WANG LABORATORIES, INC.

Benincasa, M. 3118

TO: 2200 Series User

From: Wang Laboratories 2200 Development

Subject: New releases of the 2200 MVP and 2200 VP Operating Systems

Date:June 15,1981

Dear 2200 User:

Wang is pleased to supply you with the latest release of the 2200 Series Operating System. Along with the System Diskette should be enclosed a Marketing Release. This describes the changes and features of this release of the Operating System. We are also enclosing an updated System Utilities User Manual that describes the operation and use of all of the utilities that reside on the the System Disk.

If you have questions or problems with this new release of the Operating System, please call your local Wang Analyst.

2460E

ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851 • (617) 459-5000 • TWX 710-343-6769 • TELEX 94-7421

WANG LABORATORIES, INC

Computers

MARKETING RELEASE

TO DISTRIBUTION	PUBLICATION #
FROM 2200 PRODUCT DEVELOPMENT	DATE JUNE 1981
SUBJECT MVP MULTI-USER OPERATING SYSTEM RELEASE 2.2	REORDER FROM:
THIS RELEASE SUPERSEDES:	DESTROY SUPERSEDED INFORMATION ☐ YES ☐ NO

Wang Laboratories is pleased to announce release 2.2 of the MVP Multi-user Operating System. This release includes several new features and enhancements to earlier versions of the MVP Multi-user Operating System.

SUMMARY OF FEATURES

- *The Operating System Loading Sequence has been modified to allow for easier use of the system diagnostics.*
- *Many of the System Utilites have been improved to run faster, and to be more user oriented. Among these utilities are @BACKUP, @RECOVER, and @MOVEFIL.*
- *Two new BASIC-2 Statements have been added to the operating system. These are #ID and $ALERT.*

REQUIREMENTS

- *All 2200MVP and 2200LVP sytems should be upgraded to Release 2.2 of the MVP Multi-user Operating Systems.*

ORDERING INFORMATION

Order through Software Distribution.

	2200 MVP	*2200 LVP*
Package Number	*195-0049-3*	*195-2162-5*
Diskette Number	*701-2294N*	*704-0002C*

AVAILABILITY

In Stock, allow one week for delivery.

PRICE

No charge to 2200MVP, 2200LVP users.

SUPPORT

This is a Category 1, Wang-Supported product. Any suspected errors or anomalies found in this package should be documented and forwarded to Wang Laboratories via the local district analyst.

MVP (Multi-User) BASIC-2 RELEASE 2.2

Release 2.2 of MVP BASIC-2 replaces all previous MVP BASIC-2 releases. This release provides all 2200 MVP and 2200 LVP systems with several new features and corrects all known system anomalies. The system platter includes the MVP (multi-programming) Operating System and BASIC-3 language processor, system diagnostics and several utilities.

I. System Utilities

Significant changes have been made to several of the utilities. The 2200 BASIC Utilities Manual (preliminary copy attached) should be consulted for an up-to-date description.

The utilities described below can be accessed by entering LOAD RUN (RETURN). A menu will be loaded providing access to the utilities. Certain utilities are for particular devices and do not have a function in all 2200 configurations.

@GENPART: Configuration Definition and Execution

This utility creates, saves, and executes system configurations which divide the 2200 resources among the system users. (See 2200 MVP Introductory Manual or 2200 LVP Introductory Manual).

@PSTAT: Partition Status

This program displays the current status of each partition in the current MVP configuration. (See $PSTAT in the BASIC-2 Language Reference Manual.)

@INSTALL:

This utility moves the system files from the release diskette onto a destination platter of the user's choice. Moving the configuration file is optional.

@MENU: Program Menu

@MENU provides a menu structure for program selection. Multiple levels of menu can be set up with each successive screen displaying the next menu mode. (See program REMarks for customization). The system platter contains a START program that merely overlays in @MENU.

@FORMAT: Format Disk Platter

This program formats software formattable disk platters, such as 2260C, 2260BC, 2280 platters, dual sided double density diskettes and LVP fixed platters. (Refer to the appropriate disk reference manual for detailed formatting information).

@DAVFU: Vertical Format Control

This utility defines printer vertical format control sequences. (See the appropriate printer reference manuals). This utility was previously named @2273VFU.

@BACKUP: Platter Backup

This program provides a multi-volume platter backup capability. It is particularly useful for systems such as 2200 LVP's, which have different size fixed and removable platters.

@RECOVER: Backup Recovery

This is the companion recovery program for @BACKUP. The entire platter, active files only, or selected files can be recovered from the backup platter(s).

@MOVEFIL: Move File

This program moves selected files or all active files from one platter to another. If files are too large, multiple platters will be used. File data can be recorded in 3741 format for software transport between VP/MVP and SVP/LVP systems.

@MRTIAN: System Space Game

This program allows a user to play on the latest space war games.

II. System Changes for MVP.

Listed below are the system changes since Release 2.1. The following files on the system platter differ from Release 2.1

@SYSMVPB	-	Menu Mode Data (name changed from .SYSMVPB)
@@	-	Microcoded Menu Program (new)
@MRTIAN	-	System Game (new)
@MOVE	-	Move System Files (removed)
@INSTALL	-	Move Systems File (new)
@DAVFU	-	Vertical Format Control (replaces @2273VFU)
@ ACKUP	-	Platter Backup (improved)
@BOOT	-	Menu Node Bootstrap (new)
@MOVEFIL	-	Move File (improved)
@MVP		MVP OS and BASIC-2 Language Processor (new)
@MENU	-	Menu Utility (improved)
@OG	-	System Diagnostics Menu (new)
@STARTD	-	File used by @MENU (name changed from .STARTD)
@HELP	-	System Help Documentation (removed)

Wang Multi-user BASIC-2 for 2200 MVP or 2200 LVP
Release 2.2
Corrected Anomalies

1. The work buffer used during the execution of an INPUT statement was not properly freed when an error X73 or X75 was fielded with :ERROR. Over a long period of time, this could lead to a memory full error, A01.

2. The 2200 T BASIC form of the COPY statement (COPY FR) has been corrected to work properly with the 2200 SVP diskette controller and to use the optimal copying strategy when used with the 2280 disk drive.

3. The sector address of the sector in error returned during the execution of a VERIFY statement was incorrect if a disk hardware error (I90 or I91) was encountered.

4. $RELEASE TERMINAL TO, name. did not correctly handle the situation where two or more global partitions in different memory banks had the same name. $RELEASE TERMINAL TO, name. now attaches the terminal to the lowest numbered partition with the specified name that is available to the terminal from which the $RELEASE TERMINAL statement is executed. A partition is available to a terminal if it is currently assigned to the terminal or if it is assigned to the null terminal, terminal 0.

5. Global variables may now be used to dynamically dimension arrays. (i.e., DIM A$ (@X)) As with all dynamic dimensions in BASIC-2, the global variable used to dimension an array or specify the length of an alpha variable must be a scalar and must be common. Additionally, the global variable must reside within the partition being resolved. This is consistent with program resolution (RUN or LOAD) clearing the global partition pointer.

6. HALT/STOP and CONTINUE did not always function properly within a SELECT ON interrupt handling subroutine.

A. Re-definition of Existing Features

1. The INPUT statement can no longer generate the unrecoverable error, S23, during program execution. The only errors INPUT can now generate during execution are X73 and X75, both of which are recoverable with :ERROR.

2. $BREAK and $BREAK! may now be interrupted by the occurrence of a programmable (SELECT ON) interrupt. When an interrupt occurs, execution proceeds to the interrupt handling subroutine, execution continues with the statement that is either interrupted or terminated by the occurrence of a user-defined interrupt. In all other cases, interrupts may occur only after the completion of the currently executing statement and before beginning the next.

3. The value of the global partition pointer is saved upon entry to a SELECT ON interrupt handling subroutine, and restored upon exit. Interrupt handling subroutines may thus execute SELECT @PART statements without disturbing the global pointer set up by the interrupted program. Interrupt handlers should always execute a SELECT @PART statement if global variables or global subroutines are to be used; the value of the global partition pointer should be considered undefined upon entry to the interrupt handler.

B. New Features

1. A new numeric function, #ID, returns the CPU identification number. Each 2200 CPU is assigned a number (a random integer between 1 and 65535) at the time of manufacture. Machines produced prior to the implementation of this feature return a value of 0, but such machines can be field upgraded to have non-zero # ID's. CPU ID's are not guaranteed to be unique, but it is highly unlikely two given machines will have the same number.

   This function allows software to tell one CPU from another. The ability to distinguish one CPU from another is useful in restricting software to specific installations and in telling one CPU from another when disk multiplexers are used.

   An application program can inhibit program execution if an unknown or unacceptable identification number is read.

   In one or more critical sections of the application software (e.g., menu, key routine) a check can be performed to ascertain that the software is executing on the prescribed machine. The check would be of the type:

```
IF #ID        machine-id# THEN STOP "!@?$"
```

   Of course, the section(s) of code performing this check must be scramble protected (i.e., SAVE!) in order to maintain security. Scramble protect inhibits program examination on disk and after loading.

2. Partitions may now interrupt each other's execution with the $ALERT statement. This provides a means for one partition to signal the other that an infrequent event has occurred. Use of $ALERT is much more efficient of CPU and I/O time than repeated checking of disk files or global variables. $ALERT interrupts are defined and fielded according to the same rules as other programmable interrupts. (See Chapter 8 of the BASIC-2 Language Reference Manual.)

$ALERT

```
! General Form:                                                          !
!                                                                        !
!     $ALERT partition                                                   !
!                                                                        !
! Where: partition is expression whose truncated value is 1 - 16         !
!        giving partition to interrupt.                                  !
!                                                                        !
```

The $ALERT statement generates an interrupt to the specified partition. In order for the interrupt to have any effect, the $ALERTed partition must execute a SELECT ON ALERT statement defining that ALERT interrupts are to be fielded and indicating a subroutine that is to be executed when an ALERT interrupt occurs. (See SELECT ON in Chapter 8 of the BASIC-2 Language Reference Manual.)

$ALERT is used to inform another partition that a disk file or global variable should be polled. Using $ALERT consumes much less CPU or disk I/O time than repeatedly checking disk files or global variables for the occurrence of an infrequent change.

When a $ALERT interrupt is acknowledged, the programmer knows that at least one $ALERT statement specifying the program's partition has been executed by some partition on the system since the last occurrence of a $ALERT interrupt, LOAD, CLEAR, or RUN. The programmer does not know which partition executed the $ALERT or whether several $ALERTs have been executed since the last $ALERT interrupt was acknowledged.

Examples:

```
100 $ALERT 5
150 $ALERT T(N)
```

Example using $ALERT

A. The following is a portion of a program running in a foreground partition. It collects data from an operator at a terminal and stores it into a disk file. After storing each record it alerts a background program, using $ALERT. The background program is shown in 'B'.

```
0100        REM GET NEW DATA
   :        LINPUT "ENTER CODE",-C$
0110        REM WRITE INTO DISK FILE
   :        $OPEN #1
   :        DATA SAVE DC #1, I$, C$
   :        $CLOSE #1
0120        REM ALERT BACKGROUND PARTITION THAT DATA IS AVAILABLE
   :        $ALERT 2
0130        REM CONTINUE GETTING DATA
   :        GOTO 100
```

B. The following is a portion of a background program that communicates with the program in 'A'. Its task is to retrieve the data records from the disk file as soon as they are available. To accomplish this, it uses $BREAK to put itself to sleep until alerted by the program in 'A'. The SELECT ON ALERT statement allows the program to be interrupted by the $ALERT statement in 'A'.

```
0080        REM OPEN DATA FILE
   :        DATA LOAD DC OPEN T #3, "COST"
0090        REM CHECK TO SEE IF OTHER PARTITION HAS PUT DATA INTO FILE
   :        SELECT ON ALERT GOSUB 5000
0100        REM WAIT UNTIL DATA IS AVAILABLE
   :        $BREAK!
   :        GOTO 100
5000        REM PROCESS NEW DATA
   :        $OPEN #3
   :        DATALOAD DC #3, A$, B$
5010        $CLOSE #3
5020        PRINTUSING 5021, A$, B$
5021        % A$ = ####  B$ = ############
5900        RETURN
```