Computing Centre
The University of British Columbia

U67-0
December, 1968
Revised: November, 1970

SO YOU WANT TO USE THE IBM 360/67

by J. M. Kennedy

Revised Edition by Andrea Berger

U.B.C. was one of the last large Canadian universities to make the change from an IBM 7000-series machine to a 360 System. However the IBM 360, Model 67 Duplex now operating in the Computing Centre is one of the most complex systems anywhere in Canada.

This informal report is designed as a guide for new users of the U.B.C. system. The document, itself, is divided into seven main sections. They are:

1. What is the 360/67?
2. Batch runs without botches.
3. Notes on the Command Language.
4. Fiddling with files.
5. Types of files.
6. Tips on terminals.
7. High-level hints.

---

*Programmers' jargon that is used in this report is underlined and defined the first time it occurs.

1. What is the 360/67?

The 360/67 supports up to ten batch streams--many of them from card readers and printers in other university buildings--and about 50 low-speed "conversational terminals" that communicate with the machine by means of some form of typewriter keyboard. Thus the central processing unit of the 360/67 must keep something like five dozen programs in some part of its working memory at once, and it must devote some attention to each one of them every second or two. This is handled by the MTS Operating System (Michigan Terminal System, written at the University of Michigan Computing Centre); it is supposed to be "transparent to the user"--that just means that we see all these tasks running together but you don't.

Other distinctive features of the 360/67 include the large data cell and disk storage areas for users' programs and data. These can hold millions of lines of information (a "line" is the information from one punched card, or one line for a printer, or the equivalent). The data cell and disk packs are organized into a flexible and powerful filing system, and every user is entitled to some of it for his own needs. Other areas of the disk contain public files (like Fortran compilers or the subroutine library) that may be used by anyone.

Because this system is both powerful and complex, the command language of MTS is a rich one. Fortunately, MTS usually has a default option where it offers you a choice: that is, if you don't state your choice explicitly, a plausible assumption is made and the job carries on.

2. Batch runs without botches.

Here and in later sections, the emphasis will be on illustrative examples accompanied by "rules". These aren't the whole MTS story by any means, but they should put you in the right frame of mind to read the more detailed manuals.

The first real rule is:

> **Rule 1:** Every batch job must be preceded by a Request for Service Card, available at all input stations.

This card won't be shown in the examples.

Example 1 shows a trivial job that illustrates three rules about the MTS command language:

```
$SIGNON MYCO  PW=SESAME
$SIGNOFF

↑

Column 1
```

Example 1.

2. Batch runs without botches.

Rule 2. All command cards have $ in Column 1. The remainder of the card is in free format with no particular columns assigned for the information.

Rule 3. The first card must be $SIGNON accompanied by your user identification code. The first part of the identification is your 4-character I.D. code -- obtained when you apply for authorization to use the computer -- and the next is PW = your password. More on passwords in the next section.

Rule 4. The last card must be $SIGNOFF to show that you are finished with the system.

The purpose of the password is to keep others from using your I.D. code. It is prudent to conceal it from your enemies (and your friends). One step towards this goal is to leave the password off the $SIGNON card, and to supply it on the next card. This second card isn't printed with your output, and we recommend that you disable the printing on the keypunch when you are preparing it. This is shown in Example 2, which also illustrates:

```
$SIGNON MYCO T=60 P=10 C=50
SESAME
   .
   .
   .
 (Job for running)
   .
   .
   .
$SIGNOFF
```

Example 2.

Rule 5. Limits on time, printed output, and punched cards are set by T=time in seconds, P=no. of pages, C=no. of cards. If any or all are missing, MTS sets the values to 30 seconds, 20 pages, no cards.

Example 3 shows a short, self-contained WATFIV job. The following rules pertain to WATFIV usage:

Rule 6. WATFIV is invoked by $RUN *WATFIV which tells MTS to execute the program in the library file *WATFIV (note that its name is not FORTRAN).

Rule 7. The control cards $COMPILE, $DATA, $STOP are commands to WATFIV, not to MTS. $COMPILE says "here comes a program for compilation." $DATA denotes the end of the source code, and calls for execution of the compiled program; it must be included whether or not you have any input data. $STOP tells WATFIV to turn control back to MTS.

```
$SIGNON MYCO
SESAME
$RUN *WATFIV
$COMPILE
      READ 1,X
1     FORMAT(2F10.5)
      U=SQRT(X)
      PRINT 1,X,U
      STOP
      END
$DATA
  21.67035
$STOP
$SIGNOFF
```

Example 3.

2. Batch runs with botches (cont'd.)

Rule 8: A succession of WATFIV tasks can be run by alternating $COMPILE and $DATA as shown in Example 4. This is more efficient than signing off and then on again; among other things, it avoids the need to reload WATFIV from the disk file anew for each task.

```
$SIGNON MYCO
SESAME
$RUN *WATFIV
$COMPILE
   (Source cards for Job #1)
$DATA
   (Data cards for Job #1)
$COMPILE
   (Source cards for Job #2)
$DATA
  etc.
$STOP
$SIGNOFF
```

Example 4.

The program shown in Example 3 can be run under IBM Fortran-G instead of WATFIV. The card deck for this is shown below as Example 5.

```
$SIGNON MYCO
SESAME
$RUN *FORTRAN
   (Fortran source cards)
$ENDFILE
$RUN -LOAD#
  21.67035
$ENDFILE
$SIGNOFF
```

Example 5.

The different sequence of command cards exhibits the different behaviour of Fortran and WATFIV. Whereas WATFIV compiles in one pass for immediate execution, Fortran insists on storing the translated program somewhere before running it.

Rule 9: The card $ENDFILE is used to signify the end of the source code, and again to signify the end of the data. You will find that MTS uses $ENDFILE in other similar situations.

Rule 10: Execution of your compiled program is invoked by the card $RUN -LOAD#. This instructs MTS to run the program stored in a file called -LOAD#. Luckily for you, this is where Fortran has just put your object code, since you didn't tell it anywhere else to put it.

2. Batch runs without botches (cont'd.)

Additional information on batch processing and WATFIV will be found in the following Computing Centre documents:

UBC BATCH: Initiating Batch Runs from a Terminal

UBC WATFIV: WATFIV FORTRAN Compiler

U67-3-1: The Batch User's Guide

If your program has errors, you will receive error messages from the system. WATFIV defines the error explicitly while error messages from IBM Fortran-G include an error code and an indication with a dollar sign ($) of the location of the offending symbol. The codes are listed in:

U67-5: IBM FORTRAN IV(G) Error Messages and Completion Codes.

3. Notes on the Command Language

So far, we have encountered several words of the MTS Command Language. There are plenty more, some of which will be met in the section on Files that follows. In general, the Command Statements have a fairly free format. A safe rule is:

> Rule 11: Don't leave a gap between $ and the command word. Leave a single space between "words" whenever no separator appears explicitly, but don't leave other gaps unnecessarily.

The experts know abbreviations for the command words that will do just as well as the full word. You can pick them up some time too.

Several different tasks can be done between a single $SIGNON - $SIGNOFF pair. Example 4 showed a sequence of WATFIV tasks; more generally, any sequence of meaningful cards from the Command Language can be put end to end.

From time to time, you may want to change your password. It is set initially to be equal to your I.D. code when you are given authorization to use the computer. Example 6 shows the use of the $SET command to alter a password. A $SET card may be included as part of any job. The password must be 6 or fewer characters in length.

```
$SIGNON MYCO PW=SESAME
$SET  PW=SEZME
$SIGNOFF
```

Example 6.

4. Fiddling with Files

Much of the power of the 360/67 comes from the fact that it is easy (well, fairly easy) to keep programs and data in retrievable form in the disk memory of the computer.

> Definition: A line file is a set of numbered lines, normally stored in disk storage.

MTS is a file-oriented operating system. Its general philosophy can be summarized in:

> Rule 12: All information handled by MTS can be (and frequently is) in one or more files.

"All information" in Rule 12 includes your program in either source language or machine language, your data, the FORTRAN and WATFIV compilers themselves, the Computing Centre telephone list, etc., etc. From the point of view of MTS, a card reader or a printer is a pseudo-file whose essential property is the ability to deliver or receive a sequence of lines of information.

The discussion that follows ignores a lot of "high-level" facts about files that you should learn before too long.

Here are some operations you can do with files:

```
$CREATE them or $DESTROY them,
$LIST them or $COPY them,
$GET them or $RUN them,
$EMPTY them.
```

If you intend to put lines into a file, the definition says they must be numbered. One way to do this is to attach a number to each line yourself. However, files are usually filled initially with a set of sequentially numbered lines, and we might was welll let MTS number them for us.

Before putting our programs into files, let's manipulate some files of data. Example 7 shows a simple job that creates a file called FILEFACTS and then lists it.

After $SIGNOFF, the file remains available for subsequent reference by name.

```
$SIGNON MYCO
SEZME
$CREATE FILEFACTS
$NUMBER
   THIS SAMPLE FILE CONTAINS SOME
 INFORMATION ON FILES IN MTS.
   1. A FILE IS A SET OF NUMBERED
 LINES STORED UNDER SOME UNIQUE
 NAME
$UNNUMBER
$LIST FILEFACTS
$SIGNOFF
```

Example 7.

> Rule 13: A file of modest size can be created by $CREATE followed by the name you wish to attach to it. It is retained for later use (by you and nobody else) until you $DESTROY it.

Rule **14:** The command $NUMBER instructs MTS to number the lines that follow as they are put in the file. The numbering is sequential from 1 unless indication to the contrary is given.

$UNNUMBER cancels the instruction to $NUMBER.

Rule **15:** $LIST, followed by a file name, calls for a listing of the contents of the file, starting from Line #1, accompanied by the line numbers.

The file FILEFACTS in Example 7 will have five lines, numbered 1 to 5.

Suppose that you find after creating a file that you have left out a line in the middle and intended to say more in line 5. You can edit your file on a subsequent run as shown in Example 8.

```
$SIGNON MYCO
$EZME
$GET FILEFACTS
2.5, YOU SHOULD LEARN THEM.
5,NAME ASSIGNED BY THE USER.
$SIGNOFF
```

Example 8.

Rule **16:** $GET is used to notify MTS of the name of the "active file" -- the file that is currently in use for receiving new lines. The file named has usually been $CREATED previously.

Rule **17:** If $NUMBER isn't used, each line must be numbered explicitly; the number is separated from the text by a blank or a comma.

Rule **18:** A line can be inserted between existing lines by giving it some intermediate number. This number can be fractional, or even negative. (But note that according to Rule 15, $LIST still lists starting from Line 1 unless you take special precautions even if your file has a line numbered .5).

Rule **19:** A line with a number already in use in the file calls for a replacement of the old line with the new.

The effect of Example 7 followed by Example 8 is to leave FILEFACTS with six lines 1, 2, 2.5, 3, 4, 5(revised).

After a certain amount of editing and insertion of new lines, a file may get rather messy -- i.e. full of interpolated lines. For this or other reasons, you may want to copy the information into a new file. Example 9 shows how to do this.

```
$SIGNON MYCO
$EZME
$CREATE NEWFILE
$COPY FILEFACTS TO NEWFILE
$SIGNOFF
```

Example 9.

Rule **20:** The $COPY command copies the contents of one file into another. In this process the lines are numbered sequentially from 1 as they enter the new file, whatever their numbers may be in the old file. The old file is retained unaltered.

Rule **21:** Information that is no longer wanted can be deleted either by a command of the form

$EMPTY FILEFACTS

which removes the contents of the file but retains its identity, or by a command of the form

$DESTROY FILEFACTS

which deletes the file and returns the space on the disk to the pool of available storage.

4. Fiddling with Files (cont'd.)

As a matter of prudence, Example 9 doesn't $DESTROY the old file. If you mispunch either the $CREATE or $COPY card, this part of the job would be bypassed, and it would be a pity to lose the information in FILEFACTS.

Of course, most files will consist of programs or numerical data. Example 10 shows the compilation and execution of our small WATFIV program, but set up in such a way that the source program is left in a file for later use.

A problem occurs if you want to put a line like $COMPILE that begins with $ into a file. To prevent it from looking like an MTS command for immediate execution, the rule is:

> Rule 22: Lines beginning with $ are not entered into files, but are treated as MTS commands. Lines beginning with $$ are entered into a file with the first $ removed.

```
$SIGNON MYCO
SEZME
$CREATE PROG
$NUMBER
$$COMPILE
      READ(5,1) X
1     FORMAT(2F10.5)
      U=SQRT(X)
      WRITE(6,1) X,U
      STOP
      END
$$DATA
$$STOP
$UNNUMBER
$RUN *WATFIV SCARDS=PROG 5=*SOURCE*
  21.67035
$SIGNOFF
```

Example 10.

Up to $RUN, everything is covered by earlier rules. But now, when *WATFIV takes over, the source program is the file PROG, not the deck of cards of Example 3. The curious command SCARDS=PROG on the $RUN card tells the compiler about this.

> Rule 23: If the source cards for a compilation aren't coming from the same device as the $RUN card, the $RUN card should contain SCARDS=(name of file). The word SCARDS stands for "symbolic cards" or something of that sort. Similarly, the listing of the source code can be directed to another file by SPRINT=(name of file), and any punched card output to still another by SPUNCH=(name of file).

Note that in Example 10 the data comes from real cards even if the program does not. As a result, the statement 5=*SOURCE* is required. It says that input device 5 is the "pseudo-device" *SOURCE*. This simply means that the data cards will come from the same input device as the MTS commands. For this reason units 5 and 6 are often explicitly used in input-output statements rather than the simpler forms allowed by FORTRAN and WATFIV.

4. Fiddling with Files (cont'd.)

Once Example 10 has been run, you can run the same job with new data without reloading the program cards. This is shown in Example 11 :

```
$SIGNON MYCO
SEZME
$RUN *WATFIV SCARDS=PROG 5=*SOURCE*
   3.1416
$SIGNOFF
```

Example 11.

It is still necessary to call **WATFIV** in Example 11, as the **WATFIV** compiler doesn't provide facilities for retaining the object program.

The corresponding procedure for Fortran is shown in Examples 12 and 13.

```
$SIGNON MYCO
SEZME
$CREATE FPROG
$NUMBER
      READ(5,1)X
1     FORMAT(2F10.5)
      U=SQRT(X)
      WRITE(6,1)X,U
      STOP
      END
$UNNUMBER
$CREATE OBFILE
$RUN *FORTRAN SCARDS=FPROG SPUNCH=OBFILE
$RUN OBFILE 5=*SOURCE*
  21.67035
$ENDFILE
$SIGNOFF
```

Example 12.

```
$SIGNON MYCO
SEZME
$RUN OBFILE 5=*SOURCE*
   3.1416
$ENDFILE
$SIGNOFF
```

Example 13.

In Example 12, the machine-language object program is retained in a file called OBFILE by telling MTS that SPUNCH is OBFILE. Otherwise it would have been held temporarily in -LOAD# as in Example 5, but would not have been available for the later run of Example 13.

It is not necessary to follow the program text with $ENDFILE in Example 12 as it was in Example 5. An "end of file" signal is supplied to *FORTRAN automatically by MTS when the reading of FPROG is complete.

5. Types of Files.

The examples so far have dealt mainly with private files -- i.e. files created by the user for his personal use, and retained from job to job. There are two other types, library files and scratch files.

A library file has a name beginning with an asterisk. Anyone may use one, but only the elect may create one. Examples are *WATFIV or *FORTRAN.

A scratch file has a name beginning with a negative sign. You may create and use scratch files just as you do ordinary private files. The only difference is that at $SIGNOFF all scratch files are destroyed. As we limit your eligibility for private files, you should use scratch files whenever possible.

If you want to let other people read your files, you may set a "permit code" that allows it. If, for example, you feel that NEWFILE created in Example 9, ought to be made available, Example 14 shows how this is done. The code "RO" stands for "read only"; its effect can be reversed by a card with NONE instead of RO.

```
$SIGNON MYCO
SEZME
$RUN *PERMIT
NEWFILE RO
$ENDFILE
$SIGNOFF
```

Example 14.

Now another user can make a copy of your file by doing a run like that in Example 15. The manner of referring to someone else's files is given by

```
$SIGNON ABCD
ALIBAB
$CREATE XEROX
$COPY MYCO:NEWFILE TO XEROX
$SIGNOFF
```

Example 15.

Rule **24:** Reference to files other than your own requires the owner's I.D. code and a colon (:) preceding the file name. After all, dozens of users may have files called NEWFILE.

To conclude this section, it is worth remarking that within reason, input-output devices and files are interchangeable under MTS. You may either think of a file as a substitute for an input-output device, or the reverse.

6. Tips on Terminals

There is a rumour that conversational (typewriter) terminals are supposed to serve as giant desk calculators. It is possible to do small tasks at a terminal -- in fact there is a special language called PIL for this purpose -- but the main advantages of terminals lie in other directions. The importance of a terminal is that it places the full facilities of the 360/67 and MTS at your disposal for immediate access to your files and immediate execution of your programs (subject to the demands of other users, of course).

Typewriter terminals come in several varieties. Like automobiles, each model has its own collection of buttons positioned slightly differently from similar buttons on other models. This is mildly annoying when you first use them, but can be overcome by determination and exercise of the intellect -- after all, you mastered the keypunch once. Our system will be supporting IBM 2741 terminals, three models of Teletypewriters, and a few IBM 2260 display terminals.

Each terminal will be equipped with some instructions on how to perform basic operations like calling up the Computer, deleting an erroneous character, etc. A full description is contained in U67-8: The Conversational Terminal User's Guide. In this section we shall consider one type only: the direct-wired IBM 2741, of the sort provided for "public" use at the Computing Centre. The drill for the Teletypewriter is similar but not identical.

Even if you are a good typist, it is possible to type incorrect lines in the excitement of tickling the computer. You may misspell file names, or issue MTS commands that make no sense (such as trying to $RUN a data file like FILEFACTS). Fortunately MTS is reasonably patient. It will send back helpful messages when it can't understand you. Thus after a bit of orientation, a quarter of an hour spent at a terminal may clear up a lot of obscure points about the system.

Arousing the Computer from an IBM 2741.

1. Turn on the power if necessary.
2. Press the ATTN (attention) button.

The typewriter should wake up and type a line or two of text, ending by typing the symbol "#" at the start of a new line. It is now ready for use.

> Rule 25: Following the # you may type (almost) anything that you would have put on a card for a batch run. MTS Commands will be executed immediately.

Some simple operations on the IBM 2741.

> Rule 26: When you have typed a line to your satisfaction, press the RETURN key to deliver the line to the computer.

The first obvious thing to try is the $SIGNON - $SIGNOFF combination. Example 16 shows a re-run of Example 1 on a terminal. If you are reasonably nimble, the "approximate cost" to U.B.C. will be about $.03, charged against your allocation.

```
#$signon myco pw=sezme
  (CONFIRMATORY MESSAGE
   FROM THE COMPUTER)
#$signoff
  (STATISTICS ON THE RUN
   FROM THE COMPUTER)
```

Example 16

By this time you may be ready for

> Rule **27:** You can correct your typing (prior to hitting RETURN) by backspacing and typing over the old material. Note that each backspace deletes one character; you must retype everything that follows the correction. Alternatively, you can delete the whole line by typing an underscore (_) and then start again.

Manipulation of Files.

One important use of a terminal is to create or modify files. Examples 7 and 8 can be run substantially unchanged. You will find that you receive information from MTS as the task proceeds. For example, a confirmation is typed immediately when a file is $CREATED. Also, when you use $NUMBER, MTS will type the line number on each line of the page before releasing the keyboard for you to type the line.

Don't worry about the IBM 2741 typing your material in lower case letters, even if you have been conditioned to expect everything in upper case. Actually, unless you take special measures, your alphabetic text is delivered to the computer in upper case anyhow; the difference is simply to make it easier to distinguish between your thoughts and those of MTS. (The Teletypewriter has only upper case, and this question doesn't arise).

Don't worry if you want to type an MTS command like $UNNUMBER on a numbered line instead of at the extreme left of the page. MTS will cope.

> Rule **28:** If you try to remove information from a file by $DESTROY or $EMPTY, MTS will state what it intends to do and request confirmation. Type OK or O.K. to confirm. Almost anything else (including a RETURN with no message) cancels the command.

Running Programs.

It is quite possible to compose programs in FORTRAN and enter them from the terminal. Examples 3 and 10 can be typed in for immediate execution if you wish. However unless you are an avid typist you may find it more profitable to enter large programs into files by means of batch runs. In any event, you should not try programming from a terminal without at least reading the earlier sections of this report.

Running Programs (cont'd.)

Although Example 3 can be entered and run from a terminal, only the foolhardy would choose to do so. You are familiar with what happens in a batch run if there is an error in the source program -- the job is rejected and the whole program deck has to be reread after the correction is made. The same is true if you substitute a typewriter for a card reader; however this means that if you try to run a program with an error you will have to retype the whole program again.

Thus the only sensible course of action is to put the source program into a file as in Example 10. Once this has been done, you can run it or amend it from a terminal. Suppose you have done Example 10 and left the source program in the file PROG. Then Example 11 can be run from a terminal whenever you have a new piece of data. The phrase SCARDS=PROG tells WATFIV where to find the source program for re-compilation, while 5=*SOURCE* notifies MTS that the input at execution time is to be handled by the same device as the MTS lines themselves -- in this case the typewriter terminal.

> Rule 29: When entering lines into a file from a typewriter, the text begins after the line number and any separator symbol that accompanies it. Don't forget to give six spaces before "Column 7", and don't forget that the variable length of the line numbers causes a variable left margin to your text as you type it.

One difference in conventions between batch and terminal operations occurs in the running of *FORTRAN. If this is done from a typewriter, the source code is normally not listed; only the erroneous lines and their error messages appear.

> Rule 30: If you want your source program listed when running *FORTRAN from a typewriter terminal, add the phrase PAR=SOURCE to your $RUN *FORTRAN line. Note that there are no asterisks in SOURCE. This rule isn't needed for *WATFIV.

If you are running a job that wants data from the terminal it is sometimes a bit of a dilemma to decide whether the computer is busy executing a job or whether it is waiting for data. As a rule, the typewriter will take a line feed and advance the carriage one space when it wants data. The IBM 2741 locks the keyboard when it doesn't want to hear from you, but the Teletypewriter lacks this ability.

You may want to interrupt the system when it is doing some task. On the IBM 2741 this is accomplished by pressing the ATTN key. In many circumstances the task can be continued (even after some other MTS operations initiated at the keyboard) by typing $RESTART.

You will find that a computer does not work instantaneously, especially when a large number of users are doing tasks that require access to disk files. Thus the last rule is:

> Rule 31: Be patient. Some operations from a terminal, such as $CREATE or $RUN *WATFIV, are not instantaneous, and you may have to wait several seconds for any sign of activity. In fact the loading of WATFIV from the disk takes a few seconds of elapsed time. You can comfort yourself with the thought that during this interval the computer is working on other people's jobs and you aren't being charged for the full cost of the time.

Running Programs (cont'd)

There are a number of useful library programs that can be run from a terminal. A run that combines a couple of these is shown in Example 17. The program *STATUS asks MTS to give you a summary of your computer usage. The program *FILES lists the files that you have currently occupying space on disks.

```
$signon myco pw=sezme
   (SIGNON RESPONSE)
$run *status
   (STATEMENT OF YOUR USAGE)
$run *files
   (LIST OF YOUR FILES)
$signoff
   (SIGNOFF STATISTICS)
```

Example 17.

Naturally either of these programs can also be run as part of a batch job.

7. High-level Hints

This report has left many features of MTS undefined, and many keys of the terminals untouched. Serious users will want to become familiar with the full power of the Command Language and the available processors and library files.

Here are a few topics that deserve further investigation. A full list of references appears immediately after the list of topics.

1. More information on WATFIV and FORTRAN. Refer to UBC WATFIV, U67-3, U67-5, U67-11.

2. Generalized forms of commands like $NUMBER and $CREATE. Refer to U67-13.

3. Manipulation of partial files; concatenation (i.e., linking together) of files. Refer to U67-13.

4. Detailed information on typewriter terminals, including the use of special keys and features. Refer to U67-8.

5. Use of the I.B.M. 2260 Display Terminal. Refer to U67-9.

6. Initiation of Batch Runs from a Terminal. Refer to the Computing Centre write-up for *BATCH.

7. Use of other languages, such as the Assembler, PL/1, PIL, special languages for editing the contents of files, and other special-purpose languages. Refer to the various Computing Centre documents.

References: (all available from the Computing Centre)

1. UBC WATFIV: WATFIV FORTRAN Compiler (August, 1970).
2. U67-0: So You Want to Use the IBM 360/67 (December, 1968).
3. U67-3-1: The Batch User's Guide (January, 1969).
4. U67-5: IBM FORTRAN IV (G) Error Messages (December, 1968).
5. U67-6: Program Interrupts (December, 1968).
6. U67-8: The Conversational Terminal User's Guide (December, 1968).
7. U67-9: IBM 2260 Display Station User's Guide (December, 1968).
8. U67-9-1: Addenda to U67-9 (July, 1970).
9. FORTRAN Input/Output with MTS Line Files (February, 1969).
10. U67-12: The IBM (OS-Compatible) Direct Access Feature (June, 1969).
11. U67-13: A First Look at Files (March, 1969).
12. U67-15: The Debug Package (August, 1969).
13. U67-16: PL/1 Calls to Fortran Subroutines (July, 1970).
14. UBC BATCH: Initiating Batch Runs from a Terminal (October, 1970).