shell) */
  else if (belongs(vect_2e, FP_SEG(mcb), mcb->size))
    printf("%s ", getenv("COMSPEC"));
  /* presence command line is independent of program name */
  if (IS_PSP(mcb))
    display_cmdline(mcb);
  display_vectors(mcb);
  printf("\n");
}

char far *env(MCB far *mcb)
{
      char far *e;
      unsigned env_mcb;
      unsigned env_owner;

      /*
         if the MCB owner is one more than the MCB segment then
            psp := MCB owner
            env_seg := make_far_pointer(psp, 2Ch)
            e := make_far_pointer(env_seg, 0)
         else
            return NULL
      */
      if (IS_PSP(mcb))
         e = MK_FP(ENV_FM_PSP(mcb->owner), 0);
      else
         return (char far *) 0;

    /* Check to see if the selected environment belongs to the present PSP */

      env_mcb = MCB_FM_SEG(FP_SEG(e));
      env_owner = ((MCB far *) MK_FP(env_mcb, 0))->owner;
      return (env_owner == mcb->owner) ? e : (char far *) 0;
   }

   char far *progname_fm_psp(unsigned psp)
   {
      char far *e;
      unsigned len;

      /* is there an environment? */
      if (! (e = env(MK_FP(MCB_FM_SEG(psp), 0))))
         return (char far *) 0;

      /* program name only available in DOS 3+ */
      if (_osmajor >= 3)
      {
         /* skip past environment variables */
            do e += (len = fstrlen(e)) + 1;
         while (len);

         /*
               e now points to WORD containing number of strings following the
               environment; check for reasonable value: signed because it
               could be FFFFh; should  normally equal to 1
            */
            if ((*((signed far *) e) >= 1) && (*((signed far *) e) < 10))
            {
               e += sizeof(signed);
               if (isalpha(*e))
                     return e; /* could make canonical with INT 21h AH=60h */
            }
      }
      return (char far *) 0;
   }

   void display_progname(MCB far *mcb)
   {
      char far *s;
    if (IS_PSP(mcb))
            s = (char far *)progname_fm_psp((FP_SEG(mcb) + 1));
            if(s)
/*          if ( (s = (char far *)progname_fm_psp((FP_SEG(mcb) + 1)) ) )*/
               printf("%Fs ", s);
}

void display_cmdline(MCB far *mcb)
{
    /*
       psp := MCB owner
       cmdline_len := psp[80h]
       cmdline := psp[81h]
       print cmdline (display width := cmdline_len)
    */
    int len = *((BYTE far *) MK_FP(mcb->owner, 0x80));
    char far *cmdline = MK_FP(mcb->owner, 0x81);
    /* Some versions of DOS store other values in the command line area
       of the environment block thus need to check for printable or
       valid characters.
    */
    if(*cmdline >= 0x20 && *cmdline <= 0x7F)
    printf("%.*Fs ", len, cmdline);
}

void display_vectors(MCB far *mcb)
{
    static void far **vec = (void far **) 0;
    int i;
    int did_one=0;
    if (! vec)
    {
       if (! (vec = calloc(256, sizeof(void far *))))
             put_str("insufficient memory \r\n");
       for (i=0; i<256; i++)
          vec[i] = GETVECT(i);
    }
    for (i=0; i<256; i++)
       if (vec[i] && belongs(vec[i], FP_SEG(mcb), mcb->size))
       {
          if (! did_one) { did_one++; printf("["); }
          printf("%02X ", i);
          vec[i] = 0;
       }
    if (did_one) printf("]");
}
```

CRAMTOOL.C Functions not found in CRAMUTIL.C and are essential for the operation of CRAM.

```
/*
                         CRAMTOOL.C

   Some tools used to install CRAM and to manipulate the
   Directory structure at the FAT level. Included are tools
   to calculate memory checksum and to test various
   sections of memory.

*/


#include <bios.h>
#include <stdio.h>
#include <dos.h>
#include <dir.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include <time.h>
#include "cram.h"

extern struct  BPB          bpb;
extern struct  DPT          dpt;
extern struct  HST          hst;
extern struct  CRAM_HEADER  c_header;
extern long    maxtracks,
                    maxsides,
                    maxsectors,
                    skiptrack;
extern long    DOSsec;
extern DWORD   data_sec;
extern long    result;
extern BYTE     buffer[];
extern int     nsects;
extern BYTE far *conv_mem_ptr;
extern int     drive;
extern char     *drv;
extern struct  free_fat    f_free;
extern DWORD   clust_req;
extern int     reserved_sec;
extern int     DEBUG;
/*\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*/
struct CRAM_ver c_ver = {  1, 0, 1, 1 };
/*\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*/
int   getfat_info(struct free_fat *freefat, WORD clust_req);
long  mem_checksum(WORD seg_start, WORD offs_start, DWORD mem_size);
int   get_fatfree(struct free_fat *freefat, WORD clust_req);
int   put_fat(struct free_fat freefat, DWORD data_sec);
int    put_f_name(struct free_fat f_free, DWORD data_sec);
long   get_f_name(void);
char   pause(int err_num);
int    show_disk(void);
long   get_data_sec(void);
int    ckeckdisk(void);
int    format_CRAM(long start_sec, long end_sec);
/*==============================================================================
====*/

int    getfat_info(struct free_fat *freefat, WORD clust_req)
{
   struct dir_entry d_entry;
   long i, j, k, m, logical_sector, f_pos;
   DWORD f_begin, f_end, f_size=0, f_mark=0;
   WORD f_entry;
   WORD fat_entry[2];
   WORD fatsize;
   union FATS *f_ptr;
   gotoxy(1,1);clreol();
   cprintf("Getting FAT Information...");
   DOSsec = 1;
   fatsize = (bpb.nspf * bpb.nfats);
   nsects = 1;
   j = i = 0L;
  for (m=0; m<fatsize; m++)
    {
          if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
            return(result);
          f_pos = i = 0L;
          while ((f_pos + sizeof(f_ptr) +1) <= (nsects * bpb.bps))
            {
             if (m == (bpb.nspf-1))   /*(j % ( bpb.bps * bpb.nspf )) == 0)*/
               {
                    j = 0L;
                    pause(0);
                    printf("\n================\nNext Copy of FAT \n================\n");
               }

            switch(dpt.sysid)
               {
                    case D_FAT16:
                      {     f_pos = ((i * 2) % bpb.bps);
                                 f_ptr = (union FATS *) (&buffer[f_pos]);
                                 f_entry = f_ptr->fat_16.fat16;
                                 i++;j++;
                                 break;
                      }
                    case D_FAT12:
                    case      0:
                      {
                                 f_pos = (((i * 3) / 2) % bpb.bps);
                                 f_ptr = (union FATS *) (&buffer[f_pos]);
                                 if(i & 1) f_entry = f_ptr->fat_12_hi.fat12;
                                 else   f_entry = f_ptr->fat_12_lo.fat12;
                                 i++;j++;
                                 break;
                      }
             }
   if (f_entry = = 0 && f_mark = = 0)
    {
          f_begin = j-1;
          f_size = 1;
          f_end   = j-1;
          f_mark =1;
    }
   if (f_entry = = 0 && f_mark = = 1)
    {
     f_size+ +;
     f_end = j-1;
    }
   else
    {
          if (f_size > = clust_req && f_mark != 0)
            {
            freefat->fbegin = f_begin;
            freefat->fend = f_end;
            freefat->fsize = f_size;
            cprintf("\n\r More Than %u Free clusters at %lu = = = %lu ( size %lu )",
                                  clust_req, f_begin, f_end, f_size);
           }
 f_mark = 0;
    }
  goto LOOP;
  if ((f_entry < 0xFF0) && (f_entry> 0x000))
    printf("\n%ld  {%-3.3u} [Sectors %ld - %ld ] <",
             j-1, f_entry, cluster2sector(f_entry),
              cluster2sector(f_entry) + 1);
  else
    printf("\n%ld  {%-3.3X}  <", j-1, f_entry);
  switch (f_entry)
   {
         case 0x000:  printf("FREE");break;
         default :
           {
            if (dpt.sysid = = D_FAT16)
                    {
                     if (f_entry = = 0xFFF7)   printf("BAD");
                     else
                     if (f_entry > = 0xFFF0 && f_entry < = 0xFFF6 )
                      printf ("RESERVED");
                     else
                     if (f_entry > = 0xFFF8 && f_entry < = 0xFFFF )
                      printf ("EOF");
                     else
                      printf("OCCUPIED");
                    }
           else
                    {
                     if (f_entry = = 0xFF7)      printf ("BAD");
                     else
                     if (f_entry > = 0xFF0 && f_entry < = 0xFF6 ) printf ("RESERVED");
                     else
                     if (f_entry > = 0xFF8 && f_entry < = 0xFFF ) printf ("EOF");
                     else
                    printf("OCCUPIED");
                    }
           break;
```

```
                          }
                 }/* end switch */
               printf(">     ");
               if((j%20) == 0)
                  {
                         if (pause(0) == 0x1B) return(1);
                  }
                  LOOP:
                  ;
             } /* end while */
       DOSsec++;
    } /* end for loop */
   return(0);
}


int get_fatfree(struct free_fat *freefat, WORD clust_req)
{
   long i, j, m, f_pos;
   DWORD f_begin, f_end, f_size=0, f_mark=0;
   WORD f_entry;
   WORD fatsize;
   union FATS *f_ptr;

   gotoxy(1,1);clreol();
   cprintf("Checking for Free Clusters ...");
   DOSsec= 1;
   fatsize = bpb.nspf;
   nsects = 1;
   j = i = 0L;
   for (m=0; m<fatsize; m++)
     {
           if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
             return(-result);
           f_pos = i = 0L;
           while ((f_pos + sizeof(f_ptr) +1) <= (nsects * bpb.bps))
            {
              switch(dpt.sysid)
                {
                      case D_FAT16:
                        {     f_pos = ((i * 2) % bpb.bps);
                                  f_ptr = (union FATS *) (&buffer[f_pos]);
                                  f_entry = f_ptr->fat_16.fat16;
                                  i++;j++;
                                  break;
                        }
                      case D_FAT12:
                      case      0:
                        {
                                  f_pos = (((i * 3) / 2) % bpb.bps);
                                  f_ptr = (union FATS *) (&buffer[f_pos]);
                                  if(i & 1) f_entry = f_ptr->fat_12_hi.fat12;
                                  else   f_entry = f_ptr->fat_12_lo.fat12;
                                  i++;j++;
                                  break;
                        }
               }
             if (f_entry == 0 && f_mark == 0)
               {
                    f_begin = j-1;
                    f_size = 1;
                    f_end   = j-1;
                    f_mark =1;
               }
             if (f_entry == 0 && f_mark == 1)
               {
               f_size++;
               f_end = j-1;
               }
             else
               {
                    if (f_size >= clust_req && f_mark != 0)
                      {
                      if(f_size > freefat->fsize)
                        {
                              freefat->fbegin = f_begin;
                              freefat->fend = f_end;
                              freefat->fsize = f_size;
                        }
                      if(DEBUG)
                      cprintf("\n\r More Than %u Free clusters at %lu === %lu {size %lu} [Sec %lu]",
                                         clust_req, f_begin, f_end, f_size, DOSsec);
                      }

            f_mark = 0;
               }

            } /* end while */
        DOSsec++;
      } /* end for loop */
      if (f_size >= clust_req && f_mark != 0)
                      {
                      if(f_size > freefat->fsize)
                        {
                              freefat->fbegin = f_begin;
                              freefat->fend = f_end;
                              freefat->fsize = f_size;
                        }
                      if(DEBUG)
                      cprintf("\n\r More Than %u Free clusters at %lu === %lu {size %lu} [Sec %lu]",
                                         clust_req, f_begin, f_end, f_size, DOSsec);
                      }
    if(freefat->fsize < clust_req)
      return(-1);
    else
      return(freefat->fsize);
}

int put_fat(struct free_fat freefat, DWORD data_sec)
{
   DWORD i, j, k, m, f_pos;
   DWORD f_begin, f_end, f_size=0, f_mark=0;
   WORD f_entry;
   WORD fatsize, cps;
   union FATS *f_ptr;

   cps = dpt.sysid == D_FAT12 ? (bpb.bps * 2 / 3)+1 : (bpb.bps / 2 );
   for (k=0; k<bpb.nfats; k++)
```

```
{
gotoxy(1,1);clreol();
cprintf("Updating FAT ...%d", k+1);
DOSsec= ((k*bpb.nspf)+1+(f_free.fbegin / cps));
fatsize = (f_free.fsize / cps) + 1;
nsects = 1;
j = ((f_free.fbegin / cps) * cps);
i = 0L;
if(DEBUG)
 {
        cprintf("\n\rFat Locations : Dos Sector %lu ; Size %d ; Data Sector at %lu",
                DOSsec, fatsize, data_sec);
        pause(0);
 }
for (m=0; m<fatsize+1; m+ +)
  {
       if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
         return(result);
       f_pos = i = 0L;
       while ( (f_pos + sizeof(f_ptr) +1) < = (nsects * bpb.bps))
         {
          switch(dpt.sysid)
            {
                case D_FAT16:
                  {    f_pos = ((i * 2) % bpb.bps);
                          f_ptr = (union FATS *) (&buffer[f_pos]);
                          f_entry = f_ptr->fat_16.fat16;
                          if (f_entry = = 0 && j > = f_free.fbegin && j < = f_free.fend)
                            {
                              if(j = = f_free.fend)
                                  {
                                   f_entry = 0xFFFF;
                                  }
                              else f_entry = j +1;
                              f_ptr->fat_16.fat16 = f_entry;
                            }
                          if(DEBUG)
                            {
                              if ((f_entry < 0xFFF0) && (f_entry> 0x0000))
                                     printf("\n%lu  {%-4.4u} [Sectors %ld - %ld ] <",
                                              j, f_entry, cluster2sector(f_entry),
                                              cluster2sector(f_entry) + bpb.nfats-1);
                              else
                                     printf("\n%d  {%-4.4X}  <", j, f_entry);
                            }
                          break;
                  }
                case D_FAT12:
                case     0:
                 {
                         f_pos = (((i * 3) / 2) % bpb.bps);
                         f_ptr = (union FATS *) (&buffer[f_pos]);
                         if(i & 1) f_entry = f_ptr->fat_12_hi.fat12;
                         else   f_entry = f_ptr->fat_12_lo.fat12;
                         if (f_entry = = 0 && j > = f_free.fbegin && j < = f_free.fend)
                           {
                            if(j = = f_free.fend)
                              {
                                   f_entry = 0xFFF;
                           }
                          else
                           f_entry = j;
                         if(i & 1) f_ptr->fat_12_hi.fat12 = f_entry;
                          else    f_ptr->fat_12_lo.fat12 = f_entry;
                        }
                       if(DEBUG)
                        {
                         if ((f_entry < 0xFF0) && (f_entry> 0x000))
                              printf("\n%d  {%-3.3u} [Sectors %ld - %ld ] <",
                                 j, f_entry, cluster2sector(f_entry),
                                 cluster2sector(f_entry) + bpb.nfats-1);
                         else
                              printf("\n%d  {%-3.3X}  <", j, f_entry);
                        }
                       break;
                }
           }
        i++;j++;

         } /* end while */
        if ((result = putsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
                return(result);
     DOSsec++;
   } /* end for loop */
  }/* end of outer for loop */
  return(0);
}


long mem_checksum(WORD seg_start, WORD offs_start, DWORD mem_size)
{
 long sum=0x0000L, cnt=0, k, l, m=0x0000L, n;
 long i, seg, offs;
 DWORD far *s_mem;
 ldiv_t   cal;

 cal = ldiv(mem_size, 0x10000);
 k = cal.quot;
 l = cal.rem;
 for (seg=seg_start; seg<=(k*0x1000); seg+=0x1000)
  {
   cnt =0;

   if (k != 0)
    {
       n = 0x10000;
       offs = 0;
    }
   else
    {
       n = mem_size;
       offs = offs_start;
    }
   s_mem = MK_FP(seg, offs);
   while(cnt < n)
   {
         if(DEBUG)
               {
                    put_hex(*s_mem);
                    put_str(" ");
                  }
              sum += *(s_mem++);
              m ^= *(s_mem++);
              cnt+=sizeof(s_mem);
      }
     if(DEBUG)
      {
              gotoxy(1,1);
              printf("%Fp",s_mem);
      }
    }
    cnt = 0;
    offs = 0;
    if (k != 0)
      while(cnt < l)
      {
              s_mem = MK_FP(seg, offs);
              if(DEBUG)
                {
                  gotoxy(1,1);
                  printf("%Fp",s_mem);
                }
              m ^= *(s_mem++);
              sum += *(s_mem++);
              put_hex(*(s_mem));
              cnt+=sizeof(s_mem);
      }
  if(DEBUG)
     {
      gotoxy(60,i);
      printf("%Fp",s_mem);
     }
  sum = (sum >> 8);  /* Put value in lower 16 bits (upper 2 bytes) */
  m   = (m << 8);    /* Put value in upper 16 bits (2 bytes)       */
  return(sum|m);     /* Ored sum with m to get a unique CHECKSUM    */
}

int  get_fat_info(void)
 {
   struct dir_entry d_entry;
   long i, j, k, m, logical_sector;
   WORD f_entry;
   WORD fat_entry[2];
   WORD fatsize;
   union FATS fats;

   DOSsec= 1;
   fatsize = (bpb.nspf * bpb.nfats);
   nsects = 1;
   j = i = 0L;

   /* use as FATS.fat_16.f16  = ?
   or    FATS.fat_12_lo.fat12 =  ?
   or    FATS.fat_12_hi.fat12 =  ?
*/

 for (m=0; m<fatsize; m++)
```

```
   {
    if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
           return(result);
   while (i < (nsects * bpb.bps))
    {
    if ((i % ( bpb.bps * bpb.nspf )) == 0)
      {
      j = 0;
      printf("\n=================\nNext Copy of FAT \n==================\n");
      }
    memcpy(&fat_entry[0], &buffer[i], 2);
    memcpy(&fat_entry[1], &buffer[i+1], 2);
    fat_entry[0] = (fat_entry[0] & 0x0FFF);
    fat_entry[1] = (fat_entry[1] >> 4);
    for (k = 0; k < 2; k++)
     {
      f_entry = fat_entry[k];
      if ((f_entry < 0xFF0) && (f_entry> 0x000))
      printf("\n%ld  {%-3.3u} [Sectors %ld - %ld ] <",
              j+k, f_entry, cluster2sector(f_entry),
              cluster2sector(f_entry) + 1);
      else
      printf("\n%ld  {%-3.3X}  <", j+k, f_entry);
      if (f_entry ==  0x000) printf("FREE");
      else
      if (f_entry == 0xFF7 ) printf ("BAD");
      else
      if (f_entry >= 0xFF0 && f_entry <= 0xFF6 ) printf ("RESERVED");
      else
      if (f_entry >= 0xFF8 && f_entry <= 0xFFF ) printf ("EOF");
      else
      printf("OCCUPIED");
      printf(">    ");
     }

   i+=3;j+=2;
   if((j%20) == 0)
     {
          if (pause(0) == 0x1B) return(1);
     }
   }
   }
  return(0);
 }

int  put_f_name(struct free_fat f_free, DWORD data_sec)
 {
   BYTE update=0;
   WORD f_entry;
   WORD fat_entry[2];
   struct dir_entry d_entry;
   struct dir_entry tmp_d_entry=
    {
     "CRAM_MEM","XMD",0xC1," CRAM V1.1", 0x00, 0x00, 0x00, 0x00
    };
   long i, j, k, logical_sector;
   WORD start_cluster = 0;
   struct date ddate;
   struct time dtime;
```

```
     gotoxy(1,1);clreol();
     cprintf("Updating Directory ...");
     start_cluster = f_free.fbegin;
     getdate(&ddate);
     gettime(&dtime);
     tmp_d_entry.f_attrib=(ARCHIVE|R_O|SYSTEM|HIDDEN);
     tmp_d_entry.f_name[4] = 0xFF;
     tmp_d_entry.f_reserved[0] = /*0x00*/ 0xFF;
     tmp_d_entry.f_start_cluster = f_free.fbegin;
     tmp_d_entry.f_size =(long)((f_free.fsize+1) * bpb.bps * bpb.spc);
     DOSsec=(bpb.nspf * bpb.nfats) + 1;
     nsects = 1;
     i = 0L;
     j = (long)((bpb.nroot_dir * 32) / bpb.bps);
     if(DEBUG)
      {
             cprintf("\n\rData Start at %lu \n\r", data_sec);
             cprintf("Start Cluster at %d\n\r", start_cluster);
      }
     for (k=0; k<j; k++)
      {
            i = 0L;
            if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer /*&buffer[(k*bpb.bps)]*/)) != 0)
               return(-result);
            while( i < (bpb.bps) )
              {
               memcpy(&d_entry, &buffer[i], 32);
               if (memcmp(tmp_d_entry.f_name, d_entry.f_name, 8) == 0x00)
                  update = 80;
               if ((   d_entry.f_name    == NULL
                        || d_entry.f_name[0] == 0x00
                        || (d_entry.f_name[0] == 0xE5 && d_entry.f_name[1] == 0x00)
                        || (d_entry.f_name[0] == 0xE5 && d_entry.f_name[4] == 0xFF)
                        || (d_entry.f_name[0] == 0xE5 &&
                          (memcmp(&tmp_d_entry.f_reserved[1], &d_entry.f_reserved[1], 4) == 0x00))
                        || (memcmp(tmp_d_entry.f_name, d_entry.f_name, 8) == 0x00)
                        ) && (update == 0 || update == 80))

                {
                       tmp_d_entry.f_time = (dtime.ti_hour << 0x0B)
                                            +(dtime.ti_min << 0x05)
                                            +(int)(dtime.ti_sec >> 1);
                       tmp_d_entry.f_date = (((ddate.da_year-1980) << 0x09)
                                            +(ddate.da_mon << 0x05)
                                            +ddate.da_day);
/*                     tmp_d_entry.f_size = c_header.f_size;*/
                       memcpy(&d_entry, &tmp_d_entry, 32);
                       memcpy(&buffer[i], &tmp_d_entry, 32);
                       update++;
                }
              else if (d_entry.f_name[0] == 0xE5 && d_entry.f_name[1] == 0x00)
                         {
                           memset(&d_entry.f_name[0], '\0', 32);
                           memcpy(&buffer[i], &d_entry, 32);
                         }
              if(DEBUG)
              printf("\n%.*s.%.*s a[%-2.2u] {%-.*s}t{%-6.6u}d{%-6.6u}c{%-6.6u}s{%-6.6lu}",
                     8,d_entry.f_name,
                     3,d_entry.f_ext,
                        d_entry.f_attrib,
                        10,d_entry.f_reserved,
                        d_entry.f_time,
                        d_entry.f_date,
                        d_entry.f_start_cluster,
                        d_entry.f_size );
                i+ =32;
              }
            if ((result = putsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer/*&buffer[k*bpb.bps]*/)) != 0)
                        return(-result);
            if (update != 0) break;
            DOSsec + +;
      }
              DOSsec = (bpb.nspf * bpb.nfats) + 1;
    if (update > 80) return (-1);
    else
    return(0);
  }

long get_f_name(void)
 {
   BYTE update = 0;
   WORD f_entry;
   WORD fat_entry[2];
   struct dir_entry d_entry;
   struct dir_entry tmp_d_entry =
    {
     "CRAM_MEM","XMD",0xC1," CRAM V1.1", 0x00, 0x00, 0x00, 0x00
    };
   long i, j, k, logical_sector;
   DWORD start_cluster = 0;

   tmp_d_entry.f_attrib = (ARCHIVE|R_O|SYSTEM|HIDDEN);
   tmp_d_entry.f_name[4] = 0xFF;
   tmp_d_entry.f_reserved[0] = /*0x00*/ 0xFF;
   strcpy(d_entry.f_name, "filename");
   DOSsec = (bpb.nspf * bpb.nfats) + 1;
   nsects = 1;
   i = 0L;
   j  = (long)((bpb.nroot_dir * 32) / bpb.bps);

   for (k=0; k<j; k+ +)
   {
           if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
              return(-result);
           DOSsec + +;
   i = 0L;
   while( (i < (bpb.bps )) && d_entry.f_name[0] != 0x00)
    {
           memcpy(&d_entry, &buffer[i], 32);
           if ((memcmp(tmp_d_entry.f_name, d_entry.f_name, 8) = = 0x00) )
             {
              start_cluster = d_entry.f_start_cluster;
              data_sec = cluster2sector(start_cluster);
              return(start_cluster);
             }
           if(DEBUG)
           if (( d_entry.f_name  = = NULL) || (d_entry.f_name[0] = = 0x00))
             {
                  i = i > > 5;
                  cprintf("\n\r (%ld) File(s) found on Disk in Drive [%.1s]", i, drv);
                }
              else
                 cprintf("\n\r%.*s.%.*s a[%-2.2u] {%-.*s}t{%-6.6u}d{%-6.6u}c{%-6.6u}s{%-6.6lu}",
                         8,d_entry.f_name,
                         3,d_entry.f_ext,
                         d_entry.f_attrib,
                         10,d_entry.f_reserved,
                         d_entry.f_time,
                         d_entry.f_date,
                         d_entry.f_start_cluster,
                         d_entry.f_size );
              i+ =32;
       }
     }
      DOSsec=(bpb.nspf * bpb.nfats) + 1;
     return(start_cluster);
  }

 char pause(int err_num)
 {
   char ch;

   switch (err_num)
    {
     case 0: putch(0x07);
                 cprintf ("\n\rPress any Key to Continue ...");
                 ch = getch();
                 break;
     case 1: putch(0x07);
                 cprintf ("\n\rPress any Key to Continue ...ESC to abort ..");
                 ch = getch();
                 if(ch = = 0x1B) exit(2);
                 break;

    }
   return(ch);
 }


int show_disk(void)
 {
  for (DOSsec=0; DOSsec<bpb.ndsksect; DOSsec+ +)
    {
    if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt, buffer)) != 0)
            return(result);
    else printsector(buffer);

    }
   return(0);
 }


long get_data_sec(void)
{
  long dossec;
  char *idptr;
```

```
    char c_id[2][5] = {"RABB", "WGM_"};

    nsects = 1;
    gotoxy(53,1);cprintf("Checking Sector ...");
    for (dossec = 0; dossec < bpb.ndsksect; dossec + + )
     {
      if ((result = getsector(drive, nsects, &dossec, &bpb, &dpt, buffer)) != 0)
              return(-result);
      else
        {
              gotoxy(73, wherey()); printf("%lu", dossec);
              if((memcmp(&buffer[strlen(SIG) + sizeof(bpb) + sizeof(dpt)], c_id[0], 4) = = 0) &&
                 (memcmp(&buffer[508], c_id[1], 4) = = 0))
                {
                  gotoxy(1, wherey());
                  cprintf("Sector %ld   String at 1st idptr %-.*s"
                          " String at 2nd idptr %-.*s ",
                  dossec,  4, buffer[strlen(SIG) + sizeof(bpb) + sizeof(dpt)], 4, &buffer[508]);
                  return(dossec);
                }
        }
     }
    gotoxy(1,wherey());
    cprintf("%c%c  Drive %.1s is NOT a CRAM disk ", 0x07, 0x07, drv);
    return(-1);
  }

 int format_CRAM(long start_sec, long end_sec)
  {
   long dossec;
   char *idptr;

   nsects = 1;
   clear_win(1,1,79,2,0xB0,0);
   gotoxy(40,1);
   cprintf("Starts [%ld]  Ends [%ld]", start_sec, end_sec);
   if(start_sec < reserved_sec)
     return(-1);

   gotoxy(1,1);cprintf("Formatting CRAM Sector ...");
   memset(buffer, 0x00, bpb.bps);
/*   gotoxy(28, wherey()); cprintf("%ld", dossec);*/
   if ((result = putsector(drive, nsects, &start_sec, &bpb, &dpt, buffer)) != 0)
     return(-result);
   memset(buffer, '€',bpb.bps);
   for (dossec = start_sec + 1; dossec < end_sec; dossec + + )
    {
     gotoxy(30, wherey()); cprintf("%ld", dossec);
     if ((result = putsector(drive, nsects, &dossec, &bpb, &dpt, buffer)) != 0)
            return(-result);
    }
   gotoxy(5,wherey() + 2);
   cprintf("%c%c  Drive %.1s is Ready as a CRAM disk ", 0x07, 0x07, drv);
   return(0);
 }


int checkdisk(void)
{
  textcolor(7);
  textbackground(1);
  cprintf("\n\r nspf=%d nfats=%d nres_sec=%d nroot_dir=%d bps=%d ",
             bpb.nspf, bpb.nfats, bpb.nres_sec, bpb.nroot_dir, bpb.bps);
  reserved_sec = ( (DWORD)(bpb.nspf*bpb.nfats) + (DWORD)bpb.nres_sec
                        + (DWORD)((bpb.nroot_dir *32)/bpb.bps));
   if(DEBUG)
   {
     cprintf("Last Result = %ld", result);
     cprintf("\n\rLargest set of free consecutive Clusters Begins: %lu   "
             "Ends %lu  size %lu \n\rCRAM will be loaded at Cluster %lu "
             "sector %lu\n\r Above Reserved Area at %lu",
             f_free.fbegin, f_free.fend, f_free.fsize, f_free.fbegin,
             data_sec, reserved_sec);
     pause(0);
   }
  cprintf("\n\rDrive Information for Drive #%d Letter %.2s",drive, drv);
  result = get_f_name();
  cprintf("\n\rReturn result value = %ld  Sector %lu", result, cluster2sector(result));
  pause(1);
  if(result < = 0)
    result = get_data_sec();
  else
    {
     f_free.fbegin = result;
     f_free.fsize = clust_req;
    }
  clrscr();
  if (result < = 0)
    {
    result = get_fatfree(&f_free, clust_req);
    data_sec = cluster2sector((f_free.fbegin));
    }
  else
    data_sec = result;

  if(DEBUG)
    {
     cprintf("Last Result = %ld", result);
     cprintf("\n\rLargest set of free consecutive Clusters Begins: %lu   "
             "Ends %lu  size %lu \n\rCRAM will be loaded at Cluster %lu "
             "sector %lu\n\r Above Reserved Area at %lu",
             f_free.fbegin, f_free.fend, f_free.fsize, f_free.fbegin,
             data_sec, reserved_sec);
    pause(0);
    }
 if (result > 0)
 if (f_free.fsize > = clust_req)
    {
          f_free.fsize = clust_req;
          f_free.fend  = f_free.fbegin + f_free.fsize;
          if(data_sec > reserved_sec)
            result = put_f_name(f_free, data_sec);
          if(result > = 0)
          result = put_fat(f_free, data_sec);
    }

 return (0);
}
```

CRAM.C The main functions for the operation of CRAM.

```
 /* CRAM.C ---- Constant Random Access Memory system  */

#include <bios.h>
#include <stdio.h>
#include <dos.h>
#include <dir.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include <time.h>
#include <setjmp.h>
#include <alloc.h>
#include "tsr.h"
#include "cram.h"

#define STACK_SIZE       9728 /*9632*/ /*8192*/
#define VIDEO_SEC_SIZE   0x20  /*Reserve sectors to save 16K video memory*/
#define BYTES_PER_TRACK  4609  /* Total bytes for 9 sectors plus 1 for null byte*/
/*==========================================================*/

extern unsigned    keycode;
extern int         hot_key;       /* keycode for activation */
extern int         shift_key;     /* shift status bits (alt,  ctrl..) */
extern int         user_key_set;
long               chksum = 0x0000;
/*==========================================================*/
unsigned char          multiplex_id;

struct dfree          dskfree;
long                  dskavail,
                                 sysmem;
DWORD                   maxmem  =  0xA000L;
static DWORD            sav_start, sav_end;
int                   skiptrack;
int                   spc;
int                   DEBUG  = 0;
struct DPT              dpt;
struct BPB              bpb;
struct HST              hst;
int                   spt;
int                      cmd;
int                      drive;
int                      head;
long                     track;
int                      sector;
int                      nsects;
BYTE                  buffer[BYTES_PER_TRACK];
BYTE                             swap_save_buf[1853];
char                             lpt_buff[80];
char                             lpt_code[4] = "XXXX";
static BYTE             signote[512];
```

```
long                   result;
char                   drv[3] = "Z ";
long                   maxsectors;
long                   maxsides;
long                   maxtracks;
long                   DOSsec;
unsigned long          data_sec;
unsigned char far       *conv_mem_ptr;
ldiv_t                seg_cal;
ldiv_t                offs_cal;
ldiv_t                seg_off;
unsigned long          seg  = 0x0000,
                               offs = 0x0000;
WORD                    lowmem;
MCB  far                      *mcb;
MCB  far                      *first_mcb;
unsigned long          START_SEG,
                               START_OFF,
                               SEC_SEG;
BYTE                   start_flag=0;
BYTE                   reset_mem;
WORD                    sec_seg1_end =0;
unsigned long          i, j;
static unsigned long    savetime = 0L;
static BYTE                   timelag  = 0x3C;
static time_t          t1, t2;
static  WORD            seg_saved=0;
struct CRAM_HEADER       c_header;

char far               *stack_ptr;
char far               *ptr;
char far               *sig_ptr;
BYTE far                *diskette;
struct  bits_16        mem_flag;  /* Bit fields for flag register */
struct  bits_16        save_flag; /* Bit fields for memory segments to save */
WORD                    word;
BYTE                   state=0x07;
unsigned long           memtop;
static jmp_buf                   cram_env;

struct date            ddate;
struct time            dtime;

extern struct   SREGS     sregs;
extern union    REGS      regs;
extern SCRLINE  far       *scr;

extern MCB far *get_mcb (void);

DWORD                    reserved_sec;
struct  free_fat       f_free;
DWORD                    clust_req =500UL;
int                 tsr_already_active   = 0; /* true if TSR active */
int                 popup_while_dos_busy = 0; /* true if hot key hit while dos busy */
int                 int_28_in_progress   = 0; /* true if INT 28 in progress */
int                 unsafe_flag          = 0; /* true if INT 13 in progress */
char                 c_id[2][5] = {"RABB", "WGM_"};

int                 int_1C_in_progress=0;
int                 keys_punched;
int                 startup = 0;
WORD                    saveloop = 0;
char  far *ptr1066 = MK_FP(0x106E,0x00);
static int onepass = 0;
/*==========================================================*/

extern void gotoXY(int x, int y);
extern void clrEol(void);

extern char  far *  swap_ptr;  /* pointer to dos swap area */
extern char  far *  swap_save; /* pointer to our local save area */
unsigned                 ss_save; /* slot for stack segment register */
unsigned                 sp_save; /* slot for stack pointer register */
/* Saved areas for old interrupt pointers */
extern     INTVECT old_int8, old_int9, old_int10, old_int13, old_int1b;
extern     INTVECT old_int23, old_int24;
extern     INTVECT old_int2a;
               INTVECT old_int1C, old_int25, old_int26;
extern     int  dos_critical;     /* used by DOSSWAP.C */

extern     void far set_stack(void);
extern     void far restore_stack(void);

/*==============================================================*/
void        screen_display(int x, int y, char *s, int attrib);
long        c_mem_checksum(WORD seg_start, WORD offs_start, DWORD mem_size);

extern  void  get_time_date(void);
extern int   LPT(char *s);

extern void interrupt far new_int13(void);  /* in TSRUTIL.ASM */
extern void interrupt far new_int10(void);  /* in TSRUTIL.ASM */
extern void interrupt far new_int25(void);  /* in TSRUTIL.ASM */
extern void interrupt far new_int26(void);  /* in TSRUTIL.ASM */
extern void interrupt far new_int2a();
extern void interrupt far new_int24();

void interrupt (*old_int28)();
void interrupt (*old_intFC)();
void interrupt (*gen_intFC)();

int       put_stack(void);
int       get_stack(void);
int           set_MCB(WORD psp);
int           save_MCB(void);
int       capture_mem(void);
int       save_video(void);
int       reset_video(void);
int       save_interrupt(void);
int       reset_interrupt(void);
int       get_header(void);
int       put_header(void);
int       put_memory(void);
int       reset_memory(void);
int       push_mem(void);

void interrupt far mem_reset();
void        mem_save();
void interrupt far new_int8();
void interrupt far new_int9();
void interrupt far new_int28();
void interrupt far new_int1C();

/**********
* DOS IDLE INTERRUPT HANDLER
**********/
void interrupt far new_int28()
{
   int_28_in_progress+ +;

   if (/*popup_while_dos_busy &&*/ !dos_critical
          && !tsr_already_active && !unsafe_flag)
   {
           tsr_already_active = 2;
           mem_save();
           tsr_already_active = 0;
   }
   int_28_in_progress--;
   (* old_int28)();
}

/*********
* TIMER INTERRUPT HANDLER
*********/
void interrupt far new_int8()
{
   if (!tsr_already_active/* && popup_while_dos_busy*/ &&
          !dos_critical && !unsafe_flag/* && !reset_mem*/)
    {
          popup_while_dos_busy = 0;
          tsr_already_active = 8;
          (*old_int8)(); /* process timer tick */
          enable(); /* turn interrupts back on */
          mem_save();
          tsr_already_active = 0;
    }
   else
          (*old_int8)(); /* process timer tick */
}


/**********
* KEYBOARD INTERRUPT HANDLER
**********/
void interrupt far new_int9()
{

/*  if (!tsr_already_active && popup_while_dos_busy &&
          !dos_critical && !unsafe_flag && !reset_mem)
  {*/
   BYTE  s_key=0;
   enable();
   keycode = inp(KEYBOARD_PORT);
   s_key  = bioskey(KEYBRD_SHIFTSTATUS);
   if( (s_key & (ALT_KEY|CTRL_KEY)) = = (ALT_KEY|CTRL_KEY) && keycode= =DELKEY)
          {
            while(bioskey(KEYBRD_READY)) bioskey(KEYBRD_READ);
```

```
             screen_display(10, 24,"\n\r WILL NOT RE-BOOT \n\r", WHITE+(RED<<4)+BLINK);
             outp(0x61, (inp(0x61)|0x80));
             (*old_int9)();
             goto kyb_ret;
           }
   if( (s_key & (CTRL_KEY|LEFT_SHIFT))
             == (CTRL_KEY|LEFT_SHIFT) && keycode==FIVEKEY)
           {
             if (       !dos_critical && !unsafe_flag && !reset_mem)
               {
                popup_while_dos_busy = 0;
                (*old_int9)();    /* send key to old int routine */
                tsr_already_active = 9;
                keys_punched =0;
/*              enable(); */
                reset_mem = 2;
/*              mem_save();
                (*gen_intFC)();*/
                enable();
                tsr_already_active = 0;
               }
             else
               {
                popup_while_dos_busy = 1;
                (* old_int9)();
               }
           }
 if (!tsr_already_active/* && popup_while_dos_busy*/ &&
          !dos_critical && !unsafe_flag/* && !reset_mem*/)
   {
     if (keycode != hot_key)
            {
            if (!unsafe_flag)
              {
               popup_while_dos_busy = 0;
               (*old_int9)();    /* send key to old int routine */
               if(keycode) keys_punched++;
               if (keys_punched >= (2 * KEY_MAX))
                     {
                       tsr_already_active = 9;
                       keys_punched =0;
                       savetime = timelag;
                       enable();
                       mem_save();
                       enable();
                       tsr_already_active = 0;
                     }
              }
            goto kyb_ret;
            }
          else
            (* old_int9)();

    if ((s_key & shift_key) == shift_key)
          {
            if (!unsafe_flag)
             {
              popup_while_dos_busy = 0;
              (*old_int9)();    /* send key to old int routine */
```

```
                tsr_already_active = 9;
                keys_punched =0;
                savetime = timelag;
                enable();
                reset_mem = 1;
                mem_save();
                enable();
                tsr_already_active = 0;
               }
              else
               {
                popup_while_dos_busy = 1;
                (* old_int9)();
               }
             }
        else
             (* old_int9)();
    }
   else
     (* old_int9)();
  kyb_ret:
  enable();
}

long c_mem_checksum(WORD seg_start, WORD offs_start, DWORD mem_size)
{
  long sum=0x0000L, cnt=0, k, l, m=0x0000L, n;
  long i, seg, offs;
  DWORD far *s_mem;
  ldiv_t   cal;

  cal = ldiv(mem_size, 0x10000);
  k = cal.quot;
  l = cal.rem;
  for (seg=seg_start; seg<=(k*0x1000); seg+=0x1000)
   {
    cnt =0;

    if (k != 0)
     {
        n = 0x10000;
        offs = 0;
     }
    else
     {
        n = mem_size;
        offs = offs_start;
     }
    s_mem = MK_FP(seg, offs);
    while(cnt < n)
     {
            if(DEBUG)
                {
                 put_hex(*s_mem);
                 put_str(" ");
                }
        sum += *(s_mem++);
        m ^= *(s_mem++);
        cnt+=sizeof(s_mem);
```

```
      }
     if(DEBUG)
       {
              gotoxy(1,1);
              printf("%Fp",s_mem);
       }
    }
    cnt = 0;
    offs = 0;
    if (k != 0)
      while(cnt < l)
       {
              s_mem = MK_FP(seg, offs);
              if(DEBUG)
                {
                  gotoxy(1,1);
                  printf("%Fp",s_mem);
                }
              m ^= *(s_mem++);
              sum += *(s_mem++);
              put_hex(*(s_mem));
              cnt+=sizeof(s_mem);
       }
  if(DEBUG)
    {
     gotoxy(60,i);
     printf("%Fp",s_mem);
    }
  sum = (sum >> 8);  /* Put value in lower 16 bits (upper 2 bytes)  */
  m   = (m << 8);    /* Put value in upper 16 bits (2 bytes)       */
  return(sum|m);     /* Ored sum with m to get a unique CHECKSUM   */
}


void interrupt mem_reset(void)
{

  reset_memory();
}

void screen_display(int x, int y, char *s, int attrib)
{
 BYTE index=0;

  scr[0] [2].s_char = popup_while_dos_busy + 0x30;
  scr[0] [3].s_char = tsr_already_active + 0x30;
  scr[0] [4].s_char = unsafe_flag + 0x30;
  scr[0] [5].s_char = int_28_in_progress+ 0x30;
  scr[0] [6].s_char = (savetime>10 ? savetime/10 : savetime) + 0x30;
  scr[0] [6].s_attr = ((scr[0] [0].s_attr >> 4)
    + (scr[0] [0].s_attr << 4)) & (0x77 +(savetime/10));
  scr[0] [7].s_char = startup + 0x30;
  scr[0] [8].s_char = dos_critical + 0x30;
  scr[0] [9].s_char = ((keys_punched > 10) ? keys_punched/10 :keys_punched) + 0x30;
  scr[0] [9].s_attr = ((scr[0] [0].s_attr >> 4)
  + (scr[0] [0].s_attr << 4)) & (0x77+(keys_punched/10));
  scr[0] [10].s_char = int_1C_in_progress + 0x30;
  word = c_header.FLAGS;
```

```
     memcpy(&mem_flag ,&word, 2);
     scr[0] [12].s_char = mem_flag.lobits.bit7 + 0x30;
     scr[0] [13].s_char = mem_flag.lobits.bit6 + 0x30;
     scr[0] [14].s_char = mem_flag.lobits.bit5 + 0x30;
     scr[0] [15].s_char = mem_flag.lobits.bit4 + 0x30;
     scr[0] [16].s_char = mem_flag.lobits.bit3 + 0x30;
     scr[0] [17].s_char = mem_flag.lobits.bit2 + 0x30;
     scr[0] [18].s_char = mem_flag.lobits.bit1 + 0x30;
     scr[0] [19].s_char = mem_flag.lobits.bit0 + 0x30;

     scr[0] [60].s_char = save_flag.hibits.bit7 + state;
     scr[0] [61].s_char = save_flag.hibits.bit6 + state;
     scr[0] [62].s_char = save_flag.hibits.bit5 + state;
     scr[0] [63].s_char = save_flag.hibits.bit4 + state;
     scr[0] [64].s_char = save_flag.hibits.bit3 + state;
     scr[0] [65].s_char = save_flag.hibits.bit2 + state;
     scr[0] [66].s_char = 0xCE;
     scr[0] [67].s_char = save_flag.hibits.bit1 + state;
     scr[0] [68].s_char = save_flag.hibits.bit0 + state;
     scr[0] [69].s_char = save_flag.lobits.bit7 + state;
     scr[0] [70].s_char = save_flag.lobits.bit6 + state;
     scr[0] [71].s_char = save_flag.lobits.bit5 + state;
     scr[0] [72].s_char = save_flag.lobits.bit4 + state;
     scr[0] [73].s_char = save_flag.lobits.bit3 + state;
     scr[0] [74].s_char = save_flag.lobits.bit2 + state;
     scr[0] [75].s_char = save_flag.lobits.bit1 + state;
     scr[0] [76].s_char = save_flag.lobits.bit0 + state;
     index = x;
     while (*s)
        if (index > 80)
              {
               index = x;
               if (y != 25) y+ +;
              }
        else
              {
               scr[y][index].s_attr = attrib;
           scr[y][index+ +].s_char = *s+ +;
              }
}

void interrupt far new_int1C()
 {
    int_1C_in_progress = 1;
   if (!tsr_already_active/* && popup_while_dos_busy*/ &&
           !dos_critical && !unsafe_flag /*&& !reset_mem*/)
    {
           (* old_int1C)();
           tsr_already_active = 1;
/*         enable();*/
           screen_display(10, 24, "",(14 + (RED < < 4)) );
           chksum = c_mem_checksum(START_SEG, 0x0000, 0x1000);
           if((c_header.save_seg_flag[(BYTE)(START_SEG/0x1000)] = chksum) = =
             c_header.seg_checksum[(BYTE)(START_SEG/0x1000)] )
                     c_header.seg_checksum[(BYTE)(START_SEG/0x1000)]=chksum;
/*         mem_save();*/
           enable();
      tsr_already_active = 0;
   }
```

```
      int_1C_in_progress = 0;

  }


 void mem_save(void)
  {
   scr[0] [0].s_char = 0x00EE;
   scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4)
    + (scr[0] [0].s_attr << 4)) & 0x77;
/*   if (onepass == 0)
   {
   scr[1] [75].s_char = *ptr1066;
    scr[1] [76].s_char = *(ptr1066+1);
    scr[1] [77].s_char = *(ptr1066+2);
    onepass++;
    }*/
   screen_display(10, 24, "",(14 + (RED << 4)) );

   set_stack();
   c_header.PSP = GetPSP();
   /*  Save the DTA for the on going process  */
   regs.h.ah = GET_DTA;
   intdosx(&regs, &regs, &sregs);
   c_header.dta_seg = sregs.es;
   c_header.dta_off = regs.x.bx;

   *sig_ptr = 'R'; *(sig_ptr + 1) = 'B';

   if (reset_mem == 2)
      {
          saveloop=0;
          maxmem = sav_end;
          START_SEG = sav_start;
          savetime = 0;
          if (_argc > 3)
            timelag =(BYTE)atoi(_argv[3]);
          if(timelag == 0)  timelag = 0x3C;

          get_header();

          SetPSP(_psp);

    /* Setup CRAM's DTA    */

          regs.h.ah = SET_DTA;
          sregs.es = _psp;
          regs.x.dx = 0x80;
          intdosx(&regs, &regs, &sregs);

          DOSsec = c_header.mem_sec;
          c_header.curr_mem_sec = DOSsec;
          start_flag = 0;
          reset_mem = 0;
          if (!int_1C_in_progress)  time(&t1);
/*        strcpy(lpt_buf,
  "<<==========  BEGIN  =========  RESTORATION REPORT   ====================>>");
          LPT(lpt_buf);
```

```
                lpt_code[0] = 0x0A;
                lpt_code[1] = LINEFEED;
                lpt_code[2] = 0x00;
                LPT(lpt_code); */

                (*gen_intFC)();
                START_SEG = sav_start;
                maxmem    = sav_start + 0x1000;
                DOSsec    = c_header.mem_sec;
                c_header.curr_mem_sec = DOSsec;
                start_flag  =  0;
                sec_seg1_end = 0;
                capture_mem();
                put_header();
                if (!int_1C_in_progress)  time(&t1);
                putstr("End of Memory Installation");
                reset_mem = 0;
                goto contd;
      }
  if (reset_mem  == 1 )
     {
               SaveDosSwap();
               start_flag = 0;
               SetPSP(c_header.cram_psp);
                /* Setup CRAM's DTA    */
               regs.h.ah = SET_DTA;
               sregs.es=c_header.cram_dta_seg;
               regs.x.dx = c_header.cram_dta_off;
               intdosx(&regs, &regs, &sregs);
               GetExtErr(&c_header.ErrInfo);
               capture_mem();
               put_header();
               push_mem();
               get_header();
               start_flag =0;
               reset_mem = 0;
               get_time_date();
               capture_mem();
               put_header();
               if (!int_1C_in_progress) time(&t1);
               savetime = 0;
               if (_argc > 3)
                 timelag = (BYTE)atoi(_argv[3]);
               if (timelag == 0)  timelag = 0x3C;
               RestoreDosSwap();
               SetExtErr(&c_header.ErrInfo);
               goto contd;
    }

 if (savetime >= timelag && !reset_mem)
  {
   save_DSA();
   SaveDosSwap();
   SetPSP(c_header.cram_psp);
           /* Setup CRAM's DTA    */
   regs.h.ah = SET_DTA;
   sregs.es=c_header.cram_dta_seg;
   regs.x.dx = c_header.cram_dta_off;
   intdosx(&regs, &regs, &sregs);
  GetExtErr(&c_header.ErrInfo);
  if( ! mcb_chk(get_mcb()))
   {
        putstr(" <<-oo->>======== ERROR in MCB Chain ========<<-oo->>");
        result = set_MCB(0);
        scr[0] [79].s_char = result  + 0x30;
   }
  else
   {
         result = save_MCB();
         scr[0] [79].s_char = result  + 0x30;
   }
  get_time_date();
  DOSsec = c_header.curr_mem_sec;
  if ( (START_SEG >= sav_end) || (maxmem > sav_end))
        {
          START_SEG = sav_start;
          maxmem    = sav_start + 0x1000;
          DOSsec    = c_header.mem_sec;
          c_header.curr_mem_sec = DOSsec;
          start_flag  =  0;
          /*c_header.sec_segl_end = sec_segl_end = 0;*/
          sec_segl_end = 0;
        }
          /*     c_header.clusters = sector2cluster((c_header.dsa_sec+4));*/
 capture_mem();
 put_header();
 save_interrupt();
 if(DEBUG)
  {
        put_hex(START_SEG);put_str(COLON);
        put_hex(maxmem);put_str(SPACE);
        put_hex(DOSsec);put_str(SPACE);
        put_str(LDGT);
        put_hex(sec_segl_end);put_str(RDGT);
  }

 if(put_memory() != 0)
         {
         savetime = 0;
         DOSsec = c_header.mem_sec;
         c_header.curr_mem_sec = DOSsec;
         start_flag = 0;
         putstr("\n\rERROR saving memory::save_mem ");
         if (!int_1C_in_progress) time(&t1);
         }
else
       {
         c_header.curr_mem_sec = DOSsec;
         save_video();
         DOSsec += bpb.nspt;
         savetime = 0;
         put_stack();
         START_SEG += 0x1000;
         maxmem  = sav_end; /* += 0x1000;*/
         capture_mem();
         put_header();
       }
if (!int_1C_in_progress)  time(&t1);
```

```
       SetExtErr(&c_header.ErrInfo);
       RestoreDosSwap();
      } /* end of if savetime = timelag */

    contd:
    restore_stack();
    setjmp(cram_env);
    if (!int_1C_in_progress )
      {
       time(&t2);
       savetime = (long)(t2 - t1);
      }
     else
      if(reset_mem)   savetime++;

    SetPSP (c_header.PSP);
    regs.h.ah = SET_DTA;
    sregs.ds  = c_header.dta_seg;
    regs.x.dx = c_header.dta_off;
    intdosx(&regs, &regs, &sregs);

 /*  (*old_int28)();*/
  }

 int capture_mem(void)
   {
    unsigned k=0, m, save_k;
    scr[0] [0].s_char = 0x00EF;
    scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;

    disable();
    c_header.AX = _AX;
    c_header.BX = _BX;
    c_header.CX = _CX;
    c_header.DX = _DX;
    c_header.CS = _CS;
    c_header.DS = _DS;
    c_header.ES = _ES;
    c_header.SS = _SS;
    c_header.BP = _BP;
    c_header.DI = _DI;
    c_header.SI = _SI;
    c_header.SP = _SP;
    c_header.FLAGS = _FLAGS;
 /*   c_header.PSP = GetPSP();*/ /*_psp;*/
    enable();
    c_header.int_checksum = (DWORD)c_mem_checksum(0x0000, 0x0000, 0x0400);
    c_header.checksum = 0;
    for (m=0; m<k; m++) c_header.checksum = signote[m];
    memcpy(&signote[k],SIG,strlen(SIG) );
    k = strlen(SIG);
    signote[k-1] = 0x1A;
    memcpy(&signote[k],&dpt,sizeof(dpt) );
    k += sizeof(dpt);
    memcpy(&signote[k],&bpb,sizeof(bpb) );
    k+= sizeof(bpb);
    save_k = k;
    memcpy(&signote[k],&c_header,sizeof(c_header) );
    k += sizeof(c_header);
    memcpy(&signote[k],&cram_env, sizeof(cram_env));
    k += sizeof(cram_env);
    for (m=0; m<k; m++) c_header.checksum += signote[m];
    memcpy(&signote[save_k],&c_header,sizeof(c_header) );
    memcpy(&signote[508],&c_id[1],4);
    memcpy(&signote[506],&k,2);
    signote[505] = 0x01;
    return(k);
  }


 int get_header(void)
  {
   long k=0L;

   nsects = 1;
   scr[0] [0].s_char = 0x00F0;
   scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
   DOSsec = c_header.data_sec;
   stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
   if ((result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
             hst.SECTOR, nsects, signote)) != 0)
   return(result);
   k = strlen(SIG) + sizeof(dpt) + sizeof(bpb);
   memcpy(&c_header, &signote[k], sizeof(c_header) );
   k += sizeof(c_header);
   memcpy(&cram_env, &signote[k], sizeof(cram_env));
   k += sizeof(cram_env);
   return(k);
  }

int put_header(void)
{
 nsects = 1;
 scr[0] [0].s_char = 0x00EC;
 scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
 DOSsec = c_header.data_sec;
 if (DOSsec < reserved_sec) return(-1);
 stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
 result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
           hst.SECTOR, nsects, signote);
 return(result);
}

int put_stack(void)
{
 int i;

 nsects = 1;
 scr[0] [0].s_char = 0x001F;
 scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
 DOSsec = c_header.stack_sec;
 if (DOSsec < reserved_sec) return(-1);
 for (i=0; i<(STACK_SIZE/bpb.bps); i++)
  {
   disable();
   movedata(FP_SEG(stack_ptr), FP_OFF(stack_ptr)+(i*bpb.bps*nsects),
            FP_SEG(buffer), FP_OFF(buffer), (bpb.bps * nsects));
```

```
      enable();
      stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
      result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
              hst.SECTOR, nsects, buffer);
      DOSsec++;
  }
  return(result);
}

int get_stack(void)
{
  int i;

  nsects = 1;
  scr[0] [0].s_char = 0x001E;
  scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
  DOSsec = c_header.stack_sec;
  for (i=0; i<(STACK_SIZE/bpb.bps); i++)
   {
     if (DOSsec < reserved_sec) return(-1);
     stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
     result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
             hst.SECTOR, nsects, buffer);
     disable();
     movedata(FP_SEG(buffer), FP_OFF(buffer), FP_SEG(stack_ptr),
                 FP_OFF(stack_ptr)+(i*bpb.bps*nsects), (bpb.bps * nsects));
     enable();
     DOSsec++;
   }
  return(result);
}


int save_video(void)
 {
  nsects = 8;
  state  = 0x09;
  DOSsec = c_header.video_sec;
  if (DOSsec < reserved_sec) return(-1);
  movedata(FP_SEG(scr)/*0xB800*/, 0x0000, FP_SEG(buffer), FP_OFF(buffer), 0x1000);
  stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
  if ((result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
    hst.SECTOR, nsects, buffer)) != 0)
                  return(result);
  seg_saved |= (1 << (0xB000/0x1000));
  memcpy(&save_flag, &seg_saved,2);
  c_header.sector_in_cram[(BYTE)(0xB000/0x1000)]=DOSsec;
  c_header.size_of_seg[(BYTE)(0xB000/0x1000)] =
             (DOSsec+VIDEO_SEC_SIZE) - c_header.sector_in_cram[(BYTE)(0xB000/0x1000)];
  nsects = 1;
  return(0);
}

int reset_video(void)
{
  nsects = 8;
  DOSsec = c_header.video_sec;
  stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
  if ((result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
     hst.SECTOR, nsects, buffer)) != 0)
                        return(result);
   movedata(FP_SEG(buffer), FP_OFF(buffer), FP_SEG(scr)/*0xB800*/, 0x0000, 0x1000);
   nsects = 1;
   return(0);
 }


int save_interrupt(void)
 {
   ldiv_t    rem_sec;

   nsects = 3;
   state  = 0x09;
   scr[0] [0].s_char = 0x00ED;
   scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
   disable();
   movedata(0x0000, 0x0000, FP_SEG(buffer), FP_OFF(buffer), 0x600);
   enable();
   DOSsec = c_header.data_sec+1;
   rem_sec = ldiv(DOSsec, bpb.nspt);
   if (rem_sec.rem < 3)
      DOSsec += (bpb.nspt - rem_sec.rem);
   if (DOSsec < reserved_sec) return(-1);
   stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
   result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
            hst.SECTOR, nsects, buffer);
   seg_saved |= (1 << (0x0000/0x1000));;
   memcpy(&save_flag, &seg_saved,2);
   c_header.sector_in_cram[(BYTE)(0x0000/0x1000)]=DOSsec;
   c_header.size_of_seg[(BYTE)(0x0000/0x1000)] =
                (DOSsec + nsects) - c_header.sector_in_cram[(BYTE)(0x0000/0x1000)];
     return(result);
}

int reset_interrupt(void)
{
  int x;

  nsects = 3;
  scr[0] [0].s_char = 0x00EA;
  scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
  DOSsec = c_header.int_sec;
  stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
  if ((result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
    hst.SECTOR, nsects, buffer)) != 0)
                        return(result);
  while(*diskette != 0x00); /* wait for disk drive to stop spinning */
  if(DEBUG)
   {
          put_str("{");put_hex(DOSsec);put_str("}");
          for (x=0; x<0x600; x++)
             {
               put_str(" ");put_bit(buffer[x]);
             }
             LPT(buffer);
   }
  disable();
  movedata(FP_SEG(buffer), FP_OFF(buffer), 0x000, 0x000, 0x600);
```

```
   enable();
   return(0);
  }


  int put_memory(void)
  {



   spt     = bpb.nspt;
   nsects  = 1;
   track   = 1;
   head    = 0;
   state   = 0x09;
   scr[0] [0].s_char = 0x00E8;
   scr[0] [0].s_attr = ((scr[0] [0].s_attr > > 4)
       + (scr[0] [0].s_attr < < 4)) & 0x77;
   if(DEBUG)
    {
     put_hex(START_SEG);put_str(":");
     put_hex(maxmem);put_str(" { = ");
     put_hex(DOSsec);put_str(" = } ");
    }
   for (seg=START_SEG; seg < maxmem; seg+ =4096)
    {
     i = 0L;
     scr[0] [0].s_char = 0x00D8;
     scr[0] [0].s_attr = ((scr[0] [0].s_attr > > 4)
            + (scr[0] [0].s_attr < < 4)) & 0x77;
     if (start_flag = = 0)
           {
             offs = START_OFF;
             DOSsec = (c_header.mem_sec + ((WORD)((seg - sav_start) > > 12)
                 * (0x10000/0x200)));
           }
     else
           {
            offs = 0;
            DOSsec = (long)(c_header.mem_sec + (((seg - sav_start) > > 12)
               * (0x10000/0x200)) - ((0x10000/0x200) - sec_seg1_end));
           }
     c_header.sector_in_cram[(BYTE)(seg/0x1000)] =DOSsec;
     screen_display(10, 24, "",(14 + (RED < < 4)) );
           chksum = c_mem_checksum(START_SEG, 0x0000, 0x1000);
           if((c_header.save_seg_flag[(BYTE)(START_SEG/0x1000)] = chksum) = =
             c_header.seg_checksum[(BYTE)(START_SEG/0x1000)] )
                       c_header.seg_checksum[(BYTE)(START_SEG/0x1000)] =chksum;
/*     put_hex(chksum);put_str(" ");*/
     /* calculate the number of sectors per 64k segment */

     SEC_SEG   = (WORD)((0x10000  - offs) / (bpb.bps));
     if(DEBUG)
      {
           put_hex(seg);put_str(":");
           put_hex(offs);put_str(" { = ");
           put_hex(DOSsec);put_str(" = } ");
           put_str(" [");put_hex(SEC_SEG);put_str("] ");
      }
```

```
       if (DOSsec < reserved_sec) return(-1);

       for (j=0; j<SEC_SEG; j++)
             {
               movedata(seg, (offs+i), FP_SEG(buffer), FP_OFF(buffer),
                        (bpb.bps * nsects));
               stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
               if ((result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
                        hst.SECTOR, nsects, buffer)) != 0)
                 {
                      put_str(" [result=");put_hex(result);put_str("] ");
                      put_str(" <TRACK=");put_hex(hst.TRACK);put_str("> ");
                      put_str(" <HEAD    =");put_hex(hst.HEAD);put_str("> ");
                      put_str(" <SECTOR =");put_hex(hst.SECTOR);put_str("> ");
                      put_str(" <DOS SEC    =");put_hex(DOSsec);put_str("> ");
                      return(result);
                 }
              seg_saved |= (1 << (seg/0x1000));
              memcpy(&save_flag, &seg_saved,2);
              DOSsec ++;
              i += bpb.bps;
             }
      c_header.size_of_seg[(BYTE)(seg/0x1000)] =
      DOSsec - c_header.sector_in_cram[(BYTE)(seg/0x1000)];

      if (start_flag == 0)
            {
             c_header.sec_segl_end = DOSsec;
             sec_segl_end = (DOSsec - c_header.mem_sec);
             start_flag = 1;
            }
        c_header.curr_mem_sec = DOSsec;
    }
  if (seg >= sav_end)
    {
     seg_saved = 0;
     memcpy(&save_flag, &seg_saved,2);
     state = 0x07;
    }
  return(0);
}

int reset_memory(void)

{
 long              filler=0L;

 scr[0] [0].s_char = 0x00EB;
 scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
 spt = bpb.nspt;
 conv_mem_ptr = MK_FP(seg, offs);

 nsects = 1;
 track = 1;
 head  = 0;
 sec_segl_end = 0;
 sec_segl_end = (c_header.sec_segl_end - c_header.mem_sec);
 DOSsec = (c_header.mem_sec + ((WORD)((START_SEG - sav_start) >> 12)
                                  * (0x10000/0x200)) - sec_segl_end);
 if(DEBUG)
  {
    put_str("+ + + + ");put_hex(c_header.mem_sec);
    put_str("+ + + + ");put_hex(DOSsec);
    put_str("+ + + + ");put_hex(c_header.curr_mem_sec);
    put_str("+ + + + ");put_hex(c_header.sec_segl_end);putstr("//");
    put_str("+ + + + ");put_hex(sec_segl_end);putstr("//");
    put_hex(START_SEG);put_str(":"); put_hex(maxmem);put_str(" {");
    put_hex(DOSsec);put_str("} \n\r");
  }

 for (seg=START_SEG; seg < maxmem; seg+ =4096)
  {
   i = 0L;

   scr[0] [0].s_char = 0x009D;
   scr[0] [0].s_attr = ((scr[0] [0].s_attr > > 4) + (scr[0] [0].s_attr < < 4)) & 0x77;
/*
   ltoa(seg, lpt_buf, 16);
   LPT(lpt_buf);*/
/*    chksum = c_mem_checksum(seg, 0x0000, 0x1000);
   c_header.seg_checksum[(BYTE)(seg/0x1000)]=chksum;*/
/*    put_hex(chksum);put_str(" ");*/
/*    ltoa(chksum, lpt_buf, 16);
   LPT(lpt_buf); LPT(lpt_code);
*/
   if (start_flag = = 0)
            {
             DOSsec = (c_header.mem_sec + ((WORD)((seg - sav_start) > > 12)
               * (0x10000/0x200)));
             START_OFF - = c_header.offs_filler;
             start_flag = 1;
             offs = START_OFF;
            }
          else
            {
             offs = 0;
             DOSsec = (long)(c_header.mem_sec + (((seg - sav_start) > > 12)
              * (0x10000/0x200)) - ((0x10000/0x200) - sec_segl_end));
            }

 /* calculate the number of sectors per 64k segment */
          SEC_SEG   = (WORD)((0x10000 - offs) / (bpb.bps));
          conv_mem_ptr = MK_FP(seg, offs);
          if(DEBUG)
            {
             gotoXY(1,wherey()); clrEol();
             put_hex(seg);put_str(":"); put_hex(offs+i);put_str(" {");
             put_hex(DOSsec);putstr("} ");
            }
          for (j=0; j<SEC_SEG; j+ +)
            {
             scr[0] [0].s_char = 0x0087;
             scr[0] [0].s_attr = ((scr[0] [0].s_attr > > 4) + (scr[0] [0].s_attr < < 4)) & 0x77;
             conv_mem_ptr = MK_FP(seg, offs+i);

             stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
             if ((result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
                          hst.SECTOR, nsects, buffer)) != 0)
```

```
                    return(result);
                if(filler == 0)
                    {
                     if (DEBUG)
                      {
                           put_str("<<"); put_str((char far *)(conv_mem_ptr+c_header.offs_filler));
                           putstr(" >> ");
                      }
                     movedata(FP_SEG(&buffer[c_header.offs_filler]),
                              FP_OFF(&buffer[c_header.offs_filler]),seg,
                              (offs+i+c_header.offs_filler),
                              ((bpb.bps * nsects)-c_header.offs_filler));
                    if(DEBUG)
                      {
                           put_str("<<");put_hex((offs+i+c_header.offs_filler));
                           put_str(" >>");put_str("<<"); put_str((char far *)(conv_mem_ptr+c_header.offs_filler));
                           putstr(" >> ");put_str("<<"); put_str((char *)buffer[c_header.offs_filler]);putstr(" >> ");
                      }
                    conv_mem_ptr = MK_FP(seg, offs+i+c_header.offs_filler);
                    j=0L;
                    filler = 1;
                   }
                else
                   {
                    if (seg == 0x2000 && DEBUG && ((offs+i) > 0x7E00));
                    else
                    movedata(FP_SEG(&buffer[0]), FP_OFF(&buffer[0]),seg, (offs+i),
                             (bpb.bps * nsects));
                   }
                DOSsec++;
                if(DEBUG == 1)
                 {
                   gotoxy(1,wherey());
                   clreol();
                   printf("%Fp %lu",conv_mem_ptr, j);
                 }
                i += bpb.bps;
              }
 }

/*  lpt_code[0] = 0x0C;
  LPT(lpt_code);*/
/*  put_header();*/
/*\\\\\\\\\\\\\\\\\\\\\\\\\ RESET REGISTERS AND DSA /////////////////////*/

  reset_video();
  scr[0] [0].s_char = 0x00EB;
  DOSsec += bpb.nspt;
  while(*diskette != 0x00); /* wait for disk drive to stop spinning */

  disable();
  _AX = c_header.AX ;
  _BX = c_header.BX ;
  _CX = c_header.CX ;
  _DX = c_header.DX ;
  _ES = c_header.ES ;
  _DS = c_header.DS ;
  _CS = c_header.CS ;
  _SS = c_header.SS ;
```

```
    _SP = c_header.SP ;
    _DI = c_header.DI ;
    _SI = c_header.SI ;
    _BP = c_header.BP ;
    _FLAGS = c_header.FLAGS ;
    enable();

    SetExtErr(&c_header.ErrInfo);
   /* Restoring DTA for saved process in CRAM */
  /* regs.h.ah  = SET_DTA;
    regs.x.dx  = c_header.dta_off;
    sregs.ds   = c_header.dta_seg;
    intdosx(&regs, &regs, &sregs);
    SetPSP(c_header.PSP);
  */

 ret:
    while(*diskette != 0x00); /* wait for disk drive to stop spinning */
    reset_interrupt();
    delay(500);         /* wait for interrupt if called */
    while(*diskette != 0x00); /* wait for disk drive to stop spinning */
    reset_DSA();
    scr[0] [0].s_char = 0x00EB;
    while(*diskette != 0x00); /* wait for disk drive to stop spinning */
    savetime=0;
    reset_mem = 0;
    START_SEG = sav_start;
    maxmem    = sav_start + 0x1000;
    DOSsec    = c_header.mem_sec;
    c_header.curr_mem_sec = DOSsec;
    start_flag  =  0;
    sec_seg1_end = 0;
    time(&t1);
      if( ! mcb_chk(get_mcb()))
        {
            putstr(" <<-oo->>======== ERROR in MCB Chain ========<<-oo->>");
            putstr(" <<-oo->>Fixing= ERROR in MCB Chain =========<<-oo->>");
            result = set_MCB(0);
            scr[0] [79].s_char = result  + 0x30;
        }
   get_stack();
   longjmp(cram_env,1);
   return(result);
}

int push_mem(void)
{
   lowmem = low_mem(first_mcb);     /* find address where programs can be loaded */
   seg_off = ldiv(lowmem, 0x1000L); /* calculate the segment:offset pair */
   START_OFF = (seg_off.rem << 4);
   /* Make sure memory is align with sector size */
   offs_cal = ldiv((START_OFF), (long)bpb.bps);
   START_OFF -= c_header.offs_filler = offs_cal.rem;
   START_SEG = (seg_off.quot << 12);/* using 64k segment boundries */
   c_header.start_addr = lowmem;
   c_header.data_sec = data_sec;
   c_header.curr_mem_sec  = data_sec + 1;
   c_header.sec_seg1_end = 0;
   sec_seg1_end = 0;
```

```
    get_time_date();
    capture_mem();
    save_interrupt();
    c_header.int_sec = DOSsec;
    sav_start = START_SEG;
    maxmem  = sav_end;
          /*     sav_end   = maxmem; */
          /* Skip to next track if now not at first sector in the track*/
      /*    seg_cal = ldiv(DOSsec, bpb.nspt);*/
    DOSsec + =3; /* advance the number of sector for interrupt vector table */
    if(DEBUG)
     {
          put_str("+ + + +");put_hex(c_header.mem_sec);
          put_str("+ + + +");put_hex(DOSsec);putstr("//");
     }

    c_header.mem_sec = DOSsec;
    start_flag = 0;
    if (put_memory() = = 0)
          {
            c_header.video_sec = DOSsec;
            save_video();
            DOSsec + =VIDEO_SEC_SIZE;
            c_header.dsa_sec = DOSsec;
            save_DSA();
            DOSsec+ =4;
            c_header.mcb_sec = DOSsec;
            DOSsec+ +;
            c_header.stack_sec = DOSsec;
            c_header.f_size = (DWORD)((DOSsec + 4 - c_header.data_sec + 1) * 512);
            c_header.clusters = sector2cluster((DWORD)(DOSsec + 4 - c_header.data_sec + 1));
            if(DEBUG)
              {
                    put_str("+ + + +");put_hex(c_header.mem_sec);
                    put_str("+ + + +");put_hex(DOSsec);
                    put_str("+ + + +");put_hex(c_header.curr_mem_sec);
                    put_str("+ + + +");put_hex(c_header.sec_seg1_end);putstr("//");
                    put_str("+ + + +");put_hex(sec_seg1_end);putstr("//");
              }
            capture_mem();
            put_header();
        DOSsec = c_header.curr_mem_sec = c_header.mem_sec;
            /*c_header.sec_seg1_end =*/
            sec_seg1_end = 0;

          }
    else
          {
           put_str("\n\rDISK ERROR saving memory\n\r");
          }
          maxmem = sav_start + 0x1000;
          start_flag = 0;
          return (0);
}

int save_MCB(void)
{
   int r=0;
```

```
      scr[0] [0].s_char = 0x00AF;
      scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4)
       + (scr[0] [0].s_attr << 4)) & 0x77;

      for (i=0; i<500; i++) buffer[i] = 0x007A;
      nsects = 1;
      r = get_mcb_chain(get_mcb(), buffer);
      memcpy(&buffer[500],(const char *)"■MCB_CHAIN■",11);
      DOSsec = c_header.mcb_sec;
      if (DOSsec < reserved_sec) return(-1);
      stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
      if ((result = biosdisk(WRITE, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
              hst.SECTOR, nsects, buffer)) != 0)
       {
             put_str(" [result=");put_hex(result);put_str("] ");
             put_str(" <TRACK=");put_hex(hst.TRACK);put_str("> ");
             put_str(" <HEAD    =");put_hex(hst.HEAD);put_str("> ");
             put_str(" <SECTOR =");put_hex(hst.SECTOR);put_str("> ");
             put_str(" <DOS SEC    =");put_hex(DOSsec);put_str("> ");
             return(result);
       }
      DOSsec ++;
      return(r);
 }

 int set_MCB(WORD psp)
 {
      int r=0;

      scr[0] [0].s_char = 0x00AE;
      scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4)
       + (scr[0] [0].s_attr << 4)) & 0x77;

      nsects = 1;
      DOSsec = c_header.mcb_sec;
      if (DOSsec < reserved_sec) return(-1);
      stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
      if ((result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
              hst.SECTOR, nsects, buffer)) != 0)
       {
             put_str(" [result=");put_hex(result);putstr("] ");
             put_str(" <TRACK=");put_hex(hst.TRACK);putstr("> ");
             put_str(" <HEAD    =");put_hex(hst.HEAD);putstr("> ");
             put_str(" <SECTOR =");put_hex(hst.SECTOR);putstr("> ");
             put_str(" <DOS SEC    =");put_hex(DOSsec);putstr("> ");
             return(result);
       }
      DOSsec ++;
      r = set_mcb_chain(buffer, psp);
      return(r);
 }


 void CRAMEXE03111992(void)
  {
  }

 int CHKEXE(char *RBWGM)
```

```
{
 return((DWORD)(strtol(RBWGM+5, &RBWGM, 16)) ^ biosequip());
}

int main(int argc, char *argv[])
{
 unsigned far *fp;

  nsects    = 1;
  scr = MK_FP((color_adpt() ? 0xB800 : 0xB000), 0x0000);
          scr[0] [0].s_attr = ((scr[0] [0].s_attr >> 4) + (scr[0] [0].s_attr << 4)) & 0x77;
  clear_screen(0xDB);
  window(1,2,80,24);
  clear_win(1,2,79,24,0xB0,(WHITE +(BLUE << 4)));
  drive = get_drive_info(drv);
  reserved_sec = ( (DWORD)(bpb.nspf*bpb.nfats) + (DWORD)bpb.nres_sec
                  + (DWORD)((bpb.nroot_dir *32)/bpb.bps));
  sec_seg1_end = 0;
  result = get_f_name();
  if (result <= 0)
   {
        gotoxy(1,1);
        cprintf("ds:%ld drive [%.1s] CRAM not Installed on disk -->> See manual ",
                result, drv);
        gotoxy(1,22);
        exit(90);
   }
  else
   {
        c_header.start_cluster = result;
        data_sec = cluster2sector(result);
        c_header.data_sec = data_sec;
        DOSsec = data_sec;
        c_header.mem_sec = DOSsec;
   }
/*  c_header.start_cluster = sector2cluster(data_sec);*/
  printf(" \n Data starts at SECTOR %lu {CLUSTER %lu}\n", data_sec,
                                  c_header.start_cluster);

  first_mcb = mcb = (MCB far *)get_mcb();        /* find first MCB     */
  lowmem = low_mem(mcb);    /* find address where programs can be loaded */
  if (lowmem == 0)
   exit(91);

  seg_off = ldiv(lowmem, 0x1000L); /* calculate the segment:offset pair */
  START_OFF = (seg_off.rem << 4);

  /* Make sure memory is align with sector size */

  offs_cal = ldiv((START_OFF), (long)bpb.bps);
  START_OFF -= c_header.offs_filler = offs_cal.rem;
  START_SEG = (seg_off.quot << 12);/* using 64k segment boundries  */

  InitInDos();
  if (InitDosSwap() != 0)
   {
       putstr("ERROR initializing DOS Swappable Area. ");
       exit(92);
   }
```

```
sav_start = START_SEG;
sav_end   = maxmem;
if (argc > 3)
  timelag = (BYTE)atoi(argv[3]);
if(timelag == 0)  timelag = 0x3C;
if(argc < 1)   exit(93);

scr[1] [75].s_char = *ptr1066;
scr[1] [76].s_char = *(ptr1066+1);
scr[1] [77].s_char = *(ptr1066+2);

if(strstr(strupr(argv[1]), "I") != NULL)
 {
  reset_mem = 1;

  c_header.cram_psp = GetPSP();

  /* Save the DTA for CRAM process  */

  regs.h.ah = GET_DTA;
  intdosx(&regs, &regs, &sregs);
  c_header.cram_dta_seg = sregs.es;
  c_header.cram_dta_off = regs.x.bx;

  c_header.ID = 0x42424152;
  setmem(&c_header.f_access, 0x40, 0xDB);
  setmem(&c_header.password, 0x10, '*');
  setmem(&c_header.reserve_for_seg, MAX_SEG, '@');
  capture_mem();
  put_header();
  if(strstr(strupr(argv[1]), "D") == NULL)
  goto KEEP;
  DEBUG = 1;
  c_header.start_addr = lowmem;
  c_header.data_sec = data_sec;
  c_header.curr_mem_sec  = data_sec + 1;
  capture_mem();
  save_interrupt();
  c_header.int_sec = DOSsec;
  sav_start = START_SEG;
  sav_end   = maxmem;

        /* Skip to next track if now not at first sector in the track*/

 seg_cal = ldiv(DOSsec, bpb.nspt);
 DOSsec += (bpb.nspt - seg_cal.rem);
 c_header.mem_sec = DOSsec;
 if (put_memory() == 0)
        {
          c_header.video_sec = DOSsec;
          save_video();
          DOSsec +=VIDEO_SEC_SIZE;
          c_header.mcb_sec = DOSsec;
          DOSsec++;
          c_header.dsa_sec = DOSsec;
          save_DSA();
          capture_mem();
          put_header();
        }
```

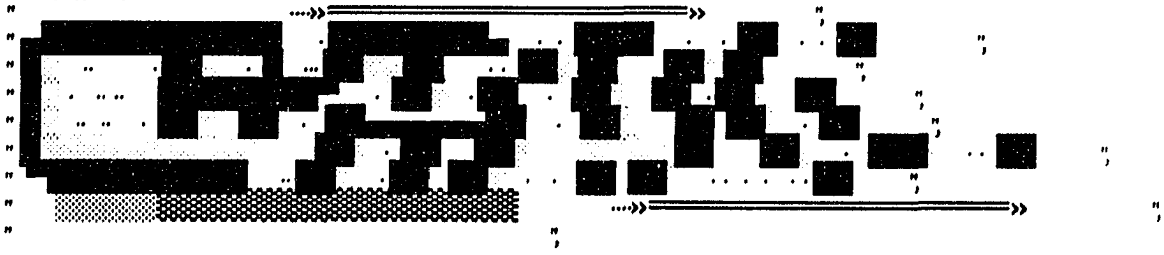
```
    else
          {
           printf("\n\rDISK ERROR saving memory\n\r");
           return (-1);
          }
  }
 if(strstr(strupr(argv[1]), "U") != NULL)
  {
   start_flag = 0;
   get_header();
   if (signote[505] == 0x01)
          {
           capture_mem();
           signote[505] = 0x02;
           put_header();
           delay(1500);
           poke(0x0040,0x0072, 0x1234);
           cram_env[0].j_ip = 0;
           cram_env[0].j_cs = 0xFFFF;
           cram_env[0].j_di = 0;
           cram_env[0].j_si = 0;
           cram_env[0].j_ds = 0xFFFF;
           cram_env[0].j_es = 0xFFFF;
           longjmp(cram_env,2);
          }
  else
        if (! signote[505])
          exit(99);
  lowmem = c_header.start_addr;
  data_sec = c_header.data_sec;
  seg_off = ldiv(lowmem, 0x1000L); /* calculate the segment:offset pair */
  START_OFF = (seg_off.rem << 4);

/* Make sure memory is align with sector size */

  offs_cal = ldiv((START_OFF), (long)bpb.bps);
  START_SEG = (seg_off.quot << 12);/* using 64k segment boundries  */
  /*c_header.sec_seg1_end = 0;*/
  sec_seg1_end = 0;
  sav_start = START_SEG;
  sav_end   = maxmem;
  reset_mem = 2;

  put_str("\n\r Restoring Memory at Address : ");
  put_hex(lowmem);
  put_str("\n\rI will give you control in a while Please Stand By...\n\r");

  if(strstr(strupr(argv[1]), "D") != NULL)
   {
         DEBUG = 1;
         reset_memory();
         DEBUG = 0;
   }
  goto KEEP;
 }

if(strstr(strupr(argv[1]), "K") != NULL)
{
 get_header();
```

```
      start_flag = 0;
      lowmem = c_header.start_addr;
      data_sec = c_header.data_sec;
      seg_off = ldiv(lowmem, 0x1000L); /* calculate the segment:offset pair */
      START_OFF = (seg_off.rem << 4);

   /* Make sure memory is align with sector size */

      offs_cal = ldiv((START_OFF), (long)bpb.bps);
      START_SEG = (seg_off.quot << 12);/* using 64k segment boundries  */
     /*c_header.sec_segl_end = 0;*/
      sec_segl_end = 0;
      sav_start = START_SEG;
      sav_end   = maxmem;
      reset_mem = 2;
      goto KEEP;
    }
   else
    {
     return(2);
    }

    KEEP:
   sig_ptr = MK_FP(FP_SEG(0x0000), FP_OFF(0x04FB));
   diskette = MK_FP(FP_SEG(0x0000), FP_OFF(0x043F));
   if ((*sig_ptr == 'R') && (*(sig_ptr+1) == 'B'))
     {
          printf("%c%c\b\b\t\n\r CRAM already Installed \n\r", 0x07,0x07);
          exit(-1);
     }
          hot_key = FIVEKEY;
          set_shift_key(ALT_KEY|CTRL_KEY);

   get_header();
   start_flag = 0;
   /*c_header.sec_segl_end = sec_segl_end = 0;*/
   sec_segl_end = 0;
   START_SEG = sav_start;          /* Set to start of segment to save */
   maxmem    = START_SEG + 0x1000; /* Set end of memory to save this  */
                                           /* is one segment boundary block   */
   DOSsec = c_header.mem_sec;
   c_header.curr_mem_sec = DOSsec;
   while(*diskette != 0x00); /* wait for disk drive to stop spinning */
   time(&t1);

   stack_ptr = malloc(STACK_SIZE);
   stack_ptr += STACK_SIZE;

   disable();
   init_intr();
   /* get interrupt vector */

   old_int8  = getvect(8);    /* timer interrupt */
   old_int9  = getvect(9);    /* keyboard interrupt */
   old_int2a = getvect(0x2A); /* dos internal int */
   old_int24 = getvect(0x24);
   old_int1C = getvect(0x1C);
   old_int28 = getvect(0x28);
   old_intFC = getvect(0xFC);
```

```
    old_int25 = getvect(0x25);
    old_int13 = getvect(0x13);
    old_int26 = getvect(0x26);
    old_int10 = getvect(0x10);

    /* set interrupts to our routines */
     disable();
/*     setvect(0x8, new_int8);*/
     setvect(0x9, new_int9);
     setvect(0x2A, new_int2a);
     setvect(0x24, new_int24);
     setvect(0x1C, new_int1C);
/*     setvect(0x10, new_int10);*/
/*     setvect(0x25, new_int25);
     setvect(0x26, new_int26);*/
/*     setvect(0x13, new_int13);*/

/*    setvect(0x28, mem_save);*/
    setvect(0x28, new_int28);
    setvect(0xFC, mem_reset);
    gen_intFC = getvect(0xFC);
    enable();
    FP_SEG(fp) = _psp;
    FP_OFF(fp) = PSP_ENV_ADDR;
    farfree(fp);
    segread(&sregs);
    memtop = sregs.ds + PARAGRAPHS(stack_ptr) - _psp;
    setblock(_psp, memtop);
    *sig_ptr = 'R'; *(sig_ptr + 1) = 'B';
    if(strstr(strupr(argv[1]), "R") != NULL) goto ret;
    printf("\r\nCRAM  installed using %lu bytes of memory ", (unsigned long)(memtop * 16));
    keep(0, memtop);

    ret:
    return(result);
}
```

INSTALL.C The installation routines used to prepare the hard disk and to install a copy of the software.

```
 #include <bios.h>
#include <dir.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#include <setjmp.h>
#include <mem.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <process.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#define  BREAK_int 0x23

#include "tsr.h"
#include "cram.h"

struct BPB bpb;
struct DPT dpt;
struct HST hst;
struct CRAM_HEADER  c_header;
long maxtracks, maxsides, maxsectors, skiptrack;
extern DWORD cluster2sector(DWORD);
long  DOSsec;
int result;
BYTE   buffer[512];
BYTE   swap_save_buf[1853];
int             nsects;
unsigned char far *conv_mem_ptr;
int             drive=0;
char             *drv="Z";
DWORD data_sec, reserved_sec;
struct free_fat f_free;
DWORD  clust_req  =600UL;
BYTE     far      *diskette;
int               DEBUG = 0;
char *logo[] ={
```



```
    " Constant Random Access Memory data recovery system for the IBM PC       ";
    "                                                                          ";
    "  Copyright (c) 1992 by Renford A. B. Brevett at Iowa State University   ";
    "                                                                          ";
    " This software replaces a standard Uninterruptible Power Supply (UPS)    ";
    " Also recover from keyboard lockup, system crash, & memory parity errors ";
    " Memory will be saved automatically and restored if power is interrupted ";
    "                                                                          "

    };



 extern int  get_drive_info(char *drv);
 extern int  get_fatfree(struct free_fat *freefat, WORD clust_req);
 extern int  put_f_name(struct free_fat f_free, DWORD data_sec);
 extern int  put_fat(struct free_fat freefat, DWORD data_sec);
 extern long get_f_name(void);
 extern int format_CRAM(long start_sec, long end_sec);

 char  curdir[MAXPATH];
 char        command_input[80];

 void interrupt (*oldctrl_c)();
 char *SAVE_DIR(char *path);
 void shell_install(void);


 void interrupt ctrl_c(void)
 {
  gotoxy(1,1);
  cprintf(" %c CTRL-C Disabled  ",0x07);
  cprintf("Do you want to abort the Instalation [Y/N] ");
  if(toupper(getch()) = =  'Y')
   {
/*  put code to  remove CRAM here */
/*     cprintf("\n\r Not Supported yet YOU need to do so on your own");
    cprintf("\n\r See manual for Instructions  ");*/
    gotoxy(1,1);
    cputs("\bThanks for trying CRAM; need help call (302) 325-0876 or mail to addr. in manual");
    gotoxy(1,24);
    disable();
    setvect(0x23,oldctrl_c);
    enable();
    exit(95);
  }
  else
   return;
}

char *SAVE_DIR(char *path)
 {
  strcpy(path, "X:\\>");
  path[0] = 'A' + getdisk();
  getcurdir(0, path+3);
  return (path);
 }

void shell_install(void)
{
  int c_handle, c2_handle;
  FILE  *inf, *outf;
  char autobuff[12]={"           "};
  char buf[80];
  BYTE bigbuf[40960];
  char *src = "A:\\autoexec.bat",
       *dst = "C:\\autoexec.bat";
  char tokensep[]= "\t/, ";
  char *token;
  int flag=0;
  struct ftime c_ftime;
  char ch[8]= "Echo  \n\0";

  src[0] = 'A' + getdisk();
  strcpy(dst, drv);
  src[1] = '\0';
  strcat(src, ":CRAM.EXE ");
  strcat(dst, ":\\CRAM.EXE");
  clear_win(1,1,79,2,0xB0,0);
  gotoxy(1,1);cprintf("Copying %.*s to %.*s", 11, src, 11, dst);
  chmod (src, S_IWRITE);
  chmod (dst, S_IWRITE);
  if((c_handle = open(src, O_RDWR|O_BINARY)) != -1)
   {
     char *chk="CRAMEXE";
     long bytesread=99, i, save_f_pos=0, f_size;

     c2_handle = open(dst, O_WRONLY|O_BINARY|O_CREAT, S_IREAD);
     getftime(c_handle, &c_ftime);
     ultoa(biosequip(), buf, 16);
     lseek(c_handle, 0L, 0L);
     lseek(c2_handle, 0L, 0L);
     f_size = filelength(c_handle);
     while( save_f_pos <= f_size && (bytesread > 0))
             {
              if (bytesread == 99) bytesread = 0;
              save_f_pos += bytesread;
              bytesread =(unsigned int)read( c_handle,bigbuf, sizeof(bigbuf));
              for(i=0; i<bytesread; i++)
                if(bigbuf[i] == chk[0])
                  if(memcmp(chk, &bigbuf[i], 7) == 0)
                         {
                           memcpy(&bigbuf[i+7], &buf, 4);
                         }
              lseek(c_handle, save_f_pos, 0L);
              write(c_handle,bigbuf,bytesread);
              lseek(c2_handle,save_f_pos, 0L);
              write(c2_handle,bigbuf,bytesread);
             }
   /* Put system protection ID check Here */
   setftime(c_handle, &c_ftime);
   setftime(c2_handle, &c_ftime);
   close(c_handle);
   close(c2_handle);
/*   chmod (src, S_IREAD);  */
   chmod (dst, S_IREAD);

   clear_win(1,1,79,2,0xB0,0);
   gotoxy(1,1);cprintf("Checking AUTOEXEC.BAT ...");
```

```
     strcpy(src, dst);
     strcpy(src+3,"autoexec.bat");
     if (access(src,2) == -1 || access(src,0) == 0)
     {
      if (access(src,2) == -1)
             chmod(src,S_IWRITE);
      gotoxy(65,1);puts(src);
      strcpy (dst, src);
      src[12]='O'; src[13]='L'; src[14]='D';
      rename(dst, src);
      inf = fopen(src,"r");
      while(fgets(command_input, sizeof(command_input), inf) != NULL)
      {
             strcpy(buf, command_input);
             token = strtok(command_input, tokensep);
             while(token != NULL)
              {
                if (strnicmp(token, "CRAM /u",7) == 0)
                  flag = 1;
                token = strtok(NULL, tokensep);
              }
      }
    rewind(inf);
    }
    if(! flag)
     {
            clear_win(1,1,79,2,0xB0,0);
            gotoxy(1,1);cprintf("Updating AUTOEXEC.BAT ...");
            outf = fopen(dst, "w");
            strcpy(autobuff,"CRAM /u ");
            strcat(autobuff,drv);
            autobuff[9] ='\n';
            autobuff[10] ='\0';
            autobuff[11] ='\0';
            fputs("Echo off\n", outf);
            fputs("Cls\n", outf);
            fputs(autobuff, outf);
            fputs("If ERRORLEVEL 99 goto install\n", outf);
            fputs("goto end\n", outf);
            fputs(":install\n", outf);
            autobuff[6] = 'i';
            fputs(autobuff, outf);
            ch[5] = 0x13;
            fputs(ch, outf);
            ch[5] = 0x07;
            fputs(ch, outf);
            ch[5] = 0x07;
            fputs(ch, outf);
            gotoxy(27,1);cprintf("Copying %.*s  to  %.*s", 11, src, 11, dst);
/*          if (access(src,0) == 0)*/
            while(fgets(command_input, sizeof(command_input), inf) != NULL)
              {
                fputs(command_input, outf);
              }
            fputs("\n:end\n", outf);
            fcloseall();
            chmod(src,S_IREAD);
            chmod(dst,S_IREAD);
   }
   }
   clear_win(1,1,79,2,0xB0,0);
 }


int main(void)
{
  unsigned long  seg, offs, i;
  ldiv_t seg_cal, offs_cal;
  long result;

  textcolor(7);
  textbackground(1);
  drive = get_drive_info(drv);
  clear_screen(0xB0);
  i=0;
  for (i = 0; i<16; i++)
    {
      gotoxy(3, 5+i);
      cputs(logo[i]);
    }
  textattr(RED+(GREEN<<4));
  gotoxy(10,5+8);
  cprintf("Version  %u.%.2u    [", c_ver.major, c_ver.minor);
  if(c_ver.test) cprintf(" EVALUATION ");
  if(c_ver.beta) cprintf(" BETA  ");
  if (c_ver.test || c_ver.beta) cprintf("COPY ]");
  else
    {
      cprintf("Licence to : %.*s  lic# %.*s",
      c_header.f_access[0], c_header.f_access,
      c_header.password[0],c_header.password);
    }
  gotoxy(1,6+i);
  textattr(WHITE+(BLUE<<4));
  reserved_sec = ( (DWORD)(bpb.nspf*bpb.nfats) + (DWORD)bpb.nres_sec
                        + (DWORD)((bpb.nroot_dir *32)/bpb.bps));
  if(strstr(strupr(_argv[1]), "-D"))
    {
      DEBUG = 1;
    }
  switch(dpt.sysid)
                  {
                          case D_FAT16:
                            {
                                      clust_req = 500UL;
                                      break;
                            }
                          case D_FAT12:
                          case      0:
                            clust_req = 600UL;
                            break;
                  }
 oldctrl_c = getvect(0x23);
 disable();
 setvect(0x23,ctrl_c);
 enable();
 result = get_f_name();
 if(result <= 0)
```

```
  {
   if (strstr(strupr(_argv[1]), "-F"))
          result = get_data_sec();
   if (result < = 0)
          {
           result = get_fatfree(&f_free, clust_req);
           data_sec = cluster2sector((f_free.fbegin));
          }
   else
          data_sec = result;
  }
else
  {
   f_free.fbegin = result;
   f_free.fsize = clust_req;
   f_free.fend = result + clust_req - 1;
   result = cluster2sector(result);
  }

if(DEBUG)
  {
   gotoxy(1,22);
   cprintf("Last Result = %ld", result);
   cprintf("\n\rLargest set of free consecutive Clusters Begins: %lu   "
          "Ends %lu  size %lu \n\rCRAM will be loaded at Cluster %lu "
          "sector %lu [Above Reserved Area at %lu]",
           f_free.fbegin, f_free.fend, f_free.fsize, f_free.fbegin,
           data_sec, reserved_sec);
   pause(0);
  }
if (result > 0)
  {
   if (f_free.fsize > = clust_req)
         {
          f_free.fsize = clust_req;
          f_free.fend  = f_free.fbegin + f_free.fsize;
          if(data_sec > reserved_sec)
            result = put_f_name(f_free, data_sec);
           SAVE_DIR(curdir);
           shell_install();
           chdir(curdir);
          if(result > = 0)
           {
            result = put_fat(f_free, data_sec);
            format_CRAM(data_sec, cluster2sector(f_free.fend));
            gotoxy(3, 22);
            textattr(WHITE+(RED< <4)+BLINK);
            cputs("\bCRAM Installed Successfully on drive ");
            textattr(YELLOW+(RED< <4));
            cputs(drv);
            cputs("; The file AUTOEXEC.BAT on ");
            cputs(drv);
            cputs(" is updated ");
            gotoxy(3,23);
            cputs("       You need to Re-Boot your computer to start CRAM             ");
            textcolor(7);
            textbackground(1);
           }
          else
          {
          gotoxy(1,1);
          cprintf("%c%cDo you want to REformat CRAM [Y/N] ", 0x07, 0x07);
          if(toupper(getch()) = = 'Y')
              {
                format_CRAM(data_sec, cluster2sector(f_free.fend));
              }
           gotoxy(3, 22);
           textattr(WHITE+(RED< <4)+BLINK);
           cputs(" \bCRAM Already Installed on drive ");
           textattr(YELLOW+(RED< <4));
           cputs(drv);
           cputs("; The file AUTOEXEC.BAT on ");
           cputs(drv);
           cputs(" is updated ");
           gotoxy(3,23);
           cputs("       You need to Re-Boot your computer to start CRAM          ");
           textcolor(7);
           textbackground(1);
           result = 0;
          }

        }
      }
  if (result < 0)
    {
     gotoxy(1,22);
     textattr(WHITE+(RED< <4)+BLINK);
     cputs(" \b\bUnable to load CRAM on drive ");
     textattr(YELLOW+(RED< <4));
     cputs(drv);
     cputs(" : Insufficient Disk space or Disk is fragmented ");
     cputs("\n\r   If you did not run a disk organizing utility,");
     cputs(" Do so now and re-run CRAM    ");
     textcolor(7);
     textbackground(1);
    }
/* clear_win(1,1,79,2,0xB0,0);*/
 gotoxy(1,1);
 cputs("\bThanks for trying CRAM; need help call (302) 325-0876 or mail to addr. in manual");
 gotoxy(1,24);
 disable();
 setvect(0x23,oldctrl_c);
 enable();
 return (0);
}
```

APPENDIX D. SOURCE CODES FOR SOME USEFUL UTILITIES

<u>GEN_UTIL.C</u> Routines used for screen display and other DOS operation but not essential to CRAM's operation.

```
/*                  GEN_UTIL.C                  */
/***
 Utilities used with CRAM for general I/O and control of memory
 Copyright for Iowa State University by Renford A. B. Brevett.


***/
#include <stdlib.h>
#include <dos.h>
#include <stdio.h>
#include <mem.h>
#include <string.h>
#include <stdarg.h>
#include <bios.h>
#include <io.h>
#include <conio.h>
#include "cram.h"


#define STDERR          fileno(stdout)
#define KEYBRD_READY     0x01
#define MAX_WID          12
/*
static union  REGS regs;*/
static union REGS rg;

/* Prototypes */
int color_adpt(void);
SCRLINE      far *scr;

unsigned put_str(char far *s);
unsigned put_num(unsigned long u, unsigned wid, unsigned radix);
unsigned put_chr(int c);
#define put_hex(u)     put_num(u, 4, 16)
#define put_long(ul)   put_num(ul, 9, 10)
#define putstr(s)   { put_str(s); put_str("\r\n"); }

int LPT(char *s);
void  gotoXY(int x, int y);
void  curr_cursor(int *x, int *y);
void  set_cursor_type(int t);
void  clear_screen(char ch);
void  clear_win(int x1, int y1, int x2, int y2, char ch, BYTE attrib);
int   vmode();
int   scroll_lock();
int   get_char();
void  clrEol(void);
void  (*helpfunc)(void);


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

int color_adpt(void)
```

```
/* Return 0 if monochrome adapter, 1 if color adapter          */
{
     return ((biosequip() & 0x0030) != 0x0030);
}

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

int LPT(char *s)
 {
   int i=0;
   while (*s)
     { i++;
          if (i > 80)
            {
              i = 0;
              biosprint(0, 0x000A, 0);
              biosprint(0, 0x000D, 0);
            }
          biosprint(0,*s, 0);
          s++;
     }
   return (0);
 }


/* TSR  STDERR output routines, no malloc */

/*  returns length of far string  */

#ifdef _MSC_VER
#define fstrlen(s)      _fstrlen(s)     // MSC 6.0
#else

size_t fstrlen(const char far *s)       // MSC 5.1
{
   size_t len = 0;
   while (*s++) len++;
   return len;
}
#endif


size_t strlen(const char  *s)       // _TURBO_C_
{
   size_t len = 0;
   while (*s++) len++;
   return len;
}

unsigned doswrite(int handle, char far *s, unsigned len)
{
   unsigned bytes;
   bytes = _write(handle,(void *)s, len  );
   return bytes;
}

unsigned put_str(char far *s)
{
   return doswrite(STDERR, s, fstrlen(s));
}

unsigned put_chr(int c)
{
   return doswrite(STDERR, (void far *) &c, 1);
}

unsigned put_num(unsigned long u, unsigned wid, unsigned radix)
{
   char buf[MAX_WID+1], *p;
   int i, digit;
   if (wid > MAX_WID)
         return (0);
   for (i=wid-1, p=&buf[wid-1]; i >= 0; i--, p--, u /= radix)
   {
      digit = u % radix;
      *p = digit + ((digit < 10) ? '0' : 'A' - 10);
   }
   buf[wid] = 0;
   return (doswrite(STDERR, (void far *) buf, wid));
}


/*  Substitute for printf function */

int    _Cdecl TSRprintf           (const char *fmt, ...)
{
   static char buf[128];
   int len;
   va_list marker;
   va_start(marker, fmt);
   len = vsprintf(buf, fmt, marker);
   va_end(marker);
   return doswrite(STDERR, (void far *) buf, len);
}

unsigned get_str(char far *s, unsigned len)
{
   unsigned rcount;

   /* give TSRs a chance by calling INT 28h */
   while (! bioskey(KEYBRD_READY))
         geninterrupt(0x28);

   if (((rcount = _read(STDERR, (void *)s, len)) != 0) || (rcount < 3))
         return 0;
   s[rcount-2] = '\0';
   return rcount-2;
}

/* --------------- ibmpc.c -------------- */

/*
 * Low-level functions addressing BIOS & PC Hardware
 */


/* ----------- position the cursor ------------- */
```

```
void gotoXY(int x, int y)
{
        rg.x.ax = 0x0200;
        rg.x.bx = 0;
        rg.x.dx = ((y << 8) & 0xff00) + x;
        int86(0x10, &rg, &rg);
}

/* ------------ return the cursor position ------------- */

void curr_cursor(int *x, int *y)
{
        rg.x.ax = 0x0300;
        rg.x.bx = 0;
        int86(16, &rg, &rg);
        *x = rg.h.dl;
        *y = rg.h.dh;
}

/* ----------- set cursor type --------------- */

void set_cursor_type(int t)
{
        rg.x.ax = 0x0100;
        rg.x.bx = 0;
        rg.x.cx = t;
        int86(16, &rg, &rg);
}
/*page*/
char attrib = 7;

/* ------------- clear the screen -------------- */

void clear_screen(char ch)
{
        scr = MK_FP((color_adpt() ? 0xB800: 0xB000), 0x0000);
        gotoXY(0, 0);
        rg.h.al = ch;
        rg.h.ah = 9;
        rg.x.bx = (scr[wherex()][wherey()].s_attr);
        rg.x.cx = 2000;
        int86(0x10, &rg, &rg);
}

void clear_win(int x1, int y1, int x2, int y2, char ch, BYTE attrib)
{
        scr = MK_FP((color_adpt() ? 0xB800: 0xB000), 0x0000);
        gotoXY(x1-1, y1-1);
        rg.h.al = ch;
        rg.h.ah = 9;
        rg.x.bx = attrib ? attrib&0x77 : (scr[wherex()][wherey()].s_attr);
        rg.x.cx = ((y2-y1)+2)* ((x2-x1) ? 80 : 80); /*2000;*/
        int86(0x10, &rg, &rg);
}

/* ----------- return the video mode ------------ */

int vmode()
{
```

```
            rg.h.ah = 15;
            int86(16, &rg, &rg);
            return rg.h.al;
    }

    /* -------- test for scroll lock -------- */

    int scroll_lock()
    {
            rg.x.ax = 0x0200;
            int86(0x16, &rg, &rg);
            return rg.h.al & 0x10;
    }

    void (*helpfunc)(void);
    int helpkey = 0;
    int helping = 0;

    /* ------------ get a keyboard character ---------------- */

    int get_char()
    {
            int c;

            while (1)  {
                    rg.h.ah = 1;
                    int86(0x16, &rg, &rg);
                    if (rg.x.flags & 0x40) {
                            int86(0x28, &rg, &rg);
                            continue;
                    }
                    rg.h.ah = 0;
                    int86(0x16, &rg, &rg);
                    if (rg.h.al = = 0)
                            c = rg.h.ah | 128;
                    else
                            c = rg.h.al;
                    if (c = = helpkey && helpfunc) {
                            if (!helping)          {
                                    helping = 1;
                                    (*helpfunc)();
                                    helping = 0;
                                    continue;
                            }
                    }
                    break;
            }
            return c;
}

/*
void gotoXY(int x, int y)
  {
   union REGS r;

   r.h.ah = 2;
   r.h.dl = x;
   r.h.dh = y;
   r.h.bh = 0;
  int86(0x10, &r, &r);
 }
*/

 void clrEol(void)
 {
  union REGS r;

  scr = MK_FP((color_adpt() ? 0xB800: 0xB000), 0x0000);
  r.h.ah = 6;
  r.h.al = 0;
  r.h.ch = wherey();
  r.h.cl = wherex();
  r.h.dl = 79;
  r.h.dh = wherey();
  r.h.bh = (scr[wherex()][wherey()].s_attr);
  int86(0x10, &r, &r);
 }
```

CRAMINFO.C Routines used to access information in CRAM. These routines can be used with any other software to gain access to information stored in CRAM.

```
 #include <mem.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <setjmp.h>
#include <time.h>
#include <alloc.h>
#include "tsr.h"
#include "cram.h"

struct date ddate;
struct time dtime;

#define SIG    "Copyright(c) 1991 Renford A. B. Brevett at Iowa State University "
#define LINEFEED          0x13
#define SPACE             0x20
#define PSP_ENV_ADDR      0x2C   /* environment address from PSP */
#define STACK_SIZE        8192
#define BYTES_PER_TRACK   4609   /* Total bytes per track plus 1 for null byte*/
#define PARAGRAPHS(x)     ((FP_OFF(x) + 15)  >> 4)
/*
extern unsigned put_str(char far *s);
extern unsigned put_num(unsigned long u, unsigned wid, unsigned radix);
#define put_hex(u)      put_num(u, 4, 16)
#define put_bit(u)      put_num(u, 2, 16)
#define put_long(ul)    put_num(ul, 9, 10)
#define putstr(s)    { put_str(s); put_str("\r\n"); }
*/
unsigned char multiplex_id;

struct dfree          dskfree;
long                  dskavail,
                              sysmem;
DWORD                    maxmem = 0xA000L;
static DWORD                  sav_start, sav_end;
int                   skiptrack;
int                   spc;
int                   DEBUG = 0;
struct DPT              dpt;
struct BPB              bpb;
struct HST              hst;
int                   spt;
int                       cmd;
int                       drive;
int                       head;
long                      track;
int                       sector;
```

```
  int                  nsects;
  BYTE                 buffer[BYTES_PER_TRACK];
  BYTE                 swap_save_buff[1852];
  static BYTE          signote[512];
  int                  result;
  char                 drv[3] = "Z ";
  long                 maxsectors;
  long                 maxsides;
  long                 maxtracks;
  long                 DOSsec;
  unsigned long        data_sec;
  unsigned char far     *conv_mem_ptr;
  ldiv_t             seg_cal;
  ldiv_t             offs_cal;
  ldiv_t             seg_off;
  unsigned long         seg  = 0x0000,
                               offs = 0x0000;
  WORD                  lowmem;
  MCB far                    *mcb;
  MCB far                    *first_mcb;
  unsigned long         START_SEG,
                               START_OFF,
                               SEC_SEG,
                               REM_SEC;
  static BYTE           start_flag=0;
  BYTE                  reset_mem;
  WORD                   sec_seg1_end =0;
  unsigned long         i, j;
  unsigned long         savetime = 0L;
  static BYTE           timelag  = 0x3C;
  static unsigned long  secloop;
  static time_t         t1, t2;

  struct CRAM_HEADER        c_header;
  char far              *stack_ptr;
  char far              *ptr;
  char far              *sig_ptr;
  char far               *sig;
  unsigned far           *fp;
  unsigned              memtop;
  jmp_buf               cram_env;
  struct ExtErr         *ErrInfo;
  WORD       header_size;
  char       c_id[2][4];
  DWORD  reserved_sec;
  struct free_fat f_free;
  DWORD  clust_req =500UL;
  extern struct   SREGS    sregs;
  extern union    REGS     regs;
  extern SCRLINE  far      *scr;
  extern MCB far *get_mcb (void);
  char far *diskette;

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

/*==================================================================*/
int get_header(void)
  {
   long k=0L;
```

```
      nsects = 1;
      scr[0] [0].s_char = 0x00F0;
      scr[0] [0].s_attr = ((scr[0] [0].s_attr > > 4) + (scr[0] [0].s_attr < < 4)) & 0x77;
      DOSsec = data_sec;
      if ((result = getsector(drive, nsects, &DOSsec, &bpb, &dpt,
                 signote)) != 0)
      return(result);
      memcpy(&sig, &signote[k], /*sizeof(sig)*/strlen(SIG) );
      k + = strlen(SIG);
      memcpy(&dpt, &signote[k], sizeof(dpt) );
      k + = sizeof(dpt);
      memcpy(&bpb, &signote[k], sizeof(bpb) );
      k + = sizeof(bpb);
      memcpy(&c_header, &signote[k], sizeof(c_header) );
      k + = sizeof(c_header);
      memcpy(&cram_env, &signote[k], sizeof(cram_env));
      k + = sizeof(cram_env);
      memcpy(&header_size,&signote[506],  sizeof(header_size));
      memcpy(&c_id[1] ,&signote[508], 4);
      memcpy(&c_id[0] ,&c_header.ID, 4);

      return(k);
    }

  int reset_video(void)
   {
    nsects = 8;
    DOSsec = c_header.video_sec;
    stohst(drive, hst.TRACK, hst.HEAD, &DOSsec, &bpb, &dpt, &hst);
    if ((result = biosdisk(READ, hst.DRIVE_NUM, hst.HEAD, hst.TRACK,
      hst.SECTOR, nsects, buffer)) != 0)
                        return(result);
    movedata(FP_SEG(buffer), FP_OFF(buffer), 0xB800, 0x0000, 0x1000);
    nsects = 1;
    return(0);
   }
  void print_regs(void)
  {
    printf("\n\r\n\r Registers saved for the ongoing process in CRAM \n\r");
    printf(   "\r\n CS = %.4X  DS = %.4X  ES = %.4X  SS = %.4X"
                        "\r\n AX = %.4X  BX = %.4X  CX = %.4X  DX = %.4X"
                        "\r\n BP = %.4X  DI = %.4X  SI = %.4X  SP = %.4X"
                        "\r\n IP = %.4X  PSP = %.4X  FLAGS  = %.4X\r\n"
                        "\r\n CRAM PSP = %.4X",
                        c_header.CS,
                        c_header.DS,
                        c_header.ES,
                        c_header.SS,
                        c_header.AX,
                        c_header.BX,
                        c_header.CX,
                        c_header.DX,
                        c_header.BP,
                        c_header.DI,
                        c_header.SI,
                        c_header.SP,
                        c_header.IP,
                        c_header.PSP,
                        c_header.FLAGS,
                    c_header.cram_psp);
}


int printHeader(void)
{
 char c_ID[5]={'x','x','x','x','\0'};
 unsigned day, month, year,hour,min,sec;
 char  am_pm[3] = {'a','m','\0'};
 if (((c_header.time & 0xF800) > > 0x0B) > = 12)
   am_pm[0] = 'p';
 else
   am_pm[0] = 'a';
 memcpy(&c_ID, &c_header.ID, 4);

 printf("\r\n CRAM ID.          %.*s        %.*s"
          "\r\n DATE & TIME OF LAST SAVE:    %.2u-%.2u-%u   %.2u:%.2u:%.2u%s"
          "\r\n ACCESS INFORMATION                      %-.*s",
           4,c_id[0],
           4,c_id[1],
           (c_header.date & 0x1E0) > > 0x05,
           (c_header.date) & 0x1F,
           1980 + ((c_header.date & 0xFE00) > > 0x09),
           (((c_header.time & 0xF800) > > 0x0B) > 12)
             ? ((c_header.time & 0xF800) > > 0x0B) -12
             : ((c_header.time & 0xF800) > > 0x0B),
           (c_header.time & 0x7E0) > > 0x05,
           (c_header.time & 0x1F) * 2,
           am_pm,
           16, (c_header.f_access[0] = = 0x00 ? "NOT assigned" : c_header.f_access)
          );

 printf("\r\n CRAM CHECKSUM                              %.4X"
          "\r\n TOTAL CLUSTERS OCCUPIED BY CRAM                %lu"
          "\r\n CLUSTER WHERE CRAM STARTS                      %lu",
           c_header.checksum,
           c_header.clusters,
           c_header.start_cluster);

 printf("\r\n ALL PASSWORDS                  %-.*s"
          "\r\n EXTENDED ERROR INFO                        %.4X:%.4X:%.4X (ax:bx:cx)"
          "\r\n SEGMENT AT START OF CRAM                          %.4X"
          "\r\n CRAM FILE SIZE                            %lu BYTES"
          "\r\n CHECKSUM FOR INTERRUPT VECTOR                     %.4X"
          "\r\n SIZE OF MEMORY FILLER                      %u  BYTES"
          "\r\n SECTOR AT END OF FIRST SEGMENT IN CRAM          %u",
           4, (c_header.password[0] = = 0x00 ? "NONE" : c_header.password ),
           c_header.ErrInfo.errax,
           c_header.ErrInfo.errbx,
           c_header.ErrInfo.errcx,
           c_header.start_addr,
           c_header.f_size,
           c_header.int_checksum,
           c_header.offs_filler,
           c_header.sec_seg1_end
          );
printf("\r\n SECTOR AT START OF VIDEO MEMORY              %u"
         "\r\n SECTOR AT START OF DSA MEMORY                %u"
         "\r\n SECTOR AT START OF MCB MEMORY                %u"
```

```
            "\r\n SECTOR AT START OF STACK MEMORY              %u"
            "\r\n SECTOR AT START OF INTERRUPT VECTOR MEMORY      %u"
            "\r\n SECTOR AT START OF CRAM MEMORY              %u"
            "\r\n SECTOR OF LAST CRAM MEMORY SAVED             %u"
            "\r\n SECTOR AT START OF CRAM HEADER              %u"
            "\r\n SIZE OF DOS SDA                     %d"
            "\r\n DISK TRANSFER (DTA) ADDRESS [  CRAM  ]     [%.4X:%.4X]"
            "\r\n DISK TRANSFER (DTA) ADDRESS [process ]     [%.4X:%.4X]",
             c_header.video_sec,
             c_header.dsa_sec,
             c_header.mcb_sec,
             c_header.stack_sec,
             c_header.int_sec,
             c_header.mem_sec,
             c_header.curr_mem_sec,
             c_header.data_sec,
             c_header.dsa_size,
             c_header.cram_dta_seg,
             c_header.cram_dta_off,
             c_header.dta_seg,
             c_header.dta_off
             );
  pause(0);
  printf("\r\nCRAM SEGMENTS REGISTERS INFORMATION \r\n\r\n"
         "SEGMENT   FLAG    CHECKSUM    SECTOR     SIZE      RESERVE");
  for (i=0; i<MAX_SEG; i++)
   {
     WORD j;

     j = i * 0x1000;
     printf("\n\r %.4X       %c        %.4X        %.4X        %.4X        %c",
            j,
            c_header.save_seg_flag[i] + 0xFA,
            c_header.seg_checksum[i],
            c_header.sector_in_cram[i],
            c_header.size_of_seg[i],
            c_header.reserve_for_seg[i]
            );
     }
     cprintf("\n\rSIZE OF HEADER : [%u] ", header_size);
  return(0);
}

void printJmp(void)
{
  printf("\r\n\r\nLast JMP information \r\n");

  printf(
          "\r\n CS = %.4X  DS = %.4X  ES = %.4X  SS = %.4X"
          "\r\n BP = %.4X  DI = %.4X  SI = %.4X  SP = %.4X"
          "\r\n IP = %.4X  FLAGS = %.4X",
           cram_env[0].j_cs,
           cram_env[0].j_ds,
           cram_env[0].j_es,
           cram_env[0].j_ss,
           cram_env[0].j_bp,
           cram_env[0].j_di,
           cram_env[0].j_si,
           cram_env[0].j_sp,
              cram_env[0].j_ip,
              cram_env[0].j_flag
             );
/* typedef struct {
    unsigned  j_sp,   j_ss,
    unsigned  j_flag, j_cs;
    unsigned  j_ip,   j_bp;
    unsigned  j_di,   j_es;
    unsigned  j_si,   j_ds;
  } jmp_buf[1];
  */
}

int main(int argc, char *argv[])
{
 unsigned long seg, offs=0L;
 ldiv_t seg_cal, offs_cal;
 DWORD cksum;

 if(argc > 1)
   if(strstr(strupr(argv[1]),"M") != NULL)
     {
           clrscr();
           i = 5;
           cksum = mem_checksum(0x0000,0x0000, 0x400);
           gotoxy(1,i+ +);
           printf ("Interrupt Memory Checksum     : %.8X",cksum);
           cksum = 0xBB;
           cksum = mem_checksum(0x0000,0x0000, 0xA0000);
           gotoxy(1,i+ +);
           printf ("Total Memory Checksum(640K)  : %.8X",cksum);
           cksum = 0xBB;
           cksum = mem_checksum(0x0000,0x0000, 0x100000);
           gotoxy(1,i+ +);
           printf ("Total Memory Checksum(1 Meg.): %.8X");
           cksum = 0xBB;
           cksum = mem_checksum(0x0000,0x0000, 0x10);
           gotoxy(1,i+ +);
           printf ("Test Memory Checksum          : %.8X",cksum);
           pause(0);
    }
 drive =  get_drive_info(drv);
 printf("\n\r Drive Information for Drive #%d  letter %.2s", drive, drv);
 result = get_f_name();
 if (result > 0 )
   data_sec = cluster2sector(result);
 else
   data_sec = get_data_sec();
 get_header();
/*  getfat_info(&f_free, clust_req);*/

 if( data_sec < = 0)  _exit(2);
 else
 {
  printBPB(bpb);
  pause(1);
  printDPT(dpt);
  pause(1);
  printJmp();
```

```
    print_regs();
    pause(1);
    printHeader();
    pause(1);
  }

  return (0);
}
```