Running MCP XIII on SimH B5500

By Richard Cornwell

07/04/18

Table of Contents

Basic Machine Architecture

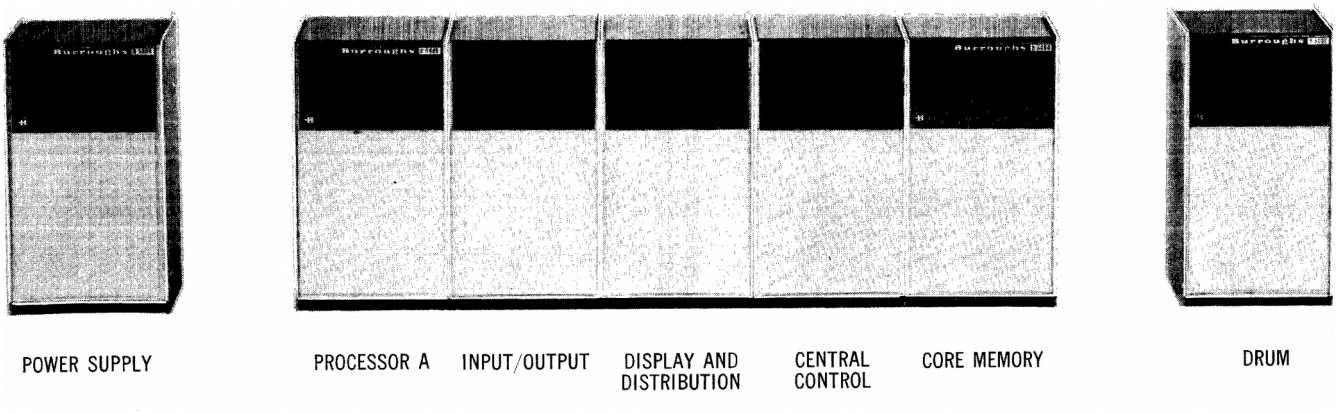

Illustration 1: B5500 Processor A Configuration

The Burroughs B5500 was a 48-bit stack based computer with either one or two CPU's, and up to 32k words of memory. Only the first CPU could preform I/O operations, the second CPU was used only as an auxiliary processor. The system could have up to 4 I/O channels that could talk to any device. All data was accessed via segments, segments could be loaded or swapped out to disk based on usage. This allowed a form of virtual memory. Also the B5500 was programmed mostly in high level languages rather then assembly languages which were common for the time.

The B5500 supported up to 16 tape drives, 10 or 20 disk units, 2 drum units, 2 line printers, 2 card readers, 1 card punch, console typewriter and a terminal controller.

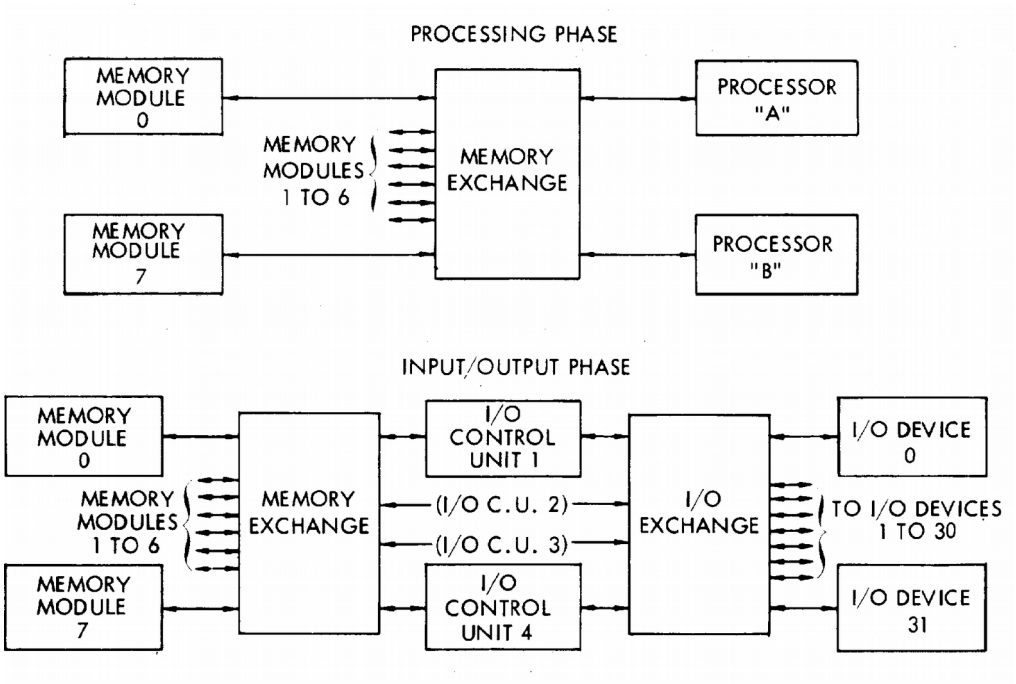

Illustration 2: Basic machine overview

Disks consist of 1 or 2 controllers, and up to 10 disk drives each. Each disk drive had up to 5 Electronic Storage Units to hold data. Disks could also be shared by both controllers to increase access time. The maximum amount of disk space that could be configured per controller was about 960 million characters of data. The B5500 simulator supports two types of disk units, Mod I or Mod IB. ModI held 48 Million Characters, ModIB held 96 Million Characters. The simulator supports up to 2 disk controllers, the default is shared. To run the system in an unshared mode requires the operating system be rebuilt. Disks can be added once the system is built, however they can't be removed.

Under the simulator disks are attached to esu units, the dk units are disk controllers and used to route I/O requests. There is no order requirement for esu units to be added, however once attached they can't be removed without reloading the entire system. Other system units can be enabled or disabled without changing the configuration. The simulator supports the full expanded system if the user desires. Generally 2 esu units are all that is required to make a usable configuration.

Getting required files

Before you can run MCP XIII on the simulator you need to get some files. First you will need to either build the simulator from source or get a binary for your system. The source for the simulator can be found at:

> https://github.com/simh/simh

The development and test version can be found here:

> https://github.com/rcornwell/sims

To build just the B5500 simulator for Linux or BSD, you can use the command:

```
git clone http://github.com/simh/simh simh
cd simh
make b5500
```

This will make in the BIN directory the file b5500.

Next you will need to get the system tapes for MCP. Burroughs became part of Unisys Corporation in 1986. Unisys still owns and maintains copyrights for the B5500 system software. There is an educational/hobbyist license from Unisys to use the Mark XIII release of that software and make it available to others. It is *not* open source software, however, and not officially part of this project, so this cannot include it on the open-source project site with the rest of the emulator files.

The Burroughs Mark XIII software consists of three binary tape image files from a release in 1971. Paul Kimpel and others are making these files available through a page on Paul's hosting service. Go to that page, review and accept the licensing terms (they are not onerous), and download one or more of the tape images.

At the minimum you will need to download the SYSTEM tape. If you wish to rebuild any of the system programs you will need to download the files SYMBOL1 and SYMBOL2. The links are:

- http://www.phkimpel.us/B5500/webSite/TapeImage-SYSTEM.html
- http://www.phkimpel.us/B5500/webSite/TapeImage-SYMBOL1.html
- http://www.phkimpel.us/B5500/webSite/TapeImage-SYMBOL2.html

To automatically rebuild or load MCP on simH B5500 simulator, you will want to download the files from the following location:

- http://sky-visions.com/burroughs/xiii/kit.zip

Or the files individually:

File	Description
CUBEB13.bcd	Cube tape B 13 also available at github retro-b5500 software.
CUBE_LBR-CORR.bcd	Cube library tape, also available at github retro-b5550 software.
README.txt	Readme documentation on these files.
build1.job	First job to rebuild MCP XIII from source.
build2.job	Second job to rebuild DCMCP which is running during job 1.
extra.job	Job to load in various useful CUBE source files and rebuild them.
load.job	Job to reload MCP without rebuilding. Also used extras.bin to load in the CUBE files.
tss_setup.job	Job to set up accounts for DCMCP remote access and TSSMCP remote access.
xiii_build.ini	SimH Init file that rebuilds MCP from source.
xiii_load.ini	SimH Init file that reloads MCP from SYSTEM tape.
xiii_run.ini	SimH Init file start execution of MCP after the system has been loaded via either load strip.
xiii_tss.ini	SimH file to run tss_setup.job and prepare MCP to run CANDE timesharing.
xiii_test.ini	Runs a series of jobs to stress the simulator and also verifies that remote access works.
stress.job	Stress test job stream
set_test.job	Job to set up for remote access test.

You will also want to go to www.bitsavers.org and download some documents to help you use the system, here is a list of helpful documents:

Link	Description
1024916_B5500_B5700_OperMan_Sep68.pdf	Basic operating manual for MCP. This is for an older version of MCP but most of the commands and procedures are the same.
1038205_B5500_TS_TerminalUG_Jun69.pdf	Good introduction to TSS/MCP CANDE usage.
Operating_the_B5500_Time-Sharing_System.pdf	Another operational manual.
1058583_B5700_TSS_RefMan_Sep72.pdf	Time Sharing system reference manual.
AA119117_B5700_MCP_Ref_1972.pdf	DCMCP system reference manual.
1028024_B5500_ExtendedAlgol_Apr69.pdf	XALGOL language document.
1031986_B5500_Handbook_Aug70.pdf	Good reference manual for MCP data structures and commands.
1051182_B5700_FORTRAN_Jan71.pdf	Fortran language document.
1032638_B5500_ESPOL_RefManOct67.pdf	ESPOL language document.
1038643_B5500_CompatibleAlgol_May69.pdf	Algol language document.

Operating commands

The first time you load MCP you will be greeted with the following display:

```
R -H/L WITH MCP/DISK MARK XIII MODS RRRRRRRR-
R  TIME IS 0000
R  DATE IS WEDNESDAY, 9/ 1/71
R #TR PLEASE
I TR1001
R  TIME IS 1001
```

The “R ” in front of the line is typed by the simulator. On the original Burroughs B5500 the console was a half duplex device. In order to enter data into the system, the operator needed to press the “Input Request” key. This interrupted the system which would light the “Ready” light. After the operator was done entering the message he would press “End of Message”. On the simulator this is simulated by the “Escape” key. This is equivalent to the “Input Request” key on the console. When the system is ready to receive input it responds with “I “ and waits for the user to type input. When done typing a message press the “Enter” or “Return” key to simulate the “End of Message” key. This will send the message to the system. “Backspace” can be used to correct mistakes. Pressing “Escape” again will abort the typing.

MCP puts a “#” before commands that require a response. It will continue periodically to repeat the question until the user answers. The system will generally ask for the time and or the date. These can be controlled by options. See the section on Cold and Warm Boot decks.

In the message above, MCP announces itself and indicates that it started with a Halt/Load. Then which MCP was loaded. MODS indicates which memory modules are online. Generally this will always be all R's. For the initial load MCP will report each Storage unit attached to the system. Finally it prints what it believes is the time and date. MCP will not do anything until the date and time are set. So press “Escape” and at the “I” prompt enter “TR hhmm<return>” to set the time and get MCP to work.

If there is input attached to the card reader MCP will start reading them in. Otherwise attach a deck to the card reader to run jobs on MCP. Do this by Pressing Control and E to interrupt the simulator. SimH will print a “sim>” prompt. Type “attach cr0 deck” to attach a deck and “continue” to continue the simulation. MCP will then report “BOJ” and “EOJ” messages as jobs are started and ended. Some jobs use what are called “PSUEDO-READERS”, it is generally a good idea after starting MCP to enable a few of these. You can do this with the “RN#” command. Typically “RN2” is a good number.

MCP runs jobs from what it calls the MIX. To find out what jobs are running you can use the “MX” command.

Some useful commands, for a complete list see “Operating the B5500 Time Sharing System”.

Command	Description
CC control information	Starts a job from console.
CE	Starts time sharing.
CI file	Sets the current intrinsics file.
CM file	Sets the next MCP to run at boot time.

CU	Displays amount of memory used by the jobs in the current mix.
<mix index> **DS**	Terminate a job in the mix.
DT <integer>/<integer>/<integer>	Sets the date.
MF #	Changes TSS space used to #. Default=16348.
MU #	Changes maximum number of time-sharing users that can be logged in a one time
MX	Prints out a list of currently running programs.
PD =/= **PD** <file>**/=** **PD** <file>/<file>	Lists files on the disk.
<mix index> **RM**	Allow job to overwrite an existing file.
RN #	Specifies the number of Pseudo readers to run.
RY device	Resets device.
SF <fractional value>	Indicates the amount of memory to allow the system to take.

Basic Messages

Basic messages from the system consist of several formats.

<program name>=<mix index>

Identifies that the message is from the given program at the specific mix. Common messages are BOJ, EOJ indicating that the job is starting or ending.

Messages that start with "#" indicate that the system needs some information to continue.

If you see a message like:

#NO FIL <*name*>

You will have several choices to respond to this command. You can attach a tape with <name> label on it. If the program is trying to create a file you can attach a blank tape and do "**PB MT**x" to declare the tape as blank. The system will grab the first blank tape for this. If you don't have a file/tape then you will be forced to "**DS**" the job which will kill it.

#DUP LIBRARY <*file*> : <*mix index*>

This message occurs when a job tries to overwrite an existing file. You can either rename the file to another name, or remove it and enter "<*mix index*>**OK**" message. If it is ok to delete the file you can enter "<*mix index*>**RM**" which will remove the file and let the job continue. If there is some error you can "<*mix index*>**DS**" to abort the job.

```
# MT RQD <file> <rdc> : <mix index>
```
This message indicates that the job is looking to write a file to a tape, and you should mount a blank tape on an available drive and issue “PB MTx”.

If you want to enter a command to be executed from the operator console you can use the “CC” command. Multiple commands can be separated by “;”’s the last command should be “END”. The system will continue asking for input until the END card is given.

When jobs fail you might see something like this displayed:

```
-STACK OVRFLW ALGOL/ESPOL= 4,NEAR LINE 1736200
```

This indicates that the stack pointer was moved below it’s limit. You can try increasing the stack size of the job or checking for programming errors.

Other errors that might be reported may have “S= #, A=#, ...” this indicates the line that caused the failure. If you look at the listing, you will see Segments listed, the S=# gives you the segment number to look at and the A=# gives you the address within the segment where the fault occurred.

Starting Time-sharing

To enable time-sharing if the system was set up to run DCMCP, boot into the system and after entering time and date, enter:

CM TSS/MCP

MCP will tell you that this is to be the next MCP run. You can reboot now. You will now get this prompt.

```
R -H/L WITH TSS/MCP MARK XIII,F=16384[MODS=RRRRRRRR]-
R  TIME IS 1526
R  DATE IS WEDNESDAY, 9/ 1/71
R #TR PLEASE
I TR2218
R  TIME IS 2218
```

Next change the intrinsics file to TSS/INT with:

```
I CI TSS/INT
```
To start timesharing you need to enter the “CE” command. This will display the following:

```
I CE
R
R  0:CANDE/TSHARER/SITE= 1 BOJ 2220
R USERS/CANDE FILE DATED 00090171
```

At this point you can telnet into the port you have set in xiii_run.ini file.

Loading The Operating System

There are two methods of loading MCP onto simulated disks. If you do not wish to rebuild the system you only need to download the SYSTEM tape. Otherwise you will want to download all the distribution file and all the files from my site. At the bare minimum you will require the “cold.deck” and the SYSTEM tape to reload the system.

The .ini files set up the configuration of the system. You can start the simulator and give the correct files to the system to cause it to load or run.

The basic method of reloading the system is to use the cold start deck (cold.deck) to build the file system and load the operating system kernel onto the disks. Once this is done the rest of the system is reloaded under MCP itself. The file is ASCII text and can be edited with any text editor. The first card is the bootstrap loader, this must be constructed by hand since while the B5500 card readers could load binary images, the punch could not do this. In the build section I will explain how to rebuild this deck. The COLD start loader begins at card 2, and continues through the lines starting with “000000”, do not modify these lines.

The first configuration line starts with DRCTRYTP this is the first block that MCP can use to store files. MCP uses this space to store overlays and other information, it can be set lower then 2000, however I would not recommend this. The next entry DIRECT tells how many segments to use to store files. If you add more ESU units you may wish to increase this. Currently the configuration is set at 2 ESU’s which allows about 200 million characters, which is about twice the size of the system I have configured.

If you change DIRECT (either larger or smaller), you will have to adjust the FILE entries. The logic here is that you have:

```
FILE <name>/<type>, <segments>|<size>, 999
                                                    <end>
```

The <end> of each FILE should be at <size>+<end previous>+1 of previous file. The first starts at DRCTRYTP.+4.

Following the files are a series of flags that can be set or not set. Check out the “B5500 B5700 OperMan Sep 68” Chapter 3 starting at page 3-20 to understand in more detail how to set these up.

Some options to add or remove, TYPE DATE will force the system to ask for the date, TYPE TIME causes the system to ask for the current time. Many of these options can be controlled in the currently running MCP with the SO and RO commands.

After the last option the basic MCP loader deck should be inserted, followed by a STOP card. Next the TAPEDSK program is included to copy MCP/DISK from the SYSTEM tape to the MCP/DISK file. After COLD runs it will reboot system which will load the TAPEDSK program, following this the system will boot into MCP and run the following commands on the card reader.

To reload the system without recompiling it you can use the xiii_load.ini file or run it manually if not on a simH simulator. To run this manually attach the SYSTEM tape to mt0, and two or more disk units to esu0,esu1, and the cold.deck to the card reader. Next boot the card reader. At the #TR prompt enter the time with the TR command. Wait until the “LIBMAIN/DISK= 1 EOJ” and then enter a “CI INT/DISK” command to set the intrinsic file. Next to load in the extras

enter “CC USER RON; LOAD FROM EXTRAS =/=; END”. Then attach the extras.bin tape to tape drive mt0. After the next “LIBMAIN/DISK= 1 EOJ” message the system is loaded and you can continue to use it. Booting “dk” will restart the system.

To rebuild the system from source you will need to run the xiii_build.ini with the simulator. This consists of 3 jobs that are submitted to the card reader. To do this manually is a bit harder, boot the cold.deck, after the SYSTEM is loaded, set the Intrinsics as above, also enable 1 pseudo-reader with “RN1”, then attach the build1.job to the card reader. Replay to the various “MC x/NEW” messages as they appear. When you see the “REBOOT NOW” message, change to the alternate MCP with “CM MCPA/DISK” and reboot. Set intrinsics to “CI NEW/INT” and one Reader, then run build2.job. This job rebuilds the base MCP. After the “REBOOT NOW” message switch MCP with “CM MCP/DISK” and reboot. When the system reboots you can now set the intrinsics with “CI INT/DISK” and readers to 1. Attach extra.job to the card reader if you wish to load in the optional CUBE software. Mount the tapes as the system asks for them with #NO FILE xxx FILE000”.

To set up for timesharing (CANDE), you can attach the job “tss_setup.job” to the card reader. This job can be edited to add in more accounts as needed. Currently there are a few (B5500, USER1, USER2 and GUEST).

To use remote access under DC/MCP you need to run the UPDATE/USERS program and update the users file.

After starting the system with xiii_tss.ini or running tss_setup.job you will need to switch to TSS/MCP in order to run timesharing. Do this by first issuing a “CM TSS/MCP” command. After the “NEXT MCP WILL BE ...” message you can reboot the machine. After entering the time you will need to do “CI TSS/INI” which will reply with “NEW INTRINSICS….”, then the “CE” message can be given to start Cande. Note this first two commands only need to be given once to switch to TSS/MCP once switched all reboots will start TSS/MCP instead of MCP/DISK. Just the “CE” command need be given after each reboot to start up timesharing.

Time-sharing setup.

To setup for timesharing a couple of jobs need to be run. One needs to run SYSDISK/MAKER this defines the lines available. The first file should be “LINE,0,0,<buffer>,0,0,<lines>,0,” then for each line there should be two lines:

```
LINE,1,<#>,<buffer>,0,0,0,0
STA,0, 0, 0, 0, “0”, “0”, 0, 0,
```

The second line is optional if this is a teletype type of interface. <buffer> can be 28, 56 or 112. And should match the value set on the DTC device.

Users are created with the USER/CANDE program. The first card is the option card.

- $ NEW causes new user accounts to be created. Or update existing users.
- $ OPTIONS updates the given users.
- $ USER “<code>” updates a given user.
- $ REMOVE “<code>” removes the given user.

After the $USER card is given the options for that user can be set:

- PASSWORD “<password>” Sets the password for the given user.
- NAME “<name>” Sets the name to be displayed at login.
- TIME “<24 0’s or 1’s>” Restrict time a user can use the machine. 1 indicates Ok, 0 indicates no access.
- CHARGE “<charge code>” Sets the charge code for the user.
- REQUEST CHARGE Will prompt the user at login time for a charge code.
- NO CHARGE No charge code is used for this user.
- LANGUAGES <list> Sets which languages the user can’t use.
- VERBS <list> Sets a list of commands that the user can’t user.
- PHONE “<number>” Sets the phone number for the user.

Time-sharing operations

The timesharing system is a typical line oriented editor. A full description of this system can be found in “1038205_B5500_TS_TerminalUG_Jun69.pdf”. The simulator attached the DTC device to a given port and TSS/MCP needs to be running to access timesharing. “Telnet” into the machine at <host> <port> and you will be presented with a login prompt after a couple of seconds:

```
Connected to the B5500 simulator DTC device, line 0


B5500 TIME SHARING - 01/00, STATION 02
ENTER USER CODE, PLEASE-B5500
AND YOUR PASSWORD
@@@@@@@
09/01/71  8:52 PM.
GOOD EVENING, USER NAME       YOU HAVE STATION 02
```

You user code must be in the system along with a password. Enter these to get started. To log off use the “BYE” command.

There are some characters that can’t be entered into the editor.

Card	ASCII	Algol	Basic
<	<	LSS	|LS
>	>	GTR	|GT
≤	{	LEQ	|LE
≥	}	GEQ	|GE
≠	!	NEQ	|NE
←	~	:=	:=

Commands either accept a file or a line number. Line numbers are specified either individually or by a list separated by commas or dashes to indicate ranges. “ALL” can be used in place of line number to indicate the whole file, or “END” or “<line> TO END” to indicate from line to end. Some common used commands are as follow: Commands that specify a <file>, the file can be omitted to indicate the current work file.

Command	Description
CHANGE PASSWORD	Change your password
CHANGE <file> TO <file>	Rename a file.
CHANGE <file> TYPE <type>	Change type of file.
COMPILE <file>, <compiler>	Compile file with Compiler. Both arguments can be omitted.
COPY	See below for explanation of options.

CREATE <file> <type> SIZE <n>	Create a new file of type <type> and <n> lines.
DELETE <line>	Delete lines from a current file.
EXECUTE <file>	Execute a file.
FILES	Lists files for user.
FIX	Line editing, see manual for details.
LIST <file> <line>	Lists a file, over given range.
LIST FILES	Lists files for user.
LIST <line>	Lists current file lines.
LOAD <file>	Load a file into edit buffer.
REMOVE <file>	Removes a file.
RENAME <file>	Renames the current file to <file>
RESEQ <file> <start>+<inc>	Updates the line numbers of a file see the manual for more options.
RUN <file>	Compiles and Runs a file.
SAVE	Saves the current edit buffer.
SEQ <start>+<inc>	Allows input starting at <start> incremented by <inc>.

This is a sample session running an Algol job.

```
Connected to the B5500 simulator DTC device, line 0


B5500 TIME SHARING - 01/00, STATION 02
ENTER USER CODE, PLEASE-B5500
AND YOUR PASSWORD
@@@@@@@
09/01/71  8:52 PM.
GOOD EVENING, USER NAME       YOU HAVE STATION 02
```

<First try at program>

```
CREATE SIEVE ALGOL
FILE:SIEVE - TYPE:ALGOL  -- CREATED
SEQ 100100+100
100100BEGIN
100200INTEGER ARRAY CANDIDATE[0:1000];
100300INTEGER I,J,K;
100400FOR I:=0 STEP 1 UNTIL 1000 DO
100500BEGIN
100600   CANDIDATES[I]:=1;
100700END;
100800CANDIDATE[0]:=0;
100900CANDIDATE[1]:=0;
101000I:=0;
```

```
101100WHILE I < 1000 BEGIN
101200   WHILE I < 1000 AND CANDIDATE[I] = 0 DO I:=I+1;
101300   IF I<1000 THEN
101400    BEGIN
101500      J := 2;
101600      K := J*I;
101700      WHILE K < 1000 DO BEGIN
101800        J :=J + 1;
101900        K := J*I;
102000      END;
102100      I:=I+1;
102200   END;
102300END;
102400FOR I:=0 STEP 1 UNTIL 999 DO
102500BEGIN
102600IF CANDIDATE[I] ! 0 THEN
102700   BEGIN
102800      WRITE (I);
102900   END
103000END.
103100
#
COMPILE
 WAIT.

 COMPILING.
NEAR LINE 100600 ERROR NUMBER 100 -- CANDIDATES.
NEAR LINE 101100 ERROR NUMBER 132 -- BEGIN.
NEAR LINE 102800 ERROR NUMBER 439 -- I.
NEAR LINE 103000 ERROR NUMBER 120 -- END.
NEAR LINE 103000 ERROR NUMBER 143 -- END.
NEAR LINE 103000 ERROR NUMBER 145 -- END.
```

<Correct some errors to see if the program will work>

```
 ERR COMPILE .0 SEC.

100600  CANDIDATES[I]:=1;
101100 WHILE I < 1000 DO BEGIN
101750        CANDIDATE[K] := 0;
100350 FILE OUT DC 19 (1,10);
100460 FORMAT F1 (I4)
10280        WRITE (DC, F1, I);
DELETE 10280
 WAIT.1 RECORDS DELETED.


 END DELETE .0 SEC.

102800     WRITE(DC, F1, I);

COMPILE
 COMPILING.
NEAR LINE 100460 ERROR NUMBER 146 -- FORMAT.
NEAR LINE 100500 ERROR NUMBER 000 -- B.
NEAR LINE 100800 ERROR NUMBER 143 -- CANDIDATE.
NEAR LINE 100800 ERROR NUMBER 144 -- 0.
NEAR LINE 102800 ERROR NUMBER 100 -- F1.
```

```
NEAR LINE 103000 ERROR NUMBER 120 -- END.
NEAR LINE 103000 ERROR NUMBER 143 -- END.
NEAR LINE 103000 ERROR NUMBER 145 – END.

 ERR COMPILE .0 SEC.

100360 FORMAT F1 (I4);
DELETE 100460
 WAIT.

 WAIT.COMPILE
1 RECORDS DELETED.


 END DELETE .0 SEC.

 COMPILING.
NEAR LINE 100600 ERROR NUMBER 100 -- CANDIDATES.
NEAR LINE 103000 ERROR NUMBER 120 -- END.
NEAR LINE 103000 ERROR NUMBER 143 -- END.
NEAR LINE 103000 ERROR NUMBER 145 -- END.


 ERR COMPILE .0 SEC.

100600    CANDIDATE[I]:=1;
COMPILE
 WAIT.

 COMPILING.
NEAR LINE 103000 ERROR NUMBER 120 -- END.
NEAR LINE 103000 ERROR NUMBER 143 -- END.
NEAR LINE 103000 ERROR NUMBER 145 -- END.


 ERR COMPILE .0 SEC.
```

<Got most of errors out, print whole program to figure out why end are in error>

```
LIST


FILE:SIEVE - TYPE:ALGOL  --09/01/71  9:03 PM.

100100 BEGIN
100200 INTEGER ARRAY CANDIDATE[0:1000];
100300 INTEGER I,J,K;
100350  FILE OUT DC 19 (1,10);
100360  FORMAT F1 (I4);
100400 FOR I:=0 STEP 1 UNTIL 1000 DO
100500 BEGIN
100600     CANDIDATE[I]:=1;
100700 END;
100800 CANDIDATE[0]:=0;
100900 CANDIDATE[1]:=0;
101000 I:=0;
101100  WHILE I ? 1000 DO BEGIN
101200    WHILE I ? 1000 AND CANDIDATE[I] = 0 DO I:=I+1;
```

```
101300    IF I?1000 THEN
101400     BEGIN
101500       J := 2;
101600       K := J*I;
101700       WHILE K ? 1000 DO BEGIN
101750         CANDIDATE[K] := 0;
101800         J :=J + 1;
101900         K := J*I;
102000       END;
102100       I:=I+1;
102200    END;
102300 END;
102400 FOR I:=0 STEP 1 UNTIL 999 DO
102500 BEGIN
102600 IF CANDIDATE[I] ? 0 THEN
102700    BEGIN
102800      WRITE(DC, F1, I);
102900    END
103000 END.


 END QUIKLST .0 SEC.

103000 END;
103100 END.
COMPILE
 WAIT.

 COMPILING.


 END COMPILE .0 SEC.

SAVE
FILE:SIEVE - TYPE:ALGOL  -- SAVED.
```

<Finally compiled without error, see it if works>

```
RUN
 RUNNING



-INTGR OVRFLW, LINE NO 101700


 ERR SIEVE .0 SEC.
```

<Nope, make some corrections>

```
101100 WHILE I LSS 1000 DO BEGIN
101200     WHILE I LSS 1000 AND CANDIDATE[I] = 0 DO I:=I+1;
101300     IF I LSS 1000 THEN
10170      WHILE K LSS 1000 DO BEGIN
DELETE 10170
 WAIT.

 WAIT.1 RECORDS DELETED.
```

```
 END DELETE .0 SEC.

101700    WHILE K LSS 1000 DO BEGIN
102600 IF CANDIDATE[I] NEQ 0 THEN
SAVE
 WAIT-

FILE:SIEVE - TYPE:ALGOL  -- SAVED.

COMPILE
 COMPILING.


 END COMPILE .0 SEC.

RUN
 RUNNING



-INTGR OVRFLW, LINE NO 101700


 ERR SIEVE .0 SEC.
```

<Nope, lets try a different algorithm>

```
100400 FOR I:=2 STEP 1 UNTIL 1000 DO
100500    CANDIDATE[I] := 1;
SAVE
 WAIT-

FILE:SIEVE - TYPE:ALGOL  -- SAVED.

RUN
 COMPILING.


 END COMPILE .0 SEC.

 RUNNING



-INTGR OVRFLW, LINE NO 101700


 ERR SIEVE .0 SEC.


DELETE 102100
 WAIT.1 RECORDS DELETED.


 END DELETE .0 SEC.

COMPILE
```

```
 COMPILING.
NEAR LINE 102300 ERROR NUMBER 143 -- END.


 ERR COMPILE .0 SEC.

DELETE 101950
 WAIT.1 RECORDS DELETED.


 END DELETE .0 SEC.

DELETE 102000
 WAIT.1 RECORDS DELETED.


 END DELETE .0 SEC.

COMPILE
 COMPILING.


 END COMPILE .0 SEC.

RUN
 RUNNING


   3
....
 997
 998
 999


 END SIEVE .0 SEC.
```

<Getting better>

```
LIST


FILE:SIEVE - TYPE:ALGOL  --09/01/71  9:35 PM.

100100 BEGIN
100200 INTEGER ARRAY CANDIDATE[0:1000];
100300 INTEGER I,J,K,M;
100350  FILE OUT DC 19 (1,10);
100360  FORMAT F1 (I4);
100370  FORMAT F2 (3I4);
100400  FOR I:=2 STEP 1 UNTIL 1000 DO
100500     CANDIDATE[I] := 1;
100800 CANDIDATE[0]:=0;
100900 CANDIDATE[1]:=0;
101000  FOR I:=1 STEP 1 UNTIL 1000 DO
101100  BEGIN
101200    IF CANDIDATE[I] NEQ 0 THEN
101250          I:=I+1;
101300    BEGIN
```

```
101400      J:=I; K:=J+J;
101500      IF K LSS 1000 THEN
101600      BEGIN
101700         FOR M:=K STEP J UNTIL 1000 DO
101750         CANDIDATE[K] := 0;
101800            CANDIDATE[I] := 0;
101900      END;
102200    END;
102300 END;
102400 FOR I:=0 STEP 1 UNTIL 999 DO
102500 BEGIN
102600  IF CANDIDATE[I] NEQ 0 THEN
102700    BEGIN
102800      WRITE(DC, F1, I);
102900    END
103000  END;
103100  END.


 END QUIKLST .0 SEC.

DELETE 101750
 WAIT.1 RECORDS DELETED.


 END DELETE .0 SEC.

101800         CANDIDATE[M] := 0;

DELETE 101250
 WAIT.1 RECORDS DELETED.


 END DELETE .0 SEC.

DELETE 100370
 WAIT.1 RECORDS DELETED.


 END DELETE .0 SEC.


RUN
 COMPILING.


 END COMPILE .0 SEC.

 RUNNING


   2
   3
   5
   7
....
 977
 983
 991
```

```
 997


 END SIEVE .0 SEC.
```

<Now it is working!>

```
SAVE
FILE:SIEVE - TYPE:ALGOL  -- SAVED.

LIST FILES
09/01/71 B5500          09:37 PM
NAME    TYPE     RECS SEGS   CREATED   ACCESSED  W/R  W/B  S-F LOCKD BY
/APLIBE UNKNOWN   800  160  09/01/71 * 09/01/71  120  120  100
SIEVE   ALGOL      30   10  09/01/71 * 09/01/71   10  300    7
SIEVE   OBJ(A)     16   16  09/01/71 * 09/01/71   30   30    7
       3  FILES        186  SEGMENTS     846  RECORDS


 END LFILES .0 SEC.

LIST


FILE:SIEVE - TYPE:ALGOL  --09/01/71  9:44 PM.

100100 BEGIN
100200 INTEGER ARRAY CANDIDATE[0:1000];
100300 INTEGER I,J,K,M;
100350  FILE OUT DC 19 (1,10);
100360  FORMAT F1 (I4);
100400  FOR I:=2 STEP 1 UNTIL 1000 DO
100500     CANDIDATE[I] := 1;
100800 CANDIDATE[0]:=0;
100900 CANDIDATE[1]:=0;
101000  FOR I:=1 STEP 1 UNTIL 1000 DO
101100  BEGIN
101200    IF CANDIDATE[I] NEQ 0 THEN
101300    BEGIN
101400      J:=I; K:=J+J;
101500      IF K LSS 1000 THEN
101600      BEGIN
101700         FOR M:=K STEP J UNTIL 1000 DO
101800          CANDIDATE[M] := 0;
101900      END;
102200    END;
102300 END;
102400 FOR I:=0 STEP 1 UNTIL 999 DO
102500 BEGIN
102600  IF CANDIDATE[I] NEQ 0 THEN
102700    BEGIN
102800      WRITE(DC, F1, I);
102900    END
103000  END;
103100  END.


 END QUIKLST .0 SEC.
```

Submitting batch jobs.

Batch jobs can be submitted either via cards or via time sharing or via R/C under DCMCP. All batch commands begin with a “?” character. To run a job under a specific user the first card should be “?USER <name>”. The two primary batch commands are “?COMPILE” or “?EXECUTE”. Following these cards comes the “?FILE” cards to specify how files are to be accessed. Typical files are “CARD” for card input, “TAPE” for input from tape/disk, or “LINE” which specifies where the output of the program will go.

The common structure of a compile deck is as follows:

```
?COMPILE <executable> [WITH] <language> [FOR] <option>
```

The words WITH and FOR are optional. <option> can be “GO” which causes the program to be execute if compile is successful. “LIBRARY” if you just wish to compile the program to be executed at a later date. “SYNTAX” compiles the program but does not generate any output.

```
? <language> FILE LINE = PRINT BACK UP DISK
```

This caused the output of the compiler to be printed on the printer.

```
? <language> FILE TAPE = <source> SERIAL DISK
```

Indicates that the input will also come from a tape file and be merged in with what is on the card reader.

```
? <language> FILE NEWTAPE = <new source> SERIAL DISK
```

Optional card the creates a new version of the source file merged with the input cards. Also the first card “the dollar card” has to include “NEW TAPE” option.

```
? FILE LINE = PRINT BACK UP DISK
```

Sends the output of the program to the printer.

```
? DATA CARD
```

This indicates that the following will be read as data until the next ? card. For a compiler the next card is typically a $ card which specifies the compiler options. Typical this might be:

```
$ CARD LIST SINGLE
```

This will take input only from the card deck and list the program single space. The default listing is double spaced. Some other common options are “PRT” to list program segments in the listing. This will help in debugging. “TAPE” to merge the card deck with the TAPE file. “NEW TAPE” to create a new merged deck for later compile.

Following the program there can be another ? DATA <name> card to give the program more data. Or an ?END card to terminate the input.

The following is an example program that will copy the file “CARD” to the file “NEWTAPE”.

```
?COMPILE O/RDR WITH ALGOL FOR LIBRARY
?ALGOL FILE LINE = PRINT BACK UP DISK
?DATA CARD
$CARD LIST SINGLE
BEGIN                                                                   0001000
LABEL EOF;                                                              0014000
SAVE  ARRAY REC[0:15];                                                  0015000
FILE IN CARD (2, 10);                                                   0016000
SAVE FILE OUT  NEWTAPE DISK SERIAL [20:3000] (2,10,150,SAVE 99);        0018000
                                                                        0020000
WHILE TRUE DO                                                           0027000
  BEGIN                                                                 0028000
  READ(CARD, 10, REC[*]) [EOF];                                         0030000
  WRITE(NEWTAPE, 10, REC[*]);                                           0034000
  END WHILE;                                                            0036000
                                                                        0037000
EOF:                                                                    0038000
LOCK(NEWTAPE,RELEASE);                                                  0040000
CLOSE(CARD)                                                             0042000
END.                                                                    0045000
END.       LAST CARD ON 0CRDING  TAPE                                   9999999
?END
```

The program can be run with the ?EXECUTE card as follows:

```
?EXECUTE O/RDR
?ALGOL FILE NEW TAPE = SYMBOL/RDR SERIAL DISK
?DATA CARD
BEGIN                                                                   0001000
LABEL EOF;                                                              0014000
SAVE  ARRAY REC[0:15];                                                  0015000
FILE IN CARD (2, 10);                                                   0016000
SAVE FILE OUT  NEWTAPE DISK SERIAL [20:3000] (2,10,150,SAVE 99);        0018000
                                                                        0020000
WHILE TRUE DO                                                           0027000
  BEGIN                                                                 0028000
  READ(CARD, 10, REC[*]) [EOF];                                         0030000
  WRITE(NEWTAPE, 10, REC[*]);                                           0034000
  END WHILE;                                                            0036000
                                                                        0037000
EOF:                                                                    0038000
LOCK(NEWTAPE,RELEASE);                                                  0040000
CLOSE(CARD)                                                             0042000
END.                                                                    0045000
END.       LAST CARD ON 0CRDING  TAPE                                   9999999
?END
```

This would store the source for the program on disk. The generated system image comes with a much more functional version of this program called “OBJECT/READER”, which by default will copy the file “S” to “LINE”, this is a simple way to print out program source. By using the “?COMMON = #” option card after the “?EXECUTE” card the operation of the program can be controlled. “?COMMON = 1” copies “CARD” to “LINE”, this can be used to print a card deck or copy cards to a file.“?COMMON = 2” copies the file from “S” to both “LINE” and “NEWTAPE”, this can be used to copy a source file to a new name. “?COMMON =3” copies “CARD” to “LINE” and “NEWTAPE” this is the preferred way to load a source deck. For example as follows:

```
?EXECUTE OBJECT/READER
?COMMON = 3
?FILE NEWTAPE = <source> DISK SERIAL
?DATA CARD
<program>
?END
```