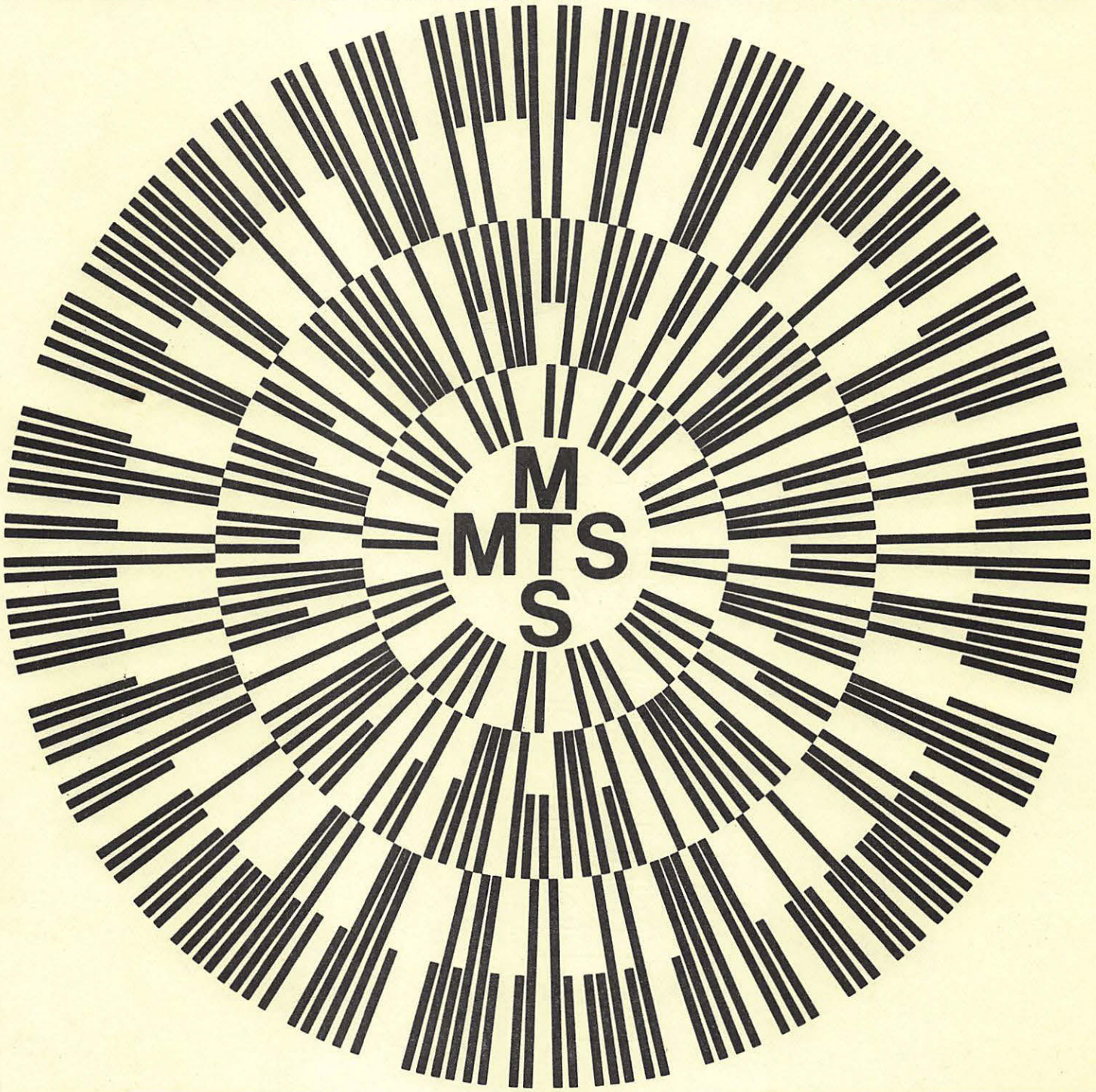
THE UNIVERSITY OF ALBERTA
COMPUTING CENTER
PUBLICATION



UTILITIES

ACKNOWLEDGEMENTS

THIS MANUAL WAS LARGELY COMPILED FROM MATERIAL PREPARED BY THE STAFF OF THE UNIVERSITY OF MICHIGAN COMPUTING CENTER. THEIR DOCUMENTATION WAS INVALUABLE AND WE ARE INDEBTED TO THEM FOR ALLOWING US TO USE IT. IN PARTICULAR, THE FOLLOWING WERE MOST USEFUL:

MTS USERS' MANUAL, SECOND EDITION, VOLUMES I AND II

MTS USERS' MANUAL, THIRD EDITION, VOLUME 2

INTRODUCTION TO MTS AND THE COMPUTING CENTER (FLANIGAN)

COMPUTING CENTER NEWS ITEMS

COMPUTING CENTER MEMOS

THE COMPUTING CENTER WISHES TO PERSONALLY ACKNOWLEDGE THE ASSISTANCE OF MIKE ALEXANDER AND DON BOETTNER WHO HELPED US TO ESTABLISH MTS AT THE UNIVERSITY OF ALBERTA.

ACKNOWLEDGEMENT SHOULD ALSO BE MADE TO THE COMPUTING CENTRE, UNIVERSITY OF BRITISH COLUMBIA, FOR INFORMATION OBTAINED FROM SOME OF THEIR DOCUMENTATION AND TO I.B.M., WHOSE MANUALS PROVIDED CERTAIN SECTIONS FOR OUR MANUALS.

UTILITIES

MAY, 1970

DISCLAIMER

This MTS manual is a combination of earlier manuals, update notices, memos and limited experience with the system itself. Because of this, certain discrepancies are bound to occur and the Computing Center would appreciate being notified of all differences between what this manual says and what the system actually does.

This publication is intended to represent the current state-of-the-system. However, it should not be construed as an obligation to maintain the system as so stated. The MTS system, like most good systems, is continually being improved. As a result, additions, extentions, changes and deletions will occur. Notice of such changes will be made and provision for a manual updating service has been planned.

Errors, comments and suggestions should be sent to:

Information Coordinator
Computing Center
University of Alberta

UTILITIES
MAY, 1970

UTILITIES

TABLE OF CONTENTS

*ASA

Contents: The object module of a program to convert lines with ASA printer control characters in column 1 to lines with machine carriage control in column 1 (which can be directly printed).

Usage: This file is to be referenced by a $RUN command.

Logical I/O units referenced:

SCARDS - the source of the lines with ASA carriage control.
SPRINT - the printer (*SINK* in batch) or place to put the output lines.

Examples:

```
$RUN *ASA SCARDS=-PRINT
     (SPRINT defaults to *SINK*)
$RUN *ASA SCARDS=-PRINT SPRINT=LISTING
```

Note: If the output of *ASA is put in a file, then when the file is finally listed, the machine carriage control modifier should be specified:

```
     $COPY LISTING TO *SINK*@MCC
```

*BCDEBCD

Contents: The object module of the BCD to EBCD conversion program.

Purpose: To convert cards punched on an 026 card punch (BCD) to the equivalent card codes as if punched on an 029 card punch (EBCD).

Usage: The BCD to EBCD conversion program is invoked by an appropriate $RUN command specifying *BCDEBCD as the file where the object cards are found.

Logical I/O units referenced:
SCARDS - BCD lines to be converted.
SPRINT - the listing of converted lines.
SPUNCH - EBCD lines resulting from the conversion.

Examples: $RUN *BCDEBCD SPUNCH=MYFILE
(SCARDS defaults to *SOURCE*, SPRINT to *SINK*)

Description: The following conversion is applied to all input lines:

BCD CARD CODE	CHARACTER	EBCD CARD CODE
0-4-8	(%	12-5-8
12-4-8	) ¤	11-5-8
4-8	' @	5-8
12	+ &	12-6-8
3-8	= #	6-8
all others		unchanged

This conversion maps those characters which had a dual symbolism (scientific and commercial) on the 026 keypunch to the appropriate 029 keypunch scientific code. Note that on an 026 keypunch, there is no way to represent the following characters:

% ¤ @ & #

as this program will convert them to their scientific equivalents.

*DOWNDATE

Contents: The object module of the down-dating program.

Purpose: To compare two versions of a source program and generate an update deck to transform the old program into the new program.

Usage: The program is invoked by an appropriate $RUN command.

Logical I/O Units Referenced:
SPUNCH - the generated update deck.

Example: $RUN *DOWNDATE SPUNCH=VOL7

Description: *DOWNDATE reads as data two decks: the original source deck for a program and a modified version of that same program. *DOWNDATE then punches update cards which would make the original deck identical to the modified deck. The program asks where the old and new source programs are stored. On line one, the program prompts for the FDname of the current source deck; on line two, it prompts for the FDname of the original deck which had been modified to produce the current version. The original source deck must contain *UPDATE-compatible sequencing. The blocking factors of the source programs may differ but must be less than or equal to 80 (i.e., eighty 80-byte records per block). A "%BEFORE FILE-MARK" terminates the generated update deck.

*HEXLIST

Contents: The object module of the hexadecimal card lister.

Purpose: To list cards (usually object cards) in hexadecimal.

Usage: This module is invoked by the appropriate $RUN command.

Logical I/O units referenced:
SCARDS - the place from which the file names are to come.
SPRINT - where the listing is to be put.

Examples:
```
$RUN *HEXLIST
    (SCARDS defaults to *SOURCE*; SPRINT to *SINK*)
```

Description: The output format is:

Line 1: The card number (advanced by 1 for each card read), the MTS file sequence number (line number), the card I.D. (columns 73-80 of the card), column 1 of the card (in hexadecimal), columns 2-4 in EBCDIC.

Line 2: Columns 5-38 in hexadecimal.

Line 3: Columns 39-72 in hexadecimal.

*HEXLIST interacts with the user in the same way as *OBJSCAN (see the description of *OBJSCAN).

Sample output:

```
$RUN *HEXLIST; SCARDS=*SOURCE* SPRINT=*SINK*
#EXECUTION BEGINS
READY!
HXL(83.24,83.24)

CARD NO.  1,FILE SEQ. NO.=83.24     , CARD ID=HXL 0003, 02 , TXT
400000E0404000384040000147F0A0D44184 11F244420A2FED202A39BA309F321A
1D4
A308DC01A1D4A2BED201A397A1D44110A3A058F0A41005EFF3E7A1D4A30CF3E7A1
E23
```

*SORT

Contents: *SORT contains the object module of a sort/merge program.

Usage: *SORT is invoked through the $RUN command.

Logical I/O Units Referenced:
SCARDS - The sort/merge control statement is read via SCARDS if it is not passed as a parameter.
SERCOM - Diagnostics are issued via SERCOM.

Description: The sort/merge program

(1) orders records according to the collating sequence defined in its control statement;

(2) provides blocking and deblocking facilities, processing data sets of types U, F, V, FB, and VB;

(3) allows user subroutines to gain control at several points, permitting the generation, modification, deletion, and comparison of records;

(4) is serially reusable and may be called as a subroutine. See the description of the SORT subroutine in Volume 3.

1. The control statement describes the collating sequence and the structure of the I/O data sets. The control statement is read via SCARDS if it is not passed as a parameter. It may extend over any number of records and may be encoded anywhere between columns one and eighty. The statement may be broken for continuation between parameters and is terminated by the first blank following the MNR keyword.

A prototype of the control statement follows:

```
function=format;order;location;length...
INPUT=name;type;record length;block length...
OUTPUT=name;type;record length;block length...
MNR=maximum number of records
```

A keyword is that part of the statement which precedes the equal sign and a group consists of the four parameters following the equal sign.

The ellipses denote additional groups that may be appended to the first. Subsequent groups should be separated by semico-

lons and the last group for each keyword must be followed by a blank.

A. The function keyword must be replaced by SORT or MERGE. The group(s) following this keyword define the collating sequence. Each group consists of four parameters: format, order, location, and length, and describes one collating field. From one to fifty groups may be used to describe the entire collating sequence. The order in which the groups are coded determines the dominance of the collating fields: the first field described is of primary importance, etc. The collating fields may overlap. The following substitutions are necessary:

1. Replace "function" by SORT or MERGE.

2. Replace "format" by the two character code that describes the structure of the data contained in the field. Permissible codes are summarized below.

FORMAT	CODE	SIGN PRESENT	FIELD LENGTH (BYTES)
character (binary)	CH	no	1 - 256
signed decimal	SD	yes	2 - 16
zoned decimal	ZD	yes	1 - 15
packed decimal (internal)	PD	yes	1 - 15
fixed point (internal)	FI	yes	1 - 256
floating point (internal)	FL	yes	2 - 8

The ascending character collating sequence orders the collating fields by increasing binary value. The characters are ordered

blank ¢.<(+|& $*);>-/,%_?:#@'="
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

Signed decimal fields are composed of a sign followed by one to fifteen digits. Valid signs are "+", "-", and blank (interpreted as a "+"). The sign must occupy the first location of the field. The ascending signed decimal sequence is

-999999999999999 ... +999999999999999

The zoned decimal format and all those below it in the preceding table are internal data structures with which many users have no contact. System/360 Principles of

Operation, IBM publication A22-6821, may be consulted for a description of these formats.

3. Replace "order" by A or D denoting an ascending or descending collating sequence, respectively.

4. Replace "location" by the position in the record of the first byte, or character, of the collating field. The initial byte of a record is at location "1". The value of the location may range between "1" and "4090".

5. Replace "length" by the number of bytes in the collating field. Restrictions on the length of a field are dependent on its format and are noted in the table summarizing the format codes.

B. The INPUT and OUTPUT keyword parameters describe the data sets to be read and written. Each group consists of four parameters, name, type, record length, and block length and describes one data set. Data sets are read in the order in which their descriptions appear (unless the function is MERGE). Data sets are written in the order in which their descriptions appear, each being "filled" before the next is used. The characteristics, i.e., type, record length, and block length, of all output data sets must be alike. The user should be aware of the implications of the implicit "trimming" of blanks in MTS. SORT does not set the not-trim modifier when reading and writing data sets. The following substitutions are necessary:

1. Replace "name" by the file or pseudodevice name of the data set. Modifiers and line number ranges may be appended to the name.

2. Replace "type" by one of the record structure codes appearing in the table below. OS/360 - Supervisor and Data Management Services, IBM publication C28-6646, contains descriptions of the structures.

CODE	RECORD STRUCTURE
U	undefined length
F	fixed length
V	variable length
FB	fixed length blocked
VB	variable length blocked

3. Replace "record length" by the byte length of the longest record occurring in the data set.

4. Replace "block length" by the byte length of the longest block occurring in the data set.

C. The MNR keyword parameter should be replaced by the maximum number of records that may be encountered during the sort or merge.

2. Examples.

A line file named SIN contains 1,000 unblocked card images. Each image bears the name of a city, columns 1 - 60, and the city's population, columns 70 - 80.

A. An alphabetized list of cities is needed.

```
$RUN   *SORT   PAR=SORT=CH;A;1;60   INPUT=SIN;F;80;80
OUTPUT=*SINK*;F;80;80 MNR=1000
```

B. A list of cities ordered by decreasing population and alphabetized within equal populations is needed. Two tapes are mounted with the pseudodevice names *DATA* and *BYPOP*. *DATA*'s records contain additional information to be included in the list. Its records have the format of SIN's but are blocked ten to the block - a total of approximately 2,000 records. *BYPOP* will contain the sorted list. Its records should be type VB, a maximum of 10,000 bytes per block.

```
$RUN *SORT
SORT=CH;D;70;10;CH;A;1;60
INPUT=SIN;F;80;80;*DATA*;FB;80;800
OUTPUT=*BYPOP*;VB;84;10000 MNR=4000
```

3. Optional exits.

User subroutines may gain control from the sort/merge program at any of four points or exits. An exit is enabled, i.e. a subroutine is called, if the name of the exit is encountered as an external symbol during the loading of the sort/merge program.

All subroutine calls are made via a standard OS linkage, type (I) S. Upon entry, general register one will point to a parameter list: the address of the first byte of the record or region; the address of the halfword length of that record or region. Note that the region and length when passed to SORTE2, SORTE3, and SORTE4 are halfword aligned. (See below for details).

The subroutines should return control to the sort/merge program via the standard OS (I) S return. The exits are as follows:

SORTE1 If this exit is enabled, it is taken to acquire a record for processing. The call to the user's subroutine replaces the call that would normally be made to READ. A block may be moved into the region addressed by the first word of the parameter list. The length of this block should be placed in the halfword addressed by the second word of the parameter list.

If this exit is enabled, a dummy data set must be defined following the INPUT keyword. The data set's parameters should describe the block to be passed to the sort/merge program by the subroutine. The deblocking facilities may be used to decompose blocks passed by the subroutine.

An end-of-file can be passed to the sort/merge program by a non-zero return code in general register 15.

SORTE2 If this exit is enabled, it is taken immediately after a record has been acquired for processing -- following deblocking, if any. The user's subroutine may delete or modify the record before it

enters the sort or merge. The first word of the parameter list will point to the record; the second, to its halfword length plus four. If a record's length is altered by the subroutine, the new length, plus four, must replace that passed to the subroutine. If a record's length is made zero, the record is discarded from the sort or merge. A non-zero return code will force the end-of-file condition on the current input data set.

SORTE3 If this exit is enabled, it is taken to dispose of a sorted or merged block. The call to the user's subroutine replaces the call that would normally be made to WRITE. The first parameter points to the block and the second to its halfword length.

If this exit is enabled, a dummy data set must be defined following the OUTPUT keyword. The data set's parameters should describe the block or record to be passed to the subroutine. The blocking facilities may be used. A non-zero return code causes the next output data set to be opened and filled.

SORTE4 If this exit is enabled, it is taken to compare two records. The call to the user's subroutine replaces the comparison normally made by *SORT. The parameter list differs from that of the preceeding exits. The two words of the list point to two regions containing the records for comparison. The first halfword of each region has a value of four plus the length of the text in that region. The text begins at an address four bytes higher than that of the region.

The subroutine's return code indicates the result of the comparison. A zero return code indicates that the record appearing in the first region, i.e., the region addressed by the first parameter, should precede the record appearing in the second region. A non-zero return code indicates that the second region's record should precede the first region's record.

If the exit is enabled, the parameters describing the collating must not appear. Only "SORT= " or "MERGE= " should be coded.

The following Fortran subroutine exemplifies the use of the exit SORTE2. It deletes records having blanks in columns 3 and 4 and terminates the reading of a data set if columns 3 and 4 contain '09'.

```
      SUBROUTINE SORTE2(REGION,LENGTH,*)
      INTEGER*2 REGION(40),BLANK/'  '/,NINE/'09'/,LENGTH
      IF (REGION(2).EQ.BLANK), GO TO 100
      IF (REGION(2).EQ.NINE), GO TO 200
      RETURN
100   PRINT 900, (REGION(I), I=1,40)
900   FORMAT(' RECORD DISCARDED:',40A2)
      LENGTH = 0
      RETURN
200   RETURN 1
      END
```

*STATUS

Contents: The object module of the user accounting status program.

Purpose: To print information regarding the user's charge, terminal time, current and cumulative disk space, plotting time, CPU and wait memory usage, CPU, time, I/O batch terminal sessions, and expiration time.

Usage: The accounting status program is invoked by an appropriate $RUN command specifying *STATUS as the file where the object cards are found.

Logical I/O units referenced:
SPRINT = listing of the accounting information

Examples:
```
$RUN *STATUS
$RUN *STATUS SPRINT=FILE
$RUN *STATUS PAR=FULL
$RUN *STATUS PAR=$
```

Description: *STATUS lists on SPRINT the following information for the user. The maximum, used, and remaining figures are given for charge in dollars, terminal time in hours, current file space in pages, and plotting time in hours. If the program is run from a terminal, this is normally all that is printed. However, printing of all information can be forced on a terminal by putting PAR=FULL on the $RUN command. If this is done or the program is run in batch, the cumulative used figures are given for file storage in page-days, CPU and wait memory in page-hours, CPU time in hours, lines and pages printed, cards read and punched, the number of batch and terminal sessions, and the expiration time. If all information about an item is zero, no information is normally printed. If PAR=$ is given on the RUN command, then only the remaining funds as of the last signoff will be printed, either on a terminal or in batch.

It must be emphasized that the information printed is only approximate. A user's true position is indicated only by his monthly bill.

*SYMBOLS

Contents: Object module to print a listing of all external symbols within the MTS system.

Usage: This file should be referenced by a $RUN with *SYMBOLS as the object file, SPRINT as output, and SCARDS as the location of a control record.

Logical I/O units referenced:

SCARDS - the specified file or device from which a control record is read.
NOTE: *SYMBOLS(0) is a valid control record.

SPRINT - the specified file or device on which the output listing is produced.

Example:
```
$RUN *SYMBOLS SCARDS=*SYMBOLS(0)
     (SPRINT default to *SINK*)
```

Description: This program produces the listing of MTS external symbols which is reproduced in Volume 3 of this MTS manual. The listing is produced by searching the external symbol dictionaries of the system and the library. Hence it is current as of the instant it is run.

The program has options to generate printed output or a deck in TEXT360 format. It also will suppress privileged system symbols if desired. These options are set by a control record which the program expects on the SCARDS file or device. The format of this control record is not provided in this writeup.

There is a control record in the file *SYMBOLS at line number 0. This record will cause the program to produce printed output with privileged symbols suppressed. Hence a user may obtain a listing by the command:

```
$RUN *SYMBOLS SCARDS=*SYMBOLS(0)
```

*TASKS

Contents: The object module of the TASKS job-status program.

Purpose: To selectively display the status of UMMPS jobs.

Usage: The program is invoked by a $RUN command specifying *TASKS as tne file containing the object module.

Logical I/O Units Referenced:
GUSER - command input.
SERCOM - command prompting and error comments.
SPRINT - job status output.

Parameters: If a parameter is given, it will be treated as one line of GUSER input (see below) and then *TASKS will stop.

Examples:
```
$RUN *TASKS
$RUN *TASKS PAR=MTS
$RUN *TASKS PAR=USER SSSS
$RUN *TASKS PAR=D DC09
```

Description: The TASKS job status program prints (on SPRINT) one line of output for each UMMPS job which fits the descriptor (entered from GUSER). Each output line contains, from left to right: the UMMPS job number, the number of virtual memory pages that the job is using if it is a relocatable job (such as MTS), the job name, the low-core address of the job's UMMPS job table, the job table parameters (if any), and the I/O devices attached to the job (if any). For MTS, the job table parameters are the four character user ID (all blanks if the job currently has no one signed on), the four character user charge number ("IDLE" if no user is attached and "BUSY" if a user is present, but a valid $SIGNON line has not yet been processed), and the S-8 (receipt) number if it is a batch job. An example is shown below.

```
$RUN *TASKS
EXECUTION BEGINS
READY:  0402
 0402    8    MTS: 02DDA8   W073   W000; LA36
READY:  MTS
 0135    8    MTS: 02D358   SS33   S988; DC05
 0402    8    MTS: 02DDA8   W073   W000; LA36
 0026    8    MTS: 02E900   K01M   K01M; LA41
 0027    4    MTS: 02EA08   N756   N756; LA42
 0029    3    MTS: 02EC18   K06D   K06D; LA44
 0333   17    MTS: 031030   IA92   IA92; LA40
READY:  DEVICE LA36
```

```
 0402   8   MTS: 02DDA8  W073  W000; LA36
READY:  T PTR
 0009  HASPLING: 02D988  PTR3; PTR3
 0010  HASPLING: 02DA90  PTR1; PTR1
READY:  $ENDFILE
EXECUTION TERMINATED
```

Descriptors are entered from GUSER allowing the user to selectively display the status of current UMMPS jobs. The allowable descriptors are:

```
an UMMPS job number (1-8 digits)
B       (for Batch)
F       (for Full)
M       (for MTS)
N       (for Non-MTS)
D XXXX  (for Device)
T XXXX  (for device Type)
U XXXX  (for User ID)
```

Leading and trailing blanks are ignored on all input lines. If a job number is entered, the first non-numeric character terminates the number and the rest of the line will be ignored. If the first non-blank character in the input line is B, F, M, or N, then the remainder of the input line is ignored. Thus it is permissible to enter MTS instead of M, for example. If the first non-blank character is D, T, or U, then the argument for the descriptor begins with the first succeeding non-blank character unless there are more than four characters given for the argument, in which case only the last four are considered.

*TIME

Contents: The object module of the clock program.

Purpose: To print out the current date and the current time.

Logical I/O Units Referenced:
SERCOM - a single output line consisting of the clock time.

Example:

```
#$RUN *TIME
#EXECUTION BEGINS
CLOCK 04:52:03 DATE 12-29-71
#EXECUTION TERMINATED
```

*UPDATE

Purpose: This program will copy tapes (or files) containing card images, making insertions and deletions, as well as blocking and unblocking.

Logical I/O Units Referenced:

SPRINT - printed output.
SPUNCH - output from %PUNCH.
SERCOM - error messages.

Commands and insertions are expected to come from the source stream (*SOURCE*). If another source of commands is wished, its FDname should be specified following the "PAR=" on the $RUN command. Commands and insertions must be less than or equal to 80 bytes in length.

Examples:

```
$RUN  *UPDATE
$RUN  *UPDATE PAR=PIL.UPDATE
```

Usage: The update input tape must consist of 80 column card images which may be blocked to any factor desired. The blocking factor, if greater than 1, must be stated on the %INPUT command. The update output tape will consist of 80 column card images blocked as specified (except for the last record, and other records which may be truncated by a %CLOSE command). [Space for the specified input and output buffering is obtained dynamically when %INPUT and %OUTPUT command are encountered; released when %CLOSE is encountered.]

All commands take the following form: column 1 must contain a percent-sign ("%") which must be immediately followed by the command (only the first three letters need be given, and they must be upper-case. Most devices are in upper case mode unless commanded otherwise). Parameters for the command are separated from the command and from each other by one or more blanks (or commas, which are treated identically with blanks). Lines which are not recognized as commands are treated as insertion lines and are copied immediately to the update output tape.

There are four different types of parameters used in the commands: numeric, filemark, character, and keyword. Numeric parameters are used for tape operation counts, deletion counts, and so forth. They consist of one to twenty digits which represent an unsigned decimal integer. A filemark parameter is used to refer to a filemark and consists of the characters FILEMARK or FILEMK. Character parameters are used for FDnames, pseudo-device names, I.D.'s, etc. There are two forms of character parameters. The first form consists of

one to eighty characters with the restrictions that the first character cannot be a digit or an apostrophe (') and neither blanks nor commas (,) can be a part of the character parameter. The second form of a character parameter consists of from one to eighty characters enclosed in apostrophes, with an apostrophe within the character parameter represented by two adjacent apostrophes. The second form does not restrict the use of a digit or apostrophe as the first character nor the use of blanks and commas within the parameter. Note that the outer apostrophes act only as delimiters and are not considered a part of the character parameter. Keyword parameters are simply keywords which are specified for a specific command, such as ON or OFF.

Examples:

Numeric Parameters	1 2 15 123
Filemark Parameters	FILEMARK FILEMK
Character Parameters	PIL6215 *T* SEQ.0001
	'12340001' 'PIL 00001'
	'''TS"0001' '*T*(1,100)'
Keyword Parameters	ON OFF *

Tape positioning and reading across a file mark while the tape is still open is considered an error. Therefore %CLOSE commands should precede positioning and the update should finish copying of a file with %BEFORE FILEMARK, not %AFTER FILEMARK.

Tape Attachment and Manipulation Commands

%INPUT INTAPE [N]

INTAPE is the pseudo-device name (or FDname) of the file or device to be established as the update input tape. N is an integer specifying the blocking factor of that tape. If omitted, a blocking factor of 1 card/record (i.e., unblocked) is assumed. This command causes the tape to be opened.

Example: %INPUT *IN* 50

%OUTPUT OUTTAPE [N]

OUTTAPE is the pseudo-device name (or FDname) of the file or device to be established as the update output tape. N is an integer specifying the blocking factor desired on that tape. If omitted a value of 1 (unblocked) is assumed.

Example: %OUTPUT *OUT* 20
%OUTPUT FILE1 1

%REWIND T

Tape T is rewound. T must not currently be open as

input or output. The rewind operation is performed by the calling the subroutine REWIND#. All files or devices which this subroutine can rewind may be specified. See the writeup on REWIND# in Volume 3 of the MTS Manual.

Example: %REWIND *OUT*

%RUN T
%UNLOAD T
Tape T is rewound and unloaded. T must not currently be open as input or output.

Example: %RUN *OUT*

%FSF T [N]
Tape T is spaced forward N files. If N is omitted a value of 1 is assumed. T must not currently be open as input or output.

Example: %FSF *IN* 3

%BSF T [N]
Tape T is spaced backwards N files. If N is omitted a value of 1 is assumed. T must not currently be open as input or output.

Example: %BSF *IN*

%WTM T [N]
%WEF T [N]
%EOF T [N]
Tape T has N tape marks (end-of-file marks) written on it. If N is missing a value of 1 is assumed. T must not currently be open as input or output.

Example: %WTM *OUT*

%FSR T [N]
Tape T is spaced forwards N records. If N is omitted, a value of 1 is assumed. T must not currently be open as input or output.

%BSR T [N]
Tape T is spaced backwards N records. If N is omitted, a value of 1 is assumed. T must not currently be open as input or output.

%CLOSE [T]
Tape T is closed. If the update output tape, the last buffer (possibly truncated) is written out. If T is omitted, the update output tape is assumed.

Update Feature Control Commands

%NEWID H

Cards written onto the update output tape following this command will have new ID's (columns 73-80). The first card written will have the ID specified in this command. Succeeding cards will have ID incremented in steps of one. The ID given in this command should consist of 8 characters. Only the numeric portion of the ID is incremented.

Example: %NEWID PIL00001

%OLDID

Suspends the re-IDing of the cards as described under %NEWID.

%LIST ON

Starts listing of deleted and inserted cards. (Initially on)

OFF

Stops listing of deleted and inserted cards [This output goes onto *SINK*]

%PUNCH ON

Starts putting all card images sent to the update output tape on SPUNCH (presumably for punching).

OFF

Turns off the "punching" described above. (Initially off)

Card Location, Copying and Deletion Commands

In the execution of these commands, the 360 collating sequence is used for comparisons.

%AFTER ID

Copies all cards having ID's less than or equal to ID from the update input tape to the update output tape

N

Copies the next N cards from the update input tape to the update output tape.

Examples: %AFTER PIL03789
%AFTER '04780000'
%AFTER 2

%BEFORE ID

Copies all cards having ID's less than ID from the update input tape to the update output tape

FILEMARK

FILEMK
Copies the rest of the file. It leaves the tape positioned after the file mark.

Examples: %BEFORE PIL07892
%BEFORE FILEMARK

%DELETE ID
Copies all cards having ID's less than ID from update input to update output, then deletes the card (or cards) having ID of ID (if any).
N
Deletes the next N cards on the update input tape
ID1 ID2
Copies all cards having ID's less than ID1 from update input to update output, then deletes all cards having ID's ID1 through ID2 inclusive from the input.
ID1 N
Copies all cards having ID's less than ID1 from update input to update output, then deletes the next N cards on the input tape.
* ID2
Deletes all cards on the update input tape from the current position up through ID2.

Examples %DELETE PIL00016
%DELETE 2
%DELETE PIL00378,PIL00379
%DELETE PIL00378,2
%DELETE * PIL00736

%FIND ID
The update input tape is searched for a card with ID equal to ID. Order of the ID's is ignored. Card passed over are not copied to the update output tape.

Example: %FIND QQSV0395

%UNTIL ID
The update input tape is searched for a card with ID equal to ID. Order of the ID's is ignored. Cards passed over are copied to the update output tape.

Return Command

%END
This command or an end-of-file encountered in the command stream causes execution of the update program to terminate. All buffers are closed.

Sample Command Stream

```
%INPUT    *IN*  50
%OUTPUT   *OUT*   1
%DELETE   PIL00016
%AFTER    PIL00079
            GETSPACE  8193,T=3
            LR     15,1
            L      14,=F'8192'
%DELETE   PIL00830,2
%DELETE   PIL07044,PIL07049
            PUTLINE
            GETLINE
%AFTER    PIL09711
SAVR#       DS   6F
SAVREG#     DS  18F
PARREG#     DS   6F
%BEFORE   FILEMARK
%CLOSE
%CLOSE    *IN*
%WTM      *OUT*
%REW      *OUT*
%RUN      *IN*
%END
```

*USERS

Purpose: To allow the user to find out how many people are using MTS.

Usage: The program is invoked by the command $RUN *USERS

Description: The number of active jobs in each of several categories and the number of virtual and real pages in use at the current time is printed on the user's terminal or batch output (through SPRINT).

Logical I/O Units Referenced:
SPRINT - output of the system usage information.