Five Dollars

Cromemco Multi-User Basic

Instruction Manual

CROMEMCO

Multi-User BASIC

Instruction Manual

CROMEMCO, Inc.
28Ø Bernardo Avenue
Mountain View, CA 94Ø43

Part No. Ø23-ØØ61 November 1979

Table of Contents

Introduction

An Overview

Cromemco's new Multi-User BASIC Operating System includes features which are usually found only on large computers. It is composed of a time-sharing operating system, Multi-User CDOS, and an enhanced version of Cromemco 16K Extended BASIC. The hardware configuration required for the operation of the Multi-User BASIC Software System is shown in chapter 11 of this manual.

The current version includes many new features. Users are no longer restricted to running BASIC. One user can run the assembler, for example, while another is running the FORTRAN compiler, and a third is running BASIC.

The new version allows users to start several jobs from the same terminal by using the Multi-User CDOS command, DETACH. Other new CDOS commands are:

- ATTACH	retrieves a DETACHed job.
- BACKG	allows a job to be run in the background.
- BAS	loads and runs BASIC.
- CHA	is the command entered in order to change diskettes.
- KILL	allows jobs to be terminated from selected terminals.
- LOGOFF	allows users to terminate their sessions.
- MORE	allows users to obtain larger memory allocations.
- PRT:	allows the users to access either a dot-matrix or fully formed character printer with CTRL-P.
- RESCUE	allows users to initialize disabled terminals.

- SET	allows the user of terminal-1 to set the time, date, and adjust the clock rate. Each user may set the number of nulls after each RETURN, an automatic line feed after each RETURN, and a user-selected ERASE character.
- SYS	displays for all users: job number, status (busy or wait), whether BASIC is loaded, which CDOS program is running, how much memory is allocated, terminal number and user name, and the time that user logged on.
- TIME	displays the current time and date.

Since Multi-User CDOS buffers input from the terminal keyboards, users can enter characters before the executing program requests them. This feature, commonly known as "type-ahead", is often very convenient to use.

Multi-User BASIC includes the following attributes in addition to those of 16K Extended BASIC:

- an in line BASIC text editor
- upper and lower case program text
- variable names up to 31 characters long
- line and subroutine names up to 31 characters long
- many new statements and commands such as
  - PROTECT which insures that a file cannot be read from, written to, or erased
  - IF...THEN DO...ELSE...ENDDO
  - COMMON for linking BASIC programs
  - LVAR which lists the variables, functions, and label names of the program together with their current values
  - NOLIST and DELREM which can be used to protect proprietary SAVEd BASIC programs

Getting Started

Turn on the terminals, disk drives, and computer. Set each of the terminals to 9600, 4800, 2400, 1200, 300, 150, or 110 baud. The terminals need not all be set the same. Place the Multi-User System diskette in Disk Drive A. Depress the RETURN key on terminal-1 several times.

Prompts will be issued for the name of the user, the date, and the time.

After the time is set, the following prompt is issued:

Place diskette in Drive A and press any key

This provides the option of exchanging the disk in Drive A before going on.

Prompts are then issued to determine whether or not diskettes are loaded into the remaining drives in the system. All files left open on the diskettes in any of the drives will be closed. This provides a way to rectify the situation when the computer is inadvertently reset with files open.

Additional users may then log on the other terminals.

Log-on for each of these users is accomplished by pushing RETURN several times so that the operating system can detect the baud rate of the terminal. A prompt will then be issued for the user's name.

Write-protecting diskettes is not necessary with Multi-User BASIC. Write-protected diskettes are those 8" disks with a notch exposed and those 5" disks with the notch covered. There is a facility for write-protecting files instead. See the PROTECT command.

Backup

It is strongly recommended that a backup disk be made immediately upon receiving your FDM software package.

User Protection

Multi-User BASIC provides several different kinds of protection for users.

Directory Protection

The operating system maintains directories which contain information about the files stored on the disks. In order to protect this information, locks are provided in the operating system to insure that a user can complete certain kinds of directory accesses, once begun, before another user can begin one.

For example, suppose that after BASIC has begun to execute a CLOSE file statement for a user, the operating system interrupts and switches to a second user. If the second user then attempts to execute a RENAME file statement, the operating system will cancel the turn and not allow further execution time until the CLOSE statement has been completed for the first user.

Protection of File Integrity

There are two different senses in which files can be safeguarded. Multi-User BASIC has the PROTECT command which can be used to specify that a file cannot be erased, written to, or read from. This helps to preserve files from inadvertant misuse.

There is another way in which files need to be protected in multi-user systems which allow more than one user to write to the same file.

Suppose that two users access a file storing charge account information for a retail store in the following sequence.

> User-1 reads the record for account no. 3ØØ and user-2 also reads the same record.
>
> User-1 adds a new charge to the account and writes the record back to the file.
>
> User-2 makes a correction to the address field of the account and updates the file.

The result is that the charge data that user-1 added to the record is lost because it has been over-written by user-2. This is called the mutual exclusion problem.

What is needed is a way to restrict access to files. This is provided by the OPEN statement. A file can be opened by a user in several ways:

(W) exclusive permission to write to the file, but non-exclusive permission to read from it,
(X) exclusive permission to read and write,
(N) a guarantee that no user will write to the file, and non-exclusive permission to read, or
(R) non-exclusive permission to read the file.

The solution to the problem illustrated by the example above is automatic since there is no way for two people to simultaneously have the same file open for writing.

This, however, can present another problem. One user can have a file open for writing for a long time, locking all others out. A way to minimize this problem is to write the BASIC program so that the file is opened only for reading (R) while the user is searching through it. When the user is actually ready to update the file, the BASIC program should open it for reading and writing (W or X) just long enough to re-read the record that is to be altered (to insure that current data is being acted upon) and to write the corrected record back to the file.

In cases where records refer to each other so that updating one record requires updating the other, the file must be open for reading and writing long enough to re-read and update both records.

Protection from Deadlock

When a data base includes more than one file, records in one file may refer to records in another file in the data base.

In this case, updating a record in one file may require updating records in other files also. Therefore, when a user is ready to update a record, the BASIC program should open all the files in the data base for reading and writing (W or X) just long enough to re-read the record and relevant cross-referenced records and to update them.

This introduces the potential for a new problem, however. Suppose that user-1 has file-1 open for writing and user-2 has file-2 open for writing, but both users need both files open for writing to complete the current operation. This is a deadlock. Neither user can proceed.

Multi-User BASIC has a feature which can be used to avoid this dilemma. If a BASIC program concatenates a series of statements from the following list, then the directory will be locked from the time the first statement in the series begins execution until a statement which is not in the list occurs.

```
OPEN
CLOSE
CREATE
PROTECT
ERASE
```

This means that after a user has opened the first file in a list of consecutive files, no other user can open a file further down in the list even if the operating system interrupts and switches to another user.

Deadlock will not occur if the BASIC program includes code such as the following to handle the opening of the data base. We assume that the data base includes files named "file-1", "file-2", and "file-3".

```
  10 Integer False, True, Try'again, Error'number
  20 False=0
  30 True=Not False
   .
   .
   .
1000 *Open'data'base
1010 On Error Goto Data'base'in'use
1020 Open \1\ "file-1", "W"
1030 Open \2\ "file-2", "W"
1040 Open \3\ "file-3", "W"
1045 Rem DO NOT INTERSPERSE ANY STATEMENT
1050 Rem AMONG THE THREE LINES ABOVE.
1060 Rem IF WE HAVE REACHED THIS POINT, THEN WE HAVE
1070 Rem SUCCEEDED IN OPENING ALL THE FILES
1080 Rem IN THE DATA BASE.
1090 Rem RESTORE THE NORMAL ERROR HANDLER.
1100 On Error Goto Normal'error'handler
      .
      .
      .
1200 *Data'base'in'use
1205 Rem IT IS IMPERATIVE THAT WE TRY TO CLOSE
1210 Rem ALL FILES IN THE DATA BASE, EVEN THOUGH
1220 Rem SOME MAY NOT HAVE BEEN OPENED
1230 Rem AT THE TIME THE ERROR OCCURRED.
1235 Rem TEMPORARILY MAKE DO'NOTHING, WHICH DOES
1240 Rem NOTHING AT ALL, THE ERROR HANDLER.
1250 Error'number = Sys(3)
1260 On Error Gosub Do'nothing
1270 Close \1\
1280 Close \2\
1290 Close \3\
1300 Rem RESTORE THE ERROR HANDLER
1310 On Error Goto Normal'error'handler
1320 Rem UNLESS THE ERROR WAS 224
1330 Rem (FILE BUSY), DO NOT TRY AGAIN.
1340 Try'again = False
1350 If Error'number = 224 Then Do
1350 Input"Data Base busy.  Try again?  ",Reply$
1360 If Reply$(0,0)="y"Then Try'again=True
1365 If Reply$(0,0)="Y"Then Try'again=True
1370 Enddo
1380 If Try'again=True Then Goto Open'data'base
      .
      .
      .
1500 *Do'nothing
1510 Return
```

Description of the Operating System

Description of the Operating System

The operating system has control of the computer. When a user logs on, the operating system finds an empty bank of memory and then enters the user into the active queue. The users in this queue take turns executing their programs. Each turn is on the order of tens of milliseconds duration. However, if a user requests input or output which is not ready, the operating system terminates the user's current turn and does not grant another turn until the input or output is ready.

Terminal input and output are serviced by means of interrupts. When a key is pushed on the user's keyboard, the current process in execution is interrupted long enough for the operating system to get the character and put it into the user's terminal buffer. If blocked awaiting this input, the user would take a place in the active queue. When a turn comes around again, the user's program can then fetch the character from the buffer. Terminal output is handled in a similar manner.

Another function of the operating system is to coordinate requests for input from and output to the disks. When a user makes such a request, it is placed into the disk I/O queue, and the user is blocked until the request has been serviced.

The requests in the disk queue are ordered by disk and by track. This minimizes the motion of the disk heads required to service the users.

Attributes of Multi-User CDOS

Type-ahead

Since Multi-User CDOS buffers input from the terminal keyboards, users can enter characters before the executing program requests them. This feature, commonly known as "type-ahead", is often helpful. When using the Screen Editor, for example, one can insert a string of characters and then begin to delete some characters following the string without having to first wait for the screen to be updated.

In case of error, typing CTRL-U will discard all the characters which have not already been utilized by the executing program.

There is a second way of entering characters at the keyboard before the executing program requires them. CDOS and many programs which run under CDOS accept keyboard input one line at a time. The user types the characters of the line and then terminates the line with a carriage return. In Multi-User CDOS, the linefeed key may be used instead of the carriage return key to separate different input lines sent to the program. The linefeed is transformed into a carriage return when it is sent to the executing program. However, none of the characters are sent to the program until an actual carriage return is typed. For example, one could enter the following line followed by a carriage return.

```
ERA *.BAK|BATCH|STAT|ASMB TESTFILE.ABZ|
```

Note that each linefeed is echoed to the terminal screen as a vertical bar "|". After the carriage return is typed, each string of characters followed by a carriage return is sent to CDOS or the executing program as a separate line. When the program receives the line, the line is echoed again. In the current example, the user would then see (assuming we are using disk drive A):

```
A.ERA *.BAK
A.BATCH
Batch version 00.05
-STAT
-ASMB TESTFILE.ABZ
-
```

CDOS Call To Display Job Information

The new CDOS system call ØDFH will return the job number, terminal number and name of the current user. To use it call ØØØ5 with ØDFH in the C-register. The current job number will be returned in the C-register and the terminal number in the B-register. These will be encoded in ASCII. If the job is detached, there will be no terminal number and an asterisk will be returned in the B-register. The DE-register containing the address of the user name in memory. The last character of the name is marked by having its parity bit set.

Multi-User CDOS Commands

command: BACKG

This command can be used to place a job in the background. A background job will only run when all jobs not in the background are waiting for input or output.

BATCH is a CDOS program which is, for all practical purposes, the same as the @ command described in the CDOS instruction manual. A job can be placed in the background by executing a BATCH file which includes the BACKG command as one of its lines. If this is done, then all of the commands which follow the BACKG command in the BATCH file will run in the background. If the command on the last line in the BATCH file completes execution and return is made to CDOS, the foreground mode of operation resumes.

For example, entering the following three lines will cause BASIC to run in the background in the current job until return is made to CDOS by means of the BASIC command BYE.

```
A.BATCH
!BACKG
!BAS SORT.SAV
```

In the above, "A." is the CDOS prompt and "!" is the BATCH prompt.

```
command:  BASic

 format:  BAS

          BAS [file-ref]
```

where:

file-ref is the name of a SAVEd file

The BAS command allows the user to enter BASIC and load and execute a SAVEd BASIC program if desired.

The request to load and execute a BASIC program will be honored upon the user's first entry into BASIC by means of the BAS command. The user may exit BASIC, run CDOS commands only, then return to BASIC via the BAS command. Upon re-entering BASIC in this fashion, the previously executed SAVEd program will still be in memory. If the user runs CDOS programs after leaving BASIC, the previously executed SAVEd program will not remain in memory.

The user will enter BASIC immediately if it has already been loaded. If BASIC has not been loaded and no one is using the memory space in which it resides BASIC will be loaded. The user will be informed:

```
Loading BASIC.
```

Otherwise, the user will be informed:

```
BASIC unavailable.
```

command: CHA

To change a diskette, type CHA followed by a carriage return. After the following prompt:

```
Disk Drive (A,B,C,D)?
```

enter the drive letter.

If there are files open on the drive, the user will be informed which users have opened them. The diskette should NOT be changed.

If there are no files open on the drive, and the diskette is in Drive A, the user will be instructed:

```
Place diskette in Drive A and press any key.
```

(The operating system requires a diskette in Drive A.) If the diskette is in any of the other drives, for example Drive B, the user will be asked:

```
Is there a diskette in Drive B (Y,N)?
```

Answer in the negative if the drive is to be left empty.

This command may be restricted to the users of selected terminals. For further information see the section on Modification of Multi-User BASIC.

command: DETACH

This command can be used to execute several jobs from a single terminal. A user can start a new job by executing a BATCH file which contains the DETACH command as one of its lines. (BATCH is a CDOS program.) When the DETACH command is executed, two things happen:

1. a. The execution of the subsequent lines in the BATCH file continues for the current job.

   b. The current job is detached from the user's terminal. This means that any terminal output is discarded, and if any terminal input is requested, the job waits until the it is again connected to the user's terminal by execution of the ATTACH command.

   c. If the command on the last line in the BATCH file completes execution and return is made to CDOS, the job is terminated.

2. a. A new job is started for the user with a new job number.

   b. The users's terminal is attached to the new job. The CDOS prompt is issued to the terminal screen and the user can enter commands from the keyboard to be executed under the new job number.

For example, entering the first three lines following will cause the current job to assemble the file PRODUCT.Z80 located on Drive A and to place the output files on Drive B. Meanwhile the user will be given the CDOS prompt for entering other commands under a new job.

```
A.BATCH
!DETACH
!ASMB PRODUCT.ABB

A.
```

In the above, the first CDOS prompt "A." and the BATCH prompts "!" are issued under the old job. The last CDOS prompt is issued under the new job.

The following is the same as the last example except that the current job will run in the background and it will include the running of the BASIC program ANALYSIS.SAV after the assembly is finished.

```
A.BATCH
!DETACH
!BACKG
!ASMB PRODUCT.ABB
!BAS ANALYSIS.SAV

A.
```

command: ATTACH

To retrieve a DETACHed job, type ATTACH followed by a RETURN. Enter the number of the job after the following prompt is issued:

```
Job number (1,...,7)?
```

If the requested job is a DETACHed job that the user previously started, the user's current job will be terminated and the user's terminal will be re-ATTACHed to the requested job. Otherwise, the user will continue running the current job.

command: KILL

To terminate a job, type KILL followed by a carriage return. After the following prompt enter the job number:

```
Job number (1,...,7; CR)?
```

The numbers of all current jobs can be determined by using the SYS command. To avoid executing the KILL command, press the carriage return only.

This command should be used with caution on a job which has files open for writing. Such files will be closed before the job is terminated and the last data that was written to the file may be lost.

This command can be restricted to the users of specified terminals. See the Modification of Multi-User BASIC section.

command: LOGOFF

The LOGOFF command may be used by any user to log off that user's job.

command: MORE

To obtain more memory, type MORE followed by a carriage return. If enough memory is available, the user will be assigned 48K and informed:

```
48,887 bytes obtained.
```

Otherwise, the user is informed:

```
Extra memory unavailable.
```

If no other memory is available and nobody is running BASIC, then the memory space between 8ØØØ Hex and CØØØ Hex in which BASIC resides will be assigned to the first user who requests MORE memory. Until this user gives up the extra space by entering the LOGOFF command or the BASic command, any other user who shares the common BASIC memory at 8ØØØ Hex with the first user, and who tries to load BASIC, will be informed:

```
BASIC unavailable.
```

It is possible to configure the system so that many users can simultaneously have extra memory. For details see the Special Configuration portion of the Hardware chapter. With this system configuration a user can load BASIC regardless of whether others users have obtained 48K of memory.

A user who has obtained 48K of memory using the MORE command can, if nobody is running BASIC, get still more memory by entering the MORE command a second time. This is the memory which BASIC uses above CØØØ Hex. If successful, the user will be informed:

```
54,263 bytes obtained.
```

Until this user gives up the space by entering LOGOFF or BAS, no other user can load BASIC, even with the special system configuration. Two requests for MORE memory are required to get this maximum amount of memory in order to limit its use to those who need it.

The second and first uses of the MORE command may be restricted to the users of selected terminals. See the section on Modification of Multi-User BASIC.

command: PRT:

In Multi-User CDOS the printer may be treated as if it were a file. Therefore anything that can be written to a file can also be written to the printer. Cromemco dot matrix printers (models 3703 and 3779) may be referenced by "par:". For example, the BASIC command

```
list "par:"
```

would list the current BASIC program to the printer, and the BASIC command

```
lvar "par:"
```

would list the program's variables to the printer.

The CTRL-P key may also be used to send output to the printer.

If one person is using a printer, anybody else who tries to access the printer is informed that the device is busy (in BASIC, error 224 is returned).

When running BASIC the printer may also be opened as a file so that data can be output to it by means of the PUT or PRINT commands. The following subroutine would output the contents of the string, Error'message$, to the printer:

```
500 *Error'routine
510 Open\1\"prt:"
520 Print\1\
530 Print\1\ Error'message$
540 Print\1\
550 Close\1\
560 Return
```

command: RESCUE

To initialize all terminals which were turned off when the operating system was loaded, type RESCUE followed by a carriage return.

command: SET

The SET command is used to set various parameters. The clock can be adjusted and the date and time set by the user of the System Console. All users can use the SET command to select their alternate erase key, the number of nulls appended to each output of the carriage RETURN, and whether a line feed is automatically appended to each output of a RETURN. To use this command, type SET followed by a RETURN. The following will be displayed:

```
1 Clock rate
2 Date
3 Erase
4 Linefeed
5 Nulls
6 Time

Enter number of desired function (1,...,6):
```

After the function number is typed a prompt will be issued, dependent upon the function, for the desired parameter change. To change the parameter, type the new value. To leave the parameter unchanged, type only a RETURN.

Notes:

1. The Clock Rate may be adjusted by entering 1 as the function number. The system will respond:

   ```
   Faster or Slower (F,S)?
   ```

   Type F for a faster or S for a slower clock rate.

2. The Date may be SET by entering 2 as the function number. The system will respond with three queries:

   ```
   Year (1979,...)?
   Month (1,...12)?
   Day(1,...31)?
   ```

3. Two Erase keys are provided. One is the DELete key. The other Erase key is, by default, the underscore key. This can be changed to suit the user by entering 3 as the function number. The system will respond:

   ```
   Press the desired Erase key:
   ```

   Enter the desired alternate Erase character, followed by a RETURN. The DELete key will still function as an Erase character. Users of 31Ø2 terminals will probably want to change the underscore key to the back-arrow key.

4. A line feed may be added after each output of a carriage return by first entering 4 as the function number. The system will respond:

   ```
   Automatic LF after CRs (Y,N)?
   ```

   Enter Y to enable the added linefeed function.

5. Nulls are needed following carriage returns in some terminals. To set the number of nulls, type 5 as the function number. The system will respond:

   ```
   Number of Nulls after CRs (Ø,...,127)?
   ```

   Enter the number of nulls needed.

6. To SET the time, type 6 as the function number. The system will respond with three queries:

   ```
   Hour (1,...,12)?
   AM or PM (A,P)?
   Minute (Ø,...,59)?
   ```

command: SYS

Type SYS to display the following data for all users who are logged on:

- job number
- job status: WAIT if waiting for input or output, BACKG if running in the background, or RUN if running in the foreground
- load: BAS if BASIC is loaded in the job's memory at 8ØØØH, otherwise the field is blank
- program: the name of the CDOS program which the job is running; the field is blank if no CDOS program is running
- memory: the number of bytes in the job's memory space
- terminal number and name of the user who owns the job; an asterisk appears in place of the terminal number if the job is detached
- time that the user logged on the job

SYS displays a report similar to the following:

```
---------------------------------------------------------------------
| Job | Stat   | Load | Program | Memory | Terminal, User   | Logon  |
---------------------------------------------------------------------
|  1  | Run    |      |         | 54,263 | 1  Joe           | Ø8:25A |
|  2  | wait   |      |         | 48,887 | 2  Mike          | Ø8:3ØA |
---------------------------------------------------------------------
```

This report will update once every two seconds. To terminate the SYS command, press any key (the space bar or the RETURN key are preferable).

command: TIME

The TIME command is used to display the current date and time. This information is derived from a software clock which is driven by an interrupting timer. It has limited accuracy. However, it can be set and adjusted using the SET command.

Multi-User BASIC Names

Format of Numeric Variables

A variable name includes from 1 to 31 characters. The first character must be alphabetic. Each of the remaining characters may be alphabetic, numeric, or the apostrophe (').

Variable names may be entered in upper or lower case characters, or any combination of the two. BASIC will convert the variable name into its own format, which is an initial upper case character followed by all lower case characters.

The following are examples of legal variable names:

```
Interest'rate
AØ
A123
Name1$
Price'to'earnings'ratio
```

Note that the $ character above is not one of the normal characters of the name, but is a suffix which BASIC uses to indicate strings.

Line Names

A statement line may be referenced either by a line number or by an alphanumeric line name. A line name includes from 1 to 31 characters. The first character must be alphabetic. Each of the remaining characters may be alphabetic, numeric, or the apostrophe (').

A line is labeled with a line name by following the line number with an asterisk and then the line name. The asterisk and name is considered an instruction which declares the name of the line. The line may contain additional instructions separated by colons as specified by BASIC syntax.

Label names are used with Goto and Gosub to transfer control from one part of a program to another. They may also be used to reference various parts of a program for Editing, statement Renumbering, etc.

Example:

```
10 *Beginning
 .
 .
50 Gosub Get'data
60 Gosub Process'19
 .
 .
90 Goto Beginning
 .
 .
300 *Get'data : Data'pointer=1
 .
 .
360 Return
400 *Process'19 : Error'flag=0
 .
 .
450 Return
```

Multi-User BASIC Commands

instruction: CLEAR

format: [Ln] CLEAR

where:

Ln	is an optional line number. If Ln is included, the instruction is executed at run time. Otherwise, it is executed immediately.

The CLEAR instruction recovers as much space as it can from deleted lines.

Notes:

1. This instruction is most useful when, upon entering a program, the user receives a space overflow message. It can be entered at this point while no other instruction can be used.

2. CLEAR can also be used after using DELETE and ENTER instructions when performing an overlay, to recover space not needed by the new overlay.

3. CLEAR recovers only part of the available space. For maximum space recovery use the following sequence:

   a. LIST the program to a disk file
   b. SCRatch the user area
   c. ReENTER the program.
   d. SAVE the program.

statement: COMMON storage area

format: Ln COMMON

where:

Ln	is a line number.

The COMMON statement may be used when chaining from programs using RUN statements.

This statement reserves an area for variable storage. The size and contents of this area are determined by the string variables and array variables which have been explicitly DIMensioned since the last Run instruction or since BASIC was loaded. The space is reserved in a byte-by-byte fashon without regard to variable type or length.

A subsequent RUN instruction will initialize all variables except those in the reserved area. The reserved area will be assigned on a byte-by-byte basis to variables as they are DIMensioned in the chained-to program.

The reserved area will cease to be reserved if two RUN instructions are issued without an intervening COMMON instruction.

Note:

1. It is the responsibility of the user to ensure that DIMensioned variable types match between programs. For example, if program A is as follows:

```
200 Dim Name$(25), Interest'rate(10,2)
300 Integer Time'period(100)
400 Common
500 Run "B"
```

then program B, if it has explicitly DIMensioned variables, must DIMension:

a. 26 bytes of string (ASCII) variables

b. 264 bytes of long floating point variables (Ø through 1Ø by Ø through 2 at 8 bytes for each long floating point number)

c. 2Ø2 bytes of INTEGER variables (Ø through 1ØØ at 2 bytes each)

in that order. The names and lengths of the strings and arrays do not matter, as long as the types match. For example, program B might use either of the following sets of statements:

x.
```
1Ø Dim Title$(9),Reference$(15),Table(1Ø,2)
2Ø Integer Time'prior(25),Time'current(24)
3Ø Integer Time'future(49)
4Ø Common
```

y.
```
1Ø DIM Last'name$(1),First'name$(1),Middle'name$(1)
2Ø Dim Initials$(19), Value(1Ø,2)
3Ø Integer Buffer(1ØØ)
4Ø Common
```

Notice that the 27th through 29Øth bytes in the reserved area must be dimensioned similarly in each program if the same indices are to be used in each program. This is true for all two and three dimensional arrays.

command: DELete REMark statements

format: DELREM

DELREM L1

DELREM L1,

DELREM L1,L2

where:

L1	is an optional line number or line name.
	If L1 is omitted, all lines of the program are processed.
	If L1 is the only argument and there is no comma following it, then L1 is the only line processed.
	If L1 is the only argument and the comma is included, L1 through the last line in the program are processed by the command.
L2	is an optional line number or line name which indicates the last line to be processed. If included, it must be preceded by L1 and a comma.

The DELREM command deletes all REMark statements which occur in the lines specified by L1 and L2.

Notes:

1. The deletion of REMark statements from a program will reduce the amount of memory space needed to run it and disk space needed to store it. However, it will have virtually no effect on the execution time of the program.

2. Because deleted REMark statements are not recoverable, the following procedure is recommended for the use of DELREM:

   a. _After_ a program has been debugged, LIST or SAVE a copy of the program to a disk file _before_ using DELREM.

   b. Use DELREM to delete REMark statements.

   c. Save a copy of the new, shorter version of the program in a different file. The file extension NSV could be used to indicate "No remarks, SaVed file".

instruction: EXPAND

format: [Ln] EXPAND str-var[(aexp-1),] aexp-2

where:

Ln	is an optional line number. If Ln is included, the instruction is executed at run time. Otherwise, it is executed immediately.
str-var	is a required string variable.
aexp-1	is an optional arithmetic expression. The default value for aexp-1 is Ø.
aexp-2	is a required arithmetic expression

The EXPAND instruction inserts null characters into a string variable.

Notes:

1. The number of nulls to be inserted is specified by aexp-2. The nulls are inserted before the character specified by aexp-1.

2. Remember that the first character in a string is located at index position Ø.

3. This instruction is very useful for inserting characters into the middle of a string.

Example:

```
100 A$ = "ABEF"
110 EXPAND A$(2),2
120 A$(2,3) = "CD"
130 PRINT A$
RUN
ABCDEF
```

statement: IF...THEN...DO...ELSE

```
format:  Ln IF exp THEN DO [:program instruction(s)]
           .
           .
           .
         [program instruction(s)]
           .
           .
           .
         [Ln ELSE [:program instruction(s)]]
           .
           .
           .
         [program instruction(s)]
           .
           .
           .
         Ln ENDDO
```

where:

Ln	are line numbers
exp	is a relational or arithmetic expression.

If exp evaluates to true (not equal to zero), then the program instructions following DO are executed. If, in addition, ELSE is coded, ELSE will transfer control to the ENDDO statement.

If exp evaluates to false (equal to zero), then control is transferred to ELSE (if ELSE is used) or to ENDDO (if ELSE is not used).

Example:

The following program will request a line of text from the user, and then count the number of words and non-blank characters in that line. The average number of characters per word will be computed and displayed and then the user will be prompted for another line of text. Entering a RETURN in response to the request for input will terminate the program.

```
100    Dim Buffer$(100)
110    Dim Blank$(0) : Blank$=" "
120    Rem Initialize blank character flag
130    Rem            word counter
140    Rem            character counter
150    Last'char'was'blank=0
160    Number'of'words=1
170    Number'of'char=0
180    Rem Prompt user
190    Input"Enter line of text: ",Buffer$
200    Rem Check for user termination (null buffer)
210    Length'of'buffer=Len(Buffer$)
220    If Length'of'buffer=0 Then,goto 510
230    Rem
240    Rem
250    Rem  Loop through buffer
260      For Index=0 To Length'of'buffer-1
270      Rem Check for blank character
280      If Buffer$(Index,-1)=Blank$ Then Do
290        Rem Check if the last character was a blank
300        If Last'char'was'blank=0 Then Do
310          Rem If not, increment word counter and
320          Rem set flag
330          Number'of'words=Number'of'words+1
340          Last'char'was'blank=1
350          Enddo
360        Rem If not a blank, increment character counter
370        Rem and reset flag
380        Else
390        Number'of'char=Number'of'char+1
400        Last'char'was'blank=0
410        Enddo
420      Next Index
430    Rem
440    Rem
450    Rem Display results and return for another user entry
460    @"Number of non-blank characters is ";Number'of'char
470    @"Number of words is ";Number'of'words
480    Avg=Number'of'char/Number'of'words
490    @"Average number of characters per word is ";Avg
500    Goto 150
510    End
```

instruction: List VARiables

format: [Ln] LVAR

[Ln] LVAR file-ref

where:

Ln	is an optional line number. If Ln is included, the instruction is executed at run time. Otherwise it is executed immediately.
file-ref	is an optional string variable or string literal file-reference denoting the destination of the list of variables. If omitted, the listing is displayed on the console.

The LVAR instruction lists all variables, functions, procedures, and alphanumeric line names. The abbreviations used in the list are:

INT	integer variable
SFP	short floating point variable
LFP	long floating point variable
LBL	alphanumeric label, line name
FUN	function name
PROC	procedure name
$	string variable
(*)	list or matrix

The current value of each scalar arithmetic variable is also displayed.

Notes:

1. If a variable, function, procedure name or line name is used in a BASIC program, and then the line containing that item is deleted, the item will still appear in the list produced by LVAR. The list can be updated by LISTing the program to a file, SCRatching the User Area, ENTERing the program, and then using LVAR.

2. If a line name, procedure name, or function has been defined in the program, its line number will be printed after its type. If a line name, procedure name, or function has been used (e.g., Goto Start'over) but not yet defined, no entry will follow the type.

command: NOLIST

format-1: NOLIST

format-2: NOLIST L1

format-3: NOLIST L1,

format-4: NOLIST L1,L2

where:

L1	is an optional line number or label name. If L1 is omitted, all lines of the program are processed. If L1 is the only argument and the comma is missing, then L1 is the only line processed. If L1 is the only argument and the comma is included, L1 through the last line in the program are processed by the instruction.
L2	is an optional line number or label name. L2 indicates the last line to be processed. It must be preceded by L1 and a comma.

The NOLIST command allows the programmer the option of keeping part or all of a BASIC file confidential. After the execution of the NOLIST command, lines L1 through L2 cannot ever be listed.

Notes:

1. Note that both line numbers and alphanumeric label names are considered label names.

2. Because NOLISTed lines can never be listed, the following procedure is recommended for the use of NOLIST:

   a. <u>After</u> a program has been debugged, SAVE or LIST a copy of the program to a disk file <u>before</u> using NOLIST.

   b. Use NOLIST to protect those parts of the program which are confidential.

   c. SAVE a copy of the NOLISTed version of the program in a different file. (One could use the file extension "PUB" to indicate that it is a PUBlic file.)

3. This command protects only those lines present at the time the command is executed. If, after the NOLIST command has been executed, new lines are entered, they will not be protected from listing even if they replace already NOLISTed lines.

instruction: OPEN file

format: [Ln] OPEN\f[,r,t]\svar-1[,svar-2]

where:

Ln	is an optional line number. If Ln is included, the instruction is executed at run time. Otherwise, it is executed immediately.
f	is an arithmetic expression. f specifies the file channel number selected by the user. The value of f ranges from one to a maximum which is installation dependent. The number is used to identify the file elsewhere in the file.
r	is an optional arithmetic expression. r specifies the file record size. The value of r ranges from 1 to 32767. Its default value is 128.
t	is an optional arithmetic expression. t specifies how the file may be written to or read from.
svar-1	is the name of the file.
svar-2	is an optional string expression. svar-2 specifies how the file may be written to or read from.

Notes:

1. Either t or svar-2 may be used to specify how the file may be written to or read from. If there is a conflict between t and svar-2, svar-2 takes precedence. The value of t is given by the following table.

3	exclusive write, non-exclusive read
2	exclusive write, exclusive read
Ø	no writing by any user, non-exclusive read
1	non-exclusive read

The first upper-case letter in svar-2 which matches a letter in the following list determines whether the user is requesting permission to write or read the file, or both, and whether other users are to be excluded from the file.

W	exclusive write, non-exclusive read
X	exclusive write, exclusive read
N	no writing by any user, non-exclusive read
R	non-exclusive read

The default value of svar-2 is "W".

instruction: PROTECT file

format: [Ln] PROTECT svar-1, svar-2

where:

Ln	is an optional line number.
	If Ln is included, the instruction is executed at run time. Otherwise, it is executed immediately.
svar-1	is the name of the file.
svar-2	is a string expression.
	svar-2 specifies which protection attributes the file is to have.

Notes:

1. The file is assigned all those attributes corresponding to the upper-case letters from the following list which svar-2 contains.

E	erase-protected
W	write-protected
R	read-protected

3. If svar-2 contains the character +, then the attributes specified by svar-2 are added to those the file already has. Otherwise, they replace them.

4. If svar-2 is the empty string (indicated by a pair of double quotes with nothing in between), then all of the attributes are removed from the file.

5. All write-protected files are also erase-protected.

<u>BASIC</u> <u>Editor</u>

CROMEMCO In-Line BASIC Editor

commands: CHANGE
EDIT
FIND

format:

```
command

command L1

command L1,

command L1,L2
```

where:

L1	is an optional line number or name.
	If L1 is omitted, all lines of the program are processed.
	If L1 is the only argument and there is no comma following it, then L1 is the only line processed.
	If L1 is the only argument and the comma is included, L1 through the last line in the program are processed by the command.
L2	is an optional line number or name which indicates the last line to be processed. If included, it must be preceded by L1 and a comma.

Notes:

1. When using terminals in uppercase mode, BASIC should be set so that it lists in uppercase by entering the command:

```
SET 6,1
```

   This is because the Find command will not find the given string unless it agrees exactly with what is typed.

EDIT

Edit is one of the three commands which are collectively called the Cromemco In-Line BASIC Editor. The other two are Change and Find. Edit lists lines of code one at a time, as specified by L1 and L2. Each line to be Edited is preceded by a dash (-) and followed on the next line by a colon (:) prompt. In response to the prompt, the user may type:

1. a RETURN, which leaves the line unchanged and displays the next line for Editing, or

2. a series of spaces, which will position the cursor under the character which is to be edited.

   At this point, the user can give the Editor the following commands:

   a. D (Delete) - deletes the character above the cursor. Several deletions can be made on one line.

   b. I (Insert) - inserts the string which follows. An insertion can follow one or more deletions.

   c. K (Kill) - deletes the rest of the line from the current position of the cursor.

Pressing RETURN causes the command line to be executed. The edited line is then typed out and a prompt is given for further changes.

When the line has been changed to the user's satisfaction, a RETURN should be pressed again in response to the prompt for additional corrections. If the Editor goes on to the next line but the user wishes to stop editing at theis time, ESCape should be pressed. This will not affect any changes made except those of the currently displayed line.

FIND

Find is another of the interactive Editor commands. Find will locate all occurrences of a string within the set of lines specified by L1 and L2. After the find command is given, the Editor will prompt with:

```
FIND:
```

The user should then enter the string to be located, followed by a RETURN. If a RETURN is entered immediately following the prompt, the Editor will return control to the BASIC monitor. Each line containing the string will be printed out with a pointer below the line indicating the occurrence of the specified string.

CHANGE

The Change command will replace all occurrences of a string within a set of lines specified by L1 and L2. After the Change command is given, the Editor will prompt with:

```
FROM:
```

In response, the user should enter the string to be replaced followed by a RETURN. The Editor will then prompt with:

```
TO:
```

The user should respond with the replacement string. The Editor will then print the first line containing the string with a pointer below the line indicating the location of the string. The user can then input:

1. a RETURN only to reject the change at that location and proceed to the next occurrence,

2. the letter (C) to accept the change and go on,

3. an asterisk (*) to accept all changes from that point on, or

4. ESCape to abort the process.

Please note that for the Change and Find commands the desired string must be entered exactly as it is LISTed, including upper and lower case, or the Editor will not be able to find it.

Multi-User BASIC Error Messages

Multi-User BASIC has several error messages in addition to those listed in the 16K Extended BASIC Instruction Manual.

220 - FILE ERASE-PROTECTED -	An attempt was made to erase a file that had been erase-protected by the BASIC instruction PROTECT.
222 - FILE WRITE-PROTECTED -	An attempt was made to write to file that had been write-protected by the BASIC instruction PROTECT.
223 - FILE READ-PROTECTED -	An attempt was made to read a file that had been read-protected by the BASIC instruction PROTECT.
224 - FILE BEING USED -	An attempt was made to access a file that was being used by another user.
225 - TOO MANY OPEN FILES -	An attempt was made to open more than the maximum of 16 files. Each user is allowed to open up to 16 files.

The function SYS(3) returns the number of the last runtime error which occurred.

<u>Modification of Multi-User BASIC</u>

Through DEBUG modifications of BOOT.COM it is possible to change the CDOS commands and programs that run upon logging on and to restrict users from using CDOS programs and the BASIC MORE, CHA, and machine level instructions.

Customizing User Startup Command Lines

When each user logs on, a series of commands is executed. These commands are contained in a command line which is stored in the file BOOT.COM. There are seven command lines in this file, one for each user. Each command line is up to twenty bytes long and consists of ASCII code. Each command line must be terminated with a byte of ØØ. This section may be modified to run other CDOS programs and commands. The DEBUG address of the command line for user-1 is located at 312 hexadecimal. The command lines for other users will be found following that of user-1.

The following is an example describing how to modify this command line. First, in CDOS we type:

```
debug boot.com
```

The response will be:

```
DEBUG VERSION ØØ.Ø9
NEXT = 7DØØ

-
```

At the dash prompt, enter:

```
dm 312 s2
```

This will display two bytes from which the address in hexadecimal of the command line for user-1 may determined by reversing their order:

```
Ø312 F3 38
```

To display the contents of the command line, at the DEBUG prompt "-" type:

```
dm 38f3 s2Ø
```

```
38F3   53 59 59 ØD ØD 42 41 53-2Ø 53 54 41 52 54 55 5Ø  SYS..BAS.STARTUP
39Ø3   2E 53 41 56 ØD ØØ FF FF-FF FF FF FF FF FF FF FF  .SAV............
```

To change the command line, at the prompt type:

```
sm 38f3
```

The response will be:

```
38F3 38F3' 53
```

On this line type in the new command line. The entries should only be CDOS commands and programs. They should be enclosed in single quotes. Entries should be separated from one another by ØD with no quotes. 'SYS' must be followed by ØD ØD unless it is the last program to be run in the command line. The last entry must be terminated by a ØD and ØØ. Remember that you only have twenty bytes for this user. To have the command line become SYS, DIR, and STAT type on the displayed line:

```
38F3 38F3' 53 'SYS' ØD ØD 'DIR' ØD 'STAT' ØD ØØ
39Ø2 39Ø2' 5Ø  .
```

In the example above, a carriage return was entered after ØØ on the 38F3 line. A period was entered on the 39Ø2 line to discontinue the entry of changes. To display the new command line, once again type dm 38f3 s2Ø. The response will be:

```
38F3   53 59 59 ØD ØD 44 49 52-ØD 53 54 41 54 ØD ØØ 5Ø  SYS..DIR.STAT..P
39Ø3   2E 53 41 56 ØD ØØ FF FF-FF FF FF FF FF FF FF FF  .SAV............
```

It is acceptable to leave in characters after the first ØØ. Characters which appear after ØØ will be ignored.

Up until this point, we have been discussing the command line for user-1 only. To modify the command line for other users we will first need to determine the address of their command line. This may be done by adding increments of 2ØH to the address for user-1.

User	Number to be added
2	2ØH
3	4ØH
4	6ØH
5	8ØH
6	AØH
7	CØH

This addition may be done in DEBUG. To add 8ØH to 38F3H, enter:

```
h 38f3 8Ø
```

at the dash prompt. The response will be:

```
3973 3773
```

where the first number is the sum and the second is the difference.

When all of the user command lines that have needed modification have been changed, at the prompt type:

```
gØ
```

In order to save these modifications, the new BOOT.COM must be SAVEd. This is done by first determining the length in pages. This is accomplished by typing:

```
dir boot.com
```

at the CDOS prompt. The length of the file will be displayed in kilobytes. This may be converted to pages by multiplying the number of kilobytes by four. To SAVE the new BOOT.COM file type:

```
save boot.com #
```

where # is the number of pages.

Restricting Access to CDOS Programs

Users may be kept from running CDOS programs, suchas STAT and ASMB, and restricted to the use of CDOS commands, suchas DIR and BAS. This is done with a DEBUG modification of BOOT.COM. The byte at DEBUG address 3ØCH contains this information. If a bit is set to 1, a terminal will be disallowed the access to CDOS programs. The following table shows the relationship between bits set and terminals disabled.

Terminal	Bit set
1	Ø
2	1
3	2
4	3
5	4
6	5
7	6

The byte at DEBUG address 3ØCH in binary is initially ØØØØ ØØØØ. To disable terminals 3, 5, and 6 we would set bits 2, 4, and 5. The byte would now be ØØ11 Ø1ØØ. In hexadecimal this is 34. In DEBUG we would substitute memory for the byte at address 3ØCH:

```
-sm 3Øc
Ø3ØC ØØ 34
Ø3ØD FE .
-dm 3Øc sl
Ø3ØC 34
```

Restricting Use of the CHA Command

Users may be disallowed the capability of using the CHA command. This is also accomplished by a DEBUG modification of BOOT.COM. The byte at DEBUG address 3ØDH contains this information. The table in the previous section applies to this section also. The initial setting of this byte is FE hexadecimal which allows only user-1 to use the CHA command.

Restricting Use of the MORE Command

The second use of the MORE command may be disabled for selected users. To restrict users to a maximum memory allotment of 48K, modify the byte at DEBUG address 30FH of BOOT.COM in the manner previously described. The initial setting of this byte is FE hexadecimal which allows only user-1 the second use of the MORE command.

The first use of the MORE command may also be disabled. Users may be restricted to a maximum memory allotment of 32K by a DEBUG modification of BOOT.COM. The byte at DEBUG address 30EH contains this information. This byte is initially 00 hexadecimal so that all users have the first use of the MORE command.

Restricting Use of the KILL Command

Users may be disallowed the capability of using the KILL command. This is accomplished by modifying the byte at DEBUG address 310H of BOOT.COM in the manner previously described. The initial setting of this byte is FE hexadecimal so that only user-1 can use the KILL command.

Restricting Use of Machine Level BASIC Instructions

The use of the OUT, INP, POKE, PEEK, and USR functions and instructions can be restricted in a multi-user environment because with them a user could do harm to the operation of the whole system. On the other hand, they are powerful commands which may be useful in selected cases.

Beginning at DEBUG location 314H of BOOT.COM is the 2-byte address of the byte that will determine accessibility of machine level BASIC instructions. If this byte is C9 all users may access these BASIC instructions. To prohibit all users except user-1 from using these instructions substitute 00 for C9.

If this is done Multi-User BASIC allows only the user who has control of the system diskette to write programs which use these commands. However, any user can LOAD and run a program which has been previously SAVEd, even one which includes these commands.

If the user of terminal-1 wishes to write a program including these commands, the previously mentioned DEBUG address change should have been made and "LOCK" must be entered as the user name when the system is brought up. This prevents additional users from logging on. After the program is written and SAVEd, the user can LOGOFF and log on again by pressing the RETURN key several times. Other users can then log on the system.

Disk Drive Configuration

The Multi-User system can accommodate up to four floppy disk drives. The file BOOT.COM contains a disk table which specifies the drive types. This table is located at DEBUG address 3Ø4H. The standard system is shipped configured for two dual 8" floppies (four drives in all). The values in the disk table for this are:

DEBUG Address	Contents	Disk Drive
3Ø4	B6H	A
3Ø5	ØØH	A
3Ø6	B2H	B
3Ø7	ØØH	B
3Ø8	B6H	C
3Ø9	ØØH	C
3ØA	B2H	D
3ØB	ØØH	D

To use a 5" floppy disk drive, change the contents of the first byte for that drive to 28H. If the system does not include a particular disk drive, change the contents to ØØ. For example, if drive C is to be a 5" floppy and there is no drive D, DEBUG location 3Ø8H should contain 28 and location 3ØAH should contain ØØ.

Hardware Configuration

Cards included in the basic two-user system are:

1	ZPU processor card
1	4FDC Disk Controller
2	64KZ R/W Memory
1	TU-ART Digital Interface
1	PRI Printer Interface

The 64KZ is the ideal memory card for use in a Cromemco Multi-User BASIC system. Upon logging on, each user is assigned a job number. This job number is is one greater than the memory bank number in which the job runs, i.e. job-1 runs in bank-Ø. Each user will be allocated 32K of read/write memory addressed at location ØØØØH. 16KZs may also be used and each user may then be allocated either 16K or 32 K starting at address ØØØØH. The operating system requires 64K of read/write memory. Of the 64K, the 32K from address 8ØØØH to FFFFH is assigned to all memory banks, Ø - 7; this area contains the BASIC interpreter and part of the operating system. The remaining 32K, addressed from ØØØØH to 7FFFH, is assigned to memory bank 7 only. This memory area currently contains most of the system disk and terminal I/O routines.

<u>System RAM Cards</u>

Either half of two 64KZ cards or four 16KZ RAM cards are required for the operating system and BASIC. These are configured as follows:

64KZ RAM Cards

64KZ Card	Actual Address	Address Switches	Reset Switches	Bank Switches
2	ØØØØH	A15-B Ø	Reset B off	7 on, others off
1	8ØØØH	A15-B 1	Reset B off	all on

16KZ RAM Cards

Actual Address	Address Switches 15	14	Bank Switches
ØØØØH	off	off	Ø - 6 off, 7 on
4ØØØH	off	on	Ø - 6 off, 7 on
8ØØØH	on	off	all on
CØØØH	on	on	Ø off, 1 - 7 on

The CØØØH card is the 16KZ card that has been jumpered to be turned off on reset.

<u>User RAM Cards</u>

Each user must have either one half of a 64KZ or a 16KZ RAM card at address Ø (address switch 15 off, 14 off). The actual address for each memory card is ØØØØH. If 16KZ cards are used, the user may also have a second 16KZ card at address 4ØØØH (switch 15 off, 14 on). The bank settings for RAM cards are given by the following table.

64KZ Cards

User	64KZ Card	Address Switches	Bank Switches
1	1	A15-A off	Ø on, all others off
2	2	A15-A off	1 on, all others off
3	3	A15-A off	2 on, all others off
4	3	A15-B off	3 on, all others off
5	4	A15-A off	4 on, all others off
6	4	A15-B off	5 on, all others off
7	5	A15-A off	6 on, all others off

The reset switch for user 1 should be set in. All other reset switches should be set out.

16KZ Cards

User	Bank Switches
1	Ø on, all others off
2	1 on, all others off
3	2 on, all others off
4	3 on, all others off
5	4 on, all others off
6	5 on, all others off
7	6 on, all others off

4FDC Disk Controller Card

The Cromemco 4FDC disk controller interfaces 5" and 8" floppy disk to the S-1ØØ bus used in Cromemco computers. One of the features of the 4FDC is a serial I/O channel with software selectable baud rates ranging from 11Ø baud to 76,8ØØ baud. The 4FDC also has a 1K resident 27Ø8 ROM pre-programmed with Cromemco's RDOS ROM-resident Disk Operating System.

Set switch 1 off, switches 2, 3, and 4 on to prepare the card for use with Multi-User CDOS and BASIC.

The 4FDC services Terminal-1 through connector J4.

TU-ART Digital Interface Cards

A Multi-User System must include from one to three TU-ARTs.

Each TU-ART can service two terminals. The even-numbered terminal is serviced through connector J4 (SERIAL A) and the odd-numbered terminal through J5 (SERIAL B).

TU-ART Switches

TU-ART	Terminals	3	4	5	6	7	8	9	1Ø
1	2 & 3	Ø	1	Ø	Ø	1	Ø	1	Ø
2	4 & 5	Ø	1	1	Ø	1	1	1	Ø
3	6 & 7	Ø	Ø	Ø	1	1	Ø	Ø	1

Note that for TU-ART switches, "Ø" indicates that the switch should be on.

Switches 1 and 2 on all TU-ARTs should be off.

If the system includes a Cromemco 37Ø3 or 3779, printer it should be connected to TU-ART 1 through connector J3 (PARALLEL B). The standard PRI non-interrupt interface card should not be used in a multi-user system.

Interrupt Priority Cable

Use the Priority Cable to connect Priority OUT of the 4FDC (connector J1, pin 3) to Priority IN of TU-ART 1 (connector J1). If the system includes more than a single TU-ART, connect Priority OUT of TU-ART 1 (J1) to Priority IN of TU-ART 2 (J1). If the system includes a third TU-ART, connect Priority OUT of TU-ART 2 (J1) to Priority IN of TU-ART 3 (J1).

Special Configuration

In the standard configuration, all users share memory at address 8ØØØH in which BASIC can be loaded. In order to allow more than one user to obtain 48K of memory in which to run CDOS programs, some of the memory banks should have private memory at 8ØØØH. This requires the use of some 16KZ cards since all banks must share memory addressed at CØØØH. For example, with the following seven-user memory configuration:

	Address	Bank(s)	Jobs
16KZ	CØØØH	Ø - 7	all (system card)
16KZ	8ØØØH	Ø,1,2	1,2,3
16KZ	8ØØØH	3	4
16KZ	8ØØØH	4	5
16KZ	8ØØØH	5	6
16KZ	8ØØØH	6	7
1/2-64KZ	ØØØØH	7	all (system card)
1/2-64KZ	ØØØØH	Ø	1
1/2-64KZ	ØØØØH	1	2
1/2-64KZ	ØØØØH	2	3
1/2-64KZ	ØØØØH	3	4
1/2-64KZ	ØØØØH	4	5
1/2-64KZ	ØØØØH	5	6
1/2-64KZ	ØØØØH	6	7

four banks, 3 - 6, are private banks since each shares memory at 8ØØØH with no other banks. This means that as many as four users may each obtain 48K of memory by using the MORE command. At the same time, three other users could run in banks Ø - 2 with 32K of memory apiece since these banks share memory at 8ØØØH. Each of these three users could run either a CDOS program in 32K of memory or a BASIC program which uses up to 32K of memory. If BASIC is loaded, it resides at 8ØØØH.

I N D E X