Number: 3, 1974 April

Contents:

From the editor page 1

Technical contributions

Jack M. Schwartz and Donald S. Wyner, Use of the Spasm Software Monitor to Evaluate the Performance of the Burroughs B6700 " 2

K.J.E. Lewis, New option for system/dumpanalyzer " 19

R.J.M. van Eyndhoven, Hardware for digital plotting on the B6700 " 22

L.R.A. Kessener and H. Willemsen, A drawing system based on a formal picture concept " 33

Notes " 60

Mailing list " 62
JUB6700, number 3, 1974 April

Editor
Harry P.J.M. Roumen
Eindhoven University of Technology
Computing Centre
P.O. Box 513
Eindhoven, Netherlands

Secretary
Miss Tiny Fransen
Eindhoven University of Technology
Computing Centre
P.O. Box 513
Eindhoven, Netherlands

JUB6700 is an informal irregularly appearing publication for the users of the Burroughs 6700, for the time being edited at the Computing Centre of Eindhoven University of Technology.
The contents of JUB6700 will mainly consist of technical papers; contributions to JUB6700 may be sent to the editor. Where possible the papers should be typed and camera-ready.
Without charge two copies are sent to the "manager" of each B6700-installation and also to some Burroughs-representatives.
Matters pertaining the mailing list should be sent to the secretary.
From the editor.

In the second issue of JUB6700 we published the article "New option for system/dumpanalyzer" by K.J.E. Lewis. Due to technical circumstances the given patch listing wasn't printed complete. Therefore the article is published again in this issue.

In the future we would like to have some publications about performance measurements. To start with it we have published in this issue the article "Use of the SPASM software monitor to evaluate the performance of the Burroughs B6700" by Jack M. Schwartz and Donald S. Wyner. This article is published before in the proceedings of the National Computer Conference 1973, pp. 93-100.

Two articles in this issue describe the hardware interface and the software necessary for on-line plotting on the B6700 as developed at the Eindhoven University of Technology. To be complete we would like to refer to the article "Use of a digital plotter" by P. Voss (ABCU-newsletter no. 45, December 1973) in which the implementation of the Danish Atomic Energy Commission is described.

Once again I request each B6700-user (especially those users who promised me) to reconsider how far there is potential knowledge valuable for other B6700-users.

Seen in Formatter 03391000-03393000:

Most of our misfortunes are more supportable than the comments of our friends upon them.

CALEB COLTON
USE OF THE SPASM SOFTWARE MONITOR TO EVALUATE THE PERFORMANCE OF THE BURROUGHS B6700

by JACK M. SCHWARTZ and DONALD S. WYNER

(Federal Reserve Bank of New York, New York, New York).

INTRODUCTION

The need for system performance measurement and evaluation.

The benefit to be derived from a large multi-purpose system, such as the B6700, is that many jobs of very diverse characteristics can (or should) be processed concurrently in a reasonable period of time. Recognizing that certain inefficiencies may result from improper or uncontrolled use, it is necessary to evaluate the computer system carefully to assure satisfactory performance. To this end, the objective of our work in the area of performance evaluation is to:

1. determine the location(s) and cause(s) of inefficiencies and bottlenecks which degrade system performance to recommend steps to minimize their effects;
2. establish a profile of the demand(s) placed upon system resources by programs at our facility to help predict the course of system expansion;
3. determine which user program routines are using inordinately large portions of system resources to recommend optimization of those routines;
4. establish control over the use of system resources.

Among the techniques which have been applied to date in meeting these objectives are in-house developed software monitors, benchmarking, and in-house developed simulations. This paper discusses the software monitor, SPASM (System Performance and Activity Software Monitor), developed at the Federal Reserve Bank of New York to evaluate the performance and utilization of its Burroughs B6700 system.

*This article is published before in the proceedings of the National Computer Conference, 1973 pp. 93-100.
THE B6700 SYSTEM

The B6700 is a large-scale multiprogramming computer system capable of operating in a multiprocessing mode which is supervised by a comprehensive software system called the Master Control Program (MCP). Some of the features of this system which distinguish it from many other systems are:

- Each task has assigned to it a non-overlayable area of memory called a stack. This area provides storage for program code and data references* associated with the task as well as temporary storage for some data, history and accounting information.

- Multiple users can share common program code via a reentrant programming feature.

- The compilers automatically divide source language programs into variable sized program code and data segments rather than fixed sized pages.

- Core storage is a virtual resource which is allocated as needed during program execution. (This feature is discussed in more detail below.)

- Secondary storage including magnetic tape and head-per-track disk is also allocated dynamically by the MCP.

- Channel assignments are made dynamically; that is they are assigned when requested for each physical I/O operation.

- I/O units are also assigned dynamically.

- Extensive interrupt facilities initiate specific MCP routines to handle the cause of the interrupt.

- The maximum possible B6700 configuration includes 3 processors, 3 multiplexors, 256 peripheral devices, 1 million words of memory (six 8-bit characters per word or 48 information bits per word), and 12 data communications processors.

*These references are called descriptors and act as pointers to the actual location of the code or data.
6 Floating Channels

I/O MPX

Card Reader

Card Punch

Printer

Printer

Processor

Adapter for 16 lines

Adapter for 16 lines

Console Displays

7 Ch

*14 memory modules, each having 93.3 KB; total 1.4 MB.

Figure 1: Configuration of the Federal Reserve Bank of New York B6700 computer system.
The current B6700 system at the Federal Reserve Bank of New York shown in Figure 1 includes one processor, one I/O multiplexer with 6 data channels, one data communications processor and a number of peripheral devices. In addition, the system includes a virtual memory consisting of 230,000 words of 1.2 micro-second memory, and 85 million words of head-per-track disk storage.

The management of this virtual memory serves to illustrate the involvement of the MCP in dynamic resource allocation. This process is diagrammed in Figure 2. Main memory is allocated by the MCP as a resource to current processes. When a program requires additional memory for a segment of code or data, an unused area of sufficient size is sought by the MCP. If it fails to locate a large enough unused area, it looks for an already allocated area which may be overlaid. If necessary, it links together adjacent available and in-use areas in an attempt to create an area large enough for the current demand. When the area is found, the desired segment is read in from disk and the segments currently occupying this area are either relocated elsewhere in core (if space is available), swapped out to disk or simply marked not present. In any case, the appropriate descriptor must be modified to keep track of the address in memory or on disk of all segments involved in the swap. All of these operations are carried out by the MCP; monitoring allows us to understand them better. For additional information on the operation and structure of the B6700 see Reference 3.

B6700 PERFORMANCE STATISTICS

The complexity of the B6700 system provides both the necessity to monitor and the ability to monitor. The pervasive nature of the MCP in controlling the jobs in the system and in allocating system resources made it necessary for the system designers to reserve areas of core memory and specific cells in the program stacks to keep data on system and program status.
Figure 2: B6700 memory allocation procedure.

1. Space is needed for a 300 word segment for one of the current tasks.
2. A large enough unused area is not located.
3. The MCP finds a contiguous location made up of areas 1 through 4 which is 300 words long.
4. Area 1 contains a 50 word data segment. The MCP relocates this segment into area 5, makes note of its new core address and removes area 5 from the unused linkage.
5. Area 2 is unused. It is removed from the unused linkage.
6. Area 3 contains a 100 word code segment. There are no unused areas large enough to contain it. Therefore, it is simply marked not present. Since code cannot be modified during execution, there is no reason to write it out to disk; it is already there.
7. Area 4 contains a 100 word data segment. It is written out to disk and its new location is recorded.
8. The 300 word segment is read into core in the area formerly occupied by areas 1 through 4 and its location is recorded.
This design enables us to access and collect data on the following system parameters:
- system core memory utilization
- I/O unit utilization
- I/O queue lengths
- processor utilization
- multiplexor utilization
- multiplexor queue length
- peripheral controller utilization
- system overlay activity
- program overlay activity
- program core memory utilization
- program processor utilization
- program I/O utilization
- program status
- scheduler queue length
- response time non-trivial requests

These data are vital to the evaluation of our computer system. Table I presents examples of the possible uses for some of these statistics.

DESCRIPTION OF THE SPASM SYSTEM
The B6700 System Performance and Activity Software Monitor, SPASM, is designed to monitor the performance of the system as a whole as well as that of individual user programs. It consists of two separate programs, a monitor and an analyzer, both of which are described below. The principal criteria governing its design are:

a. to make a software monitor capable of gathering all the pertinent data discussed in the previous section;

b. to minimize the additional load placed upon the system by the monitor itself;

c. to provide an easily used means of summarizing and presenting the data gathered by the monitor in a form suitable for evaluation by technical personnel and management.
<table>
<thead>
<tr>
<th>Data</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>System core memory utilization.</td>
<td>Determine need for additional memory.</td>
</tr>
<tr>
<td>I/O unit utilization, I/O unit queue lengths.</td>
<td>Determine need for Disk File Optimizer and/or additional disk storage electronic units, printers or disk file controllers.</td>
</tr>
<tr>
<td>Processor queue length and composition.</td>
<td>Determine need for additional processor.</td>
</tr>
<tr>
<td></td>
<td>Evaluate effect of job priority on execution.</td>
</tr>
<tr>
<td></td>
<td>Determine processor boundedness of mix.</td>
</tr>
<tr>
<td></td>
<td>Determine effect of processor utilization on demand for I/O (in conjunction with I/O unit data).</td>
</tr>
<tr>
<td>System overlay activity.</td>
<td>Determine need for additional memory.</td>
</tr>
<tr>
<td></td>
<td>Determine need for better task scheduling.</td>
</tr>
<tr>
<td></td>
<td>Determine when thrashing* occurs.</td>
</tr>
<tr>
<td>Job overlay activity.</td>
<td>Evaluate program efficiency.</td>
</tr>
<tr>
<td></td>
<td>Evaluate system effect on job execution.</td>
</tr>
<tr>
<td>Job core memory utilization.</td>
<td>Evaluate program efficiency.</td>
</tr>
<tr>
<td></td>
<td>Change job core estimates.</td>
</tr>
<tr>
<td>Scheduler queue length.</td>
<td>Determine excess demand for use of system.</td>
</tr>
<tr>
<td></td>
<td>Evaluate MCP scheduling algorithm.</td>
</tr>
</tbody>
</table>

*Thrashing is the drastic increase in overhead I/O time caused by the frequent and repeated swapping of program code and data segments. It is caused by having insufficient memory to meet the current memory demand.
Ability to gather pertinent data

The Master Control Program concept of the B6700 helps in many ways to simplify the acquisition of the data listed in Table I. Such information as a program's core usage, processor and I/O time, and usage of overlay areas on disk are automatically maintained in that program's stack by the MCP. A relatively simple modification to the MCP permits a count of overlays performed for a program to be maintained in its stack. Data describing the status of programs are maintained by the MCP in arrays. Information on system-wide performance and activity is similarly maintained in reserved cells of the MCP's stack. Pointers to the head of the processor queue, I/O queues and scheduler queue permit the monitor to link through the queues to count entries and determine facts about their nature. Other cells contain data on the system-wide core usage, overlay activity, and the utilization of the I/O multiplexor. An array is used to store the status of all peripheral devices (exclusive of remote terminals) and may be interrogated to determine this information.

All of the above data are gathered by an independently running monitor program. The program, developed with the use of a specially modified version of the Burroughs ALGOL compiler, is able to access all information maintained by the MCP. The program samples this information periodically and stores the sampled data on a disk file for later reduction and analysis.

Minimization of load upon the system

To minimize the additional load on the B6700, the monitor program is relatively simple, and very efficient. A somewhat more sophisticated analyzer program is used to read back the raw data gathered by the monitor and massage it into presentable form. This analysis is generally carried out at a time when its additional load upon the system will be negligible. The system log has indicated that the monitor does indeed present a minimal load requiring about 1/4 of 1 percent processor utilization and 2 1/4 percent utilization of one disk I/O channel.
Figure 3: Graph of Core Usage Versus Time

DCSIML/RESMON/OBJECT 0997
TIME INTERVAL 09:27 TO 17:00 DATE: 12/07/72
KEY                 SCALE
SYSAVL TTTTTT 1 UNIT = 003000 WORDS
MEAN = 19877.3756
SYSSLY UUUUUU 1 UNIT = 003000 WORDS
MEAN = 110911.904810
SYSSVE VVVVVV 1 UNIT = 003000 WORDS
MEAN = 98774.885152

TIME AT RIGHT IS: 10:13; DATE: 12/07/72
TIME INTERVAL 09:27 TO 17:00 DATE: 12/07/72
TIME SCALE: 1 UNIT = 0020 SEC.

Figure 3: Graph of Core Usage Versus Time
Easy means of analysis and presentation

The raw data gathered by the monitor can be used directly in some cases; however, to serve best the purpose for which SPASM was designed (i.e., as a management reporting system) several useful presentations have been engineered. The analyzer program, which may be run interactively or in batch mode will produce any of the following:
- Graphs of data versus time.
- Frequency distribution histograms of data.
- Correlation and regression analyses among data.
- Scanning for peak periods.

The options are selected via an input language consisting of mnemonics.

1. Graphs of data versus time are produced to show the minute-by-minute variations in parameters of interest. The graphs are "drawn" on the line printer using symbols to represent each curve. Any number of parameters may be plotted on one graph, and a key is printed at the end identifying all symbols used in the graph and listing the mean values of each. To aid in tailoring the most desirable presentation, rescaling of the ordinate and time axes is permitted. In addition, the user may select a specific time interval of interest and plot that interval only. Figure 3 presents a graph of core utilization versus time. Lines may be drawn connecting the points to aid readability. On this graph three parameters are simultaneously plotted, namely "SAVE" core (SYSSVE), "OVERLAYABLE" core (SYSOLY) and "AVAILABLE" core (SYSAVL). The key lists mnemonics, symbols plotted and the mean values of the plotted parameters.

To prepare such a graph for presentation, the analyst, using a remote terminal, first causes the analyzer to "plot" the information on the CRT. The data is then scaled to the analyst's preference, perhaps adding or deleting specific data curves to improve readability. The analyst then re-"plots" the graph on the CRT. When he is satisfied he directs this tailored graph to the line printer via an input command for hard copy.
HISTOGRAM OF SYSOLY

VALUES

0

15000

00000

FREQUENCY OBSERVED

FIGURE-3 HISTOGRAM OF OVERLAYABLE CORE USAGE
2. Frequency distribution histograms are useful in that they present a concise picture of the behavior of specific parameters. The analyzer will produce, instead of, or in addition to time graphs, a histogram of each desired parameter. The histogram is automatically scaled to the mode of the distribution, and is printed horizontally. The axis which represents the observed values may be rescaled as above to improve the clarity of the data presentation. These histograms show the distribution of observed values of the parameter in question. Figure 4 presents a histogram of system "OVERLAYABLE" core usage. It covers a longer period of time than the time graph and shows the frequency distribution of this parameter.

3. Correlation and regression analyses among data play an important role in aiding in the forecasting of future needs. If the relationships among several parameters are known, then, should a change be projected in one parameter, its effect on the others can be approximated, and suitable adjustments in configuration or scheduling can be made. For example, if one determines a relationship between the amount of core memory and the overlay rate, one can predict the degree of decrease in the overlay rate should more core be added. The analyzer will allow correlations and/or regressions to be run among any of the parameters. Tables of pertinent statistics are printed as a result.

Figure 5 presents the result of a correlation and regression analysis among core utilization, processor utilization and overlay rate parameters. These results are seen to show that, for example, the overlay statistics are highly correlated to the amount of "SAVE" core in the system. This is understandable since the larger the "SAVE" core the greater the chance of needing to swap segments.
Figure 5: Results of regression and correlation analysis.
4. Scanning for peak periods is a necessity in most computer systems, especially those operated in a data communication environment where the load fluctuates widely. The analyzer can scan the entire day's data and flag time intervals (of length greater than or equal to some specified minimum) during which the mean value of a parameter exceeded a desired threshold. For example, a period of five minutes or longer in which processor utilization exceeded 75 percent can be easily isolated (see Figure 6). Using this technique a peak period can be automatically determined and then further analyzed in more detail.

```
FOR 0740 SECONDS ENDING 09:50 PROCUT EXCEEDED THRESHOLD
FOR 2160 SECONDS ENDING 11:06 PROCUT EXCEEDED THRESHOLD
FOR 0700 SECONDS ENDING 11:49 PROCUT EXCEEDED THRESHOLD
FOR 1320 SECONDS ENDING 12:31 PROCUT EXCEEDED THRESHOLD
FOR 0480 SECONDS ENDING 12:39 PROCUT EXCEEDED THRESHOLD
FOR 2220 SECONDS ENDING 13:37 PROCUT EXCEEDED THRESHOLD
FOR 4920 SECONDS ENDING 15:03 PROCUT EXCEEDED THRESHOLD
FOR 0420 SECONDS ENDING 15:12 PROCUT EXCEEDED THRESHOLD
```

Figure 6: Periods of peak processor utilization.

The design criteria discussed above have been met and a software monitoring system has been developed which is comprehensive, and easily used, and yet presents a negligible load upon the B6700 computer.

CONCLUSION AND OUTLOOK

The SPASM system has proven to be very instrumental in the performance evaluation of the B6700 system at the Bank. Several areas in which it has been and is currently being used are as follows:
- The statistics on processor queue length, multiplexor utilization, and disk controller utilization were used to aid in the analysis of the need for a second processor*, second multiplexor and additional controllers.

*See Appendix A for a discussion of how the processor queue data was used to determine processor utilization.
- The job core utilization data have been used to evaluate the effect of alternate programming techniques on memory use.
- Disk utilization data have been examined to identify any apparent imbalance of disk accesses among the disk electronics units.
- Processor queue data are being used to determine the effect of task priority on access to the processor.
- System overlay data are being used to determine the adequacy of automatic and manual job selection and scheduling.
- Processor utilization figures, as determined from the processor queue data, were used to determine the effect of core memory expansion on processor utilization.

Some future possible uses planned for SPASM include:
- Use of the scheduler queue statistics to evaluate the efficiency of the current MCP scheduling algorithm and to evaluate the effect changes to that algorithm have on the system performance.
- Use of the response time data to evaluate system efficiency throughout the day with different program mixes.
- Evaluation of resource needs of user programs.
- Evaluation of the effect that the Burroughs Data Management System has on system efficiency.
- Building of a B6700 simulation model using the collected statistics as input.
- Building an empirical model of the B6700 system by using the collected regression data.

The SPASM system has enabled us to collect a great deal of data on system efficiency and, consequently, a great deal of knowledge on how well the system performs its functions. This knowledge is currently being used to identify system problems and to aid in evaluating our current configuration and possible future configurations. Mere conjecture on system problems or system configurations in the absence of supporting data is not the basis for a logical decision on how to increase system efficiency. Performance measurement and evaluation are essential to efficient use of the system.
REFERENCES

APPENDIX A
The use of processor queue data to determine processor utilization.

The SPASM system records the length of the processor queue periodically. The processor utilization will be based upon these examinations, taking into account that the monitor itself is processing at this instant of time. If the processor queue is not empty, the monitor is preventing some other job from processing. Consequently, if the monitor were not in the system the processor would be busy with some other task at that instant of time. This is considered to be a processor "busy" sample. On the other hand, if the processor queue is empty at the sample time there is no demand for the processor other than the monitoring program itself. Therefore, if the monitor were not in the system at that instant of time the processor would be idle. This is considered a processor "idle" sample. Processor utilization can therefore be estimated as:

\[
\text{processor utilization} = \frac{\text{No. "busy" samples}}{\text{total No. samples}}
\]

This sampling approach to determining processor utilization was validated by executing controlled mixes of programs and then comparing the results of the sampling calculation of processor utilization to the job processor utilization given by:

\[
\text{processor utilization} = \frac{\text{processor time logged against jobs in the mix}}{\text{elapsed time of test interval}}
\]
Table II compares the results of these two calculations.

**TABLE II: Comparison of Processor Utilization Statistics By Sampling Technique and By Processor Time Quotient Technique.**

<table>
<thead>
<tr>
<th>Average Processor Utilization (%)</th>
<th>Sampling Technique</th>
<th>Processor Time Quotient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Series 1</td>
<td>99.1</td>
<td>96.5</td>
</tr>
<tr>
<td>Test Series 2</td>
<td>57.6</td>
<td>53.5</td>
</tr>
</tbody>
</table>

In a second type of test, processor idle time was monitored (by means of a set of timing statements around the idling procedure) to gain a close measure of utilization. The total idle time was subtracted from the total elapsed time of the test to obtain the processor busy time and hence the utilization. Over a period of five hours the respective processor utilization calculations were:

- Sampling Technique: 46.3%
- Idle Timing: 48.0%

These results make us confident of the validity of using the processor queue sampling technique to accumulate processor utilization statistics during any given time interval.
NEW OPTION FOR SYSTEM/DUMPANALYZER
by K.J.E. Lewis
(Technical Support Group, Computer Services Department, Midland Bank Ltd.)

1. INTRODUCTION
The Midland Bank application programs are generally based on one FATHER stack firing up a series of SONS. These SONS may in turn fire up their own SONS. When an error interrupt occurs for one of the processes it is desirable to take an instant snapshot of the stack in error plus all its ancestors and their segment dictionaries.

At first it was thought that PROGRAMDUMP should be modified but this idea was rejected for the following reasons:
1. We need to halt all processors while doing this so that the FATHER stack is not altered by other SONS while we are dumping.
   The time then needed to unravel stacks while holding all the system is greater than that needed for a memory dump.
2. Vital areas could be overlayed by the program dump code.
3. A full memory dump can be re-analysed later if more information is required.
4. The DCALGOL construct MEMORYDUMP will allow programatic use of "TAPEDUMP" as well as the operators being able to force it.

2. PATCHES TO SYSTEM/DUMPANALYZER
2.1. A new option "FAMILY" was implemented which unravelled the FATHER stack(s) of a given MIX number together with all segment dictionary stacks and areas, and suppressed the listing of irrelevant information.
2.2. The MIX option was extended to output segment dictionary stacks.

3. SYNTAX OF FAMILY OPTION
   < FAMILY OPTION > = FAMILY < MIX LIST >
   < MIX LIST > = < MIX NUMBER >/< MIX LIST > < MIX NUMBER >
4. **SEMANTICS**

The FAMILY option enables the user to specify a particular MIX number or series of MIX numbers. These numbers are stored in the array DISASTERAREAS as usual but when the check for specific stack dumps is made:

1. The appropriate segment dictionary stack is requested.
2. PROCESSFAMILYLINK is accessed and if FATHER NEQ 0 then that stack and its segment dictionary stack is requested.
3. Process 2 is repeated until such time as FATHER=0.

If both FAMILY and MIX options are utilized in the OPTIONS deck then the last in overwrites the first.

The FAMILY option can be used for a non-dependant stack when a short analysis is required.

A copy of the patches to SYSTEM/DUMPANALYZER is included.
PATCH NUMBER 7

DUMPA 0006B KJEL:PRINTS SEG.DIC.STACKS & IMPLEMENTS FAMILY OPTION
6" FAMILY = ", L5

04085200

%% SDSTACK=M[X+].SEGDICTF
06044500

FATHERSTACK=M[X+12].[35:12],
06045500

FAMDUMP, 07071200

IF P2=8"MIX" OR P2=8"FAMILY" THEN
Z DUMP SPECIFIED STACKS 60040000
FAMDUMP:=P2=8"F";
60041500

IF FAMDUMP THEN GO HUH;

BEGIN
60048600

X:=M[ARRAYSTACK+SNR].ADDRESSF; % DUMP SEG.DICT STACK 60126345
SETREQUESTED(SDSTACK);
60126430

IF FAMDUMP THEN WHILE X:=FATHERSTACK NEQ 0 DO
60126450
BEGIN
Z CLimb THE FAMILY TREE 60126460
SETREQUESTED(X);
60126470

X:=M[ARRAYSTACK+X].ADDRESSF;
60126480

SETREQUESTED(SDSTACK);
60126490

END END;
60126495

IF NOT FAMDUMP THEN
60126550

IF NOT (SYSTEMLOCKS IS 0 OR FAMDUMP) THEN
60185100

IF MCPNAMESAVAIL AND NOT FAMDUMP THEN

IF NOT FAMDUMP THEN
60405000

IF LINKDUMP AND NOT FAMDUMP THEN BEGIN
60432650

IF NOT FAMDUMP THEN
60435500

BEGIN
60452250

END;
60452260

IF NOT FAMDUMP THEN
60452450

BEGIN
60452750

END;
60452760

IF NOT FAMDUMP THEN
60453050

BEGIN
60453250

END;
60453260

END;
60453450
Hardware for digital plotting on the B6700.

0. Introduction.
This article contains a brief description of the available plotter hardware at the Computing Centre of the Eindhoven University of Technology in the Netherlands.
The article "A drawing system based on a formal picture concept" written by Rens Kessener deals with the software.

1. Equipment.
At the Computing Centre are installed:
- On-line: a) 2 Calcomp drum plotters, type 565,
  characteristics: 11 inch paperwidth
  speed: 300 steps/sec.
  stepwidth: 0.1 mm.
  Each of the plotters is separately connected via a controller to a standard line-adapter of the DCP.
The controller is described in chapter 2.
b) 1 Graphical display, Tektronix T4002A, connected via a 9600 baud asynchronous line to the DCP.
c) Line plotters (= printer),
  characteristics: paper 11 x 14.5 inch
  stepwidth: 1/6 or 1/8 inch.
- Off-line: 1 Calcomp drum plotter, type 1136,
  characteristics: 30 inch paperwidth
  speed: 2600 steps/sec.
  stepwidth: 0.05 mm.
  three color plots,
  connected to a tape unit, type 915,
  9 track, NRZI, 800 bpi
  7 track, NRZI, 800, 556, 200 bpi.
2. The controller for the Calcomp plotter, type 565.

2.0. Introduction.

The controller has been developed at the University of Helsinki in Finland as a part of a Master's diploma work based on the ideas of dr. Pentti Paalero and conducted by prof. Antti Siivola of the Physics Department.

We made a few alterations but these didn't affect the basic design principles.

The controller has a standard CCITT-V24/EIA RS-232 interface, and resembles a 2400 baud synchronous modem. Therefore it can be connected to every computersystem with datacommunication facilities.

The design is kept as simple as possible.

To avoid transmission of operator messages and other nonsense to the plotter, some security measurements have been taken software wise. Also the position of the pen regarding to the paperwidth is controlled in software.

The DCP transmits standard ASCII-characters (bit sequence from the least significant bit (b1) to the most significant bit (b7) in ascending order, and parity bit) with odd parity.

Two SYN-characters precede one block of 480 characters (= plot-instructions).

The plotter accepts 6 plotcommands.

1 pen up
2 pen down
3 drum up
4 drum down
5 carriage left
6 carriage right

figure 1
There are 10 plotinstructions: - 8 directions
   - pen up
   - pen down.
The plotcommands (3,5), (3,6), (4,5) and (4,6) can be performed simultaneously.
The plotter speed is 300 steps per second.
For each plotinstruction one ASCII-character is transmitted by the DCP, whereas the transmission speed is 2400 bits per second. The plotter doesn't generate any ready signal after completing a plotinstruction. However, the instructions PEN UP and PEN DOWN are more time consuming than the other ones. Therefore the necessary delay is programmed:

IF PENUP OR PENDOWN THEN DELAY (100 MILLIS);

The 10 plotinstructions and the corresponding ASCII-characters and bitpatterns are given in the next table.

<table>
<thead>
<tr>
<th>plotinstructions</th>
<th>ASCII CHAR.</th>
<th>bitpattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>- pen up</td>
<td>A</td>
<td>b8 b7 b6 b5 b4 b3 b2 b1</td>
</tr>
<tr>
<td>- drum down</td>
<td>C</td>
<td>0 0 1 0 0 0 0 1 1</td>
</tr>
<tr>
<td>- drum down and carriage right</td>
<td>E</td>
<td>0 0 1 0 0 0 1 0 1</td>
</tr>
<tr>
<td>- carriage right</td>
<td>G</td>
<td>1 0 1 0 0 1 1 1 1</td>
</tr>
<tr>
<td>- drum up and carriage right</td>
<td>I</td>
<td>0 0 1 0 1 0 0 0 1</td>
</tr>
<tr>
<td>- drum up</td>
<td>K</td>
<td>1 0 1 0 1 0 1 1 1</td>
</tr>
<tr>
<td>- drum up and carriage left</td>
<td>M</td>
<td>1 0 1 0 1 1 0 1 1</td>
</tr>
<tr>
<td>- carriage left</td>
<td>O</td>
<td>0 0 1 0 1 1 1 1 1</td>
</tr>
<tr>
<td>- drum down and carriage left</td>
<td>Q</td>
<td>0 0 1 1 0 0 0 0 1</td>
</tr>
<tr>
<td>- pen down</td>
<td>S</td>
<td>1 0 1 1 0 0 1 1 1</td>
</tr>
<tr>
<td>- synchronization</td>
<td>SYN</td>
<td>0 0 0 1 0 1 1 0 0</td>
</tr>
</tbody>
</table>
Note: For all plot instructions stands: bit b1 = "1"
     bit b6 = "1"
     bit b7 = "0".

- The bits b5, b4, b3 and b2 are the binary representation of the decimal numbers 0 to 9.
- Using odd parity the SYN-character doesn't have these characteristics (one bit left shifted and even parity the SYN-character equals the character M).

2.1. Description of the controller.

![Diagram](image-url)

figure 2
In the diagram of the controller drawn in figure 2 one can distinguish 7 blocks.

1. Power supply (as well as for the controller as for the plotter).
2. CCITT-V24/EIA RS-232 interface.
3. Clock generator.
5. Synchronization.
6. Detector.
7. Decoder.

The synchronization detects the SYN-character and as a result the serial-parallel converter will be reset.

The detector generates a pulse to the decoder when \( b_1 = "1" \), \( b_6 = "1" \) and \( b_7 = "0" \) and resets (28 µsec. later) the serial-parallel converter.

The decoder translates the 10 plot instructions into pulses on the 6 lines of the plot commands.

In figure 3 are drawn the detailed diagrams of the blocks 1, 2 and 3, and in figure 4 the diagrams of the blocks 4, 5, 6 and 7.

2.1.1. Power supply.

The Dutch public utility service provides 220 Volt at 50 Hz. The plotter requires 115 Volt (1.5 amperes) and the stabilized regulated rectifier G, 32 Volt with central branch. Therefore a special transformer T has been built at the University.

The stabilized regulated rectifier provides +12 and -12 Volt DC.

The voltage regulator II provides +5 Volt DC.

The switch S1 is the on/off power switch; L1 is an indicator light for power-on. The original power switch of the plotter has been disabled.
fig. 3
2.1.2. CCITT-V24/EIA RS-232 interface.

The following standard signals are used:

- Data in
  - pin nr. 3
  - circuit nr. 104
  - Burroughs BB
- Request to send
  - pin nr. 4
  - circuit nr. 105
  - CA
- Ready for sending
  - pin nr. 5
  - circuit nr. 106
  - CB
- Data set ready
  - pin nr. 6
  - circuit nr. 107
  - CC
- Connect data set to line
  - pin nr. 20
  - circuit nr. 108/1
  - CD
- Receiver signal
  - pin nr. 17
  - circuit nr. 115
  - DD
- Element timing
- Signal ground
  - pin nr. 7
  - circuit nr. 102
  - AB

The signals "request to send" and "ready for sending" are directly connected without delay, to detect whether the interface is present or not. Through the switch S2 the operator has control over the signal "data set ready" so he has the ability to block the transmission of data by the DCP. The signal from S2 in combination with "connect data set to line" is driven, via nand gates I1 (a, b and c), a control light L2, "data set ready" and the serial-parallel converter.

Via I4a the incoming data are converted to TTL-level. The clock-signal is converted in I3b to CCITT-V24/EIA RS-232 level.

2.1.3. Clock generator.

Two one shots I5a and I5b generate the clocksignal whereas I5a via C5 and R6 (potentiometer) stands for the pulse frequency and I5b via C6 and R7 for the pulsewidth.

2.1.4. Serial-parallel converter.

The serial-parallel converter is an 8-bits shiftregister (I9). Serial data are entering on point 1. Point 2 is set and enables input via point 1 when "data set ready" is on (see 2.1.2.). A low level on point 9 resets the shiftregister. The clock is entering on point 8.
2.1.5. Synchronization.

Via 5 inverters (I4 b, c, d, e and f) the SYN-character (00010110) is detected by I10. By I10 the shiftregister will be reset via the nand gates I17a and I17b so the right 8 bit groups will be decoded.

2.1.6. Detector.

In fact the detector is an other kind of synchronization. The nand gates I11 a, b, c and d will drive a one shot I12a when for the bits b7, b6 and b1 of the received character stands 011. The one-shot generates a pulse (28 μsec.) for the decoder and on the negative edge of the pulse another one-shot, I12b, resets via I17a and I17b the shiftregister.

2.1.7. Decoder.

The bits b5, b4, b3 and b2 of the received character are lead to the inputs of a "4-line-to-10-line-decoder", I13. Via the nand gates, I14 and I15, the 10 output lines are connected to the 6 lines of the plotsignals. A high level on one of the output lines is leading the pulse from I12a to the buffer/drivers, I18. The 6 buffer/drivers (I18) will give the plotsignals enough power to drive the plotterlogic.

2.2. Component list.

Integrated circuits:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>L005</td>
<td>voltage regulator.</td>
</tr>
<tr>
<td>I2, I11, I16, I17</td>
<td>SN7400</td>
<td>quadruple 2-input positive nand gates.</td>
</tr>
<tr>
<td>I3</td>
<td>μA9616</td>
<td>triple EIA RS-232-C line driver.</td>
</tr>
<tr>
<td>I4</td>
<td>SN7404</td>
<td>hex inverters.</td>
</tr>
<tr>
<td>I5, I12</td>
<td>TEH9602</td>
<td>dual monostable multivibrator.</td>
</tr>
<tr>
<td>I6, I7</td>
<td>SN7473</td>
<td>dual J-K master-slave flip-flops.</td>
</tr>
<tr>
<td>I8, I14, I15</td>
<td>SN7410</td>
<td>triple 3-input positive nand gates.</td>
</tr>
<tr>
<td>I9</td>
<td>SN74164</td>
<td>8 bit parallel-out serial shiftregister.</td>
</tr>
<tr>
<td>I10</td>
<td>SN7430</td>
<td>8-input positive nand gate.</td>
</tr>
<tr>
<td>I13</td>
<td>SN7442</td>
<td>4-line-to-10-line decoder.</td>
</tr>
<tr>
<td>I18</td>
<td>SN7416</td>
<td>hex inverter buffer/drivers.</td>
</tr>
</tbody>
</table>
Resistors:

R1 820Ω 1/8w
R2, R4 2k2Ω 1/4w
R3, R5 4k7Ω 1/4w
R7, R8, R9 18kΩ 1/8w
R10, R11, R12, R13, R14, R15 1k2 1/4w
R6 a combination of: R6a 13kΩ 1/8w (1%)
R6b potentiometer 2kΩ

Condensors:

C1 1000µF 16V
C2 100µF 16V
C3 3.3nF
C4 0.1µF
C5 1200nF
C6 330nF
C7 4.7nF
C8 1.8nF

Others:

L1, L2 indicator lights 12V, 50mA
S1 switch
S2 switch
T1 transistor BC107
D1, D2, D3, D4 diodes IN914
Z fuse 2.5A (slow)
V transformer (designed at the University)
G regulated power supply, C15-1 D (Dutch product)
2.3. **Operational aspects.**

With a timer/counter on pin 8 of the shiftregister 19, the delay between 2 clockpulses is set via R6 to 417 μsec.

The on/off power switch, switches as well as the controller the plotter.

With the blocked/ready switch the operator can block the transmission by the DCP.

The switch may not be set on the blocked position while drawing a picture. In that case the transmitted block will be retransmitted and the reference point of the picture is lost.

---

R. van Eyndhoven,
Eindhoven University of Technology
Computing Centre.

---

think small
big ideas upset people
A drawing system based on a formal picture concept.

by Rens Kessener, Herman Willemsen
(Eindhoven University of Technology, Computing Centre).

Summary.
A definition of the concept "PICTURE" is used to implement a drawing system on a Burroughs B6700 of the Eindhoven University of Technology. This concept contains a set of elements which may be PICTURES as well. A number of procedures has been constructed based on the definition of the PICTURE concept.
This procedure package covers many application areas and is independent of the drawing device.
The constructed drawing may be mapped on any device suitable to draw (e.g. a storage tube, a plotter, a printer).

Introduction.
A drawing (or a PHYSICAL PICTURE) is considered to exist of the definition of a PICTURE. This PICTURE is mapped on a device that is suitable to draw. Therefore a concept "PICTURE" is necessary and a number of procedures building the PICTURE in order to draw it (i.e. a procedure package).
The aims which must be fulfilled by the concept are:
- the concept must be structured;
- the concept must be easy to manipulate and easy to understand.
For the procedure package it is required that:
- the package covers all disciplines that appear at a university (e.g. the Eindhoven University of Technology);
- the package is independent of the device;
- the use of the package involves operator interference as little as possible;
- it is possible to generate several drawings simultaneously i.e. within a program as well as in a multiprogramming environment.
To reach the aim of simultaneous generation of drawings, it is necessary to create a sort of backup mechanism. In addition, to be device independent the information must be such that it can be interpreted by a drawing machine for a specific device.

In the succeeding sections we will explain the PICTURE concept. We will discuss some differences between the concept and the actual implementation. Finally we will discuss the procedure package and the software drawing machine.

The PICTURE concept.
In order to formulate the PICTURE concept we draw distinction between PICTURES and IMAGES.

An IMAGE consists of a set of elements. Each element is determined by its type and its position. The position is expressed in units of $R_2$ (two dimensional) or $R_3$ (three dimensional). By mapping an IMAGE into another IMAGE the former becomes a PICTURE.

We will explain this map in more detail later.
More formally we may define:

< image > ::= < open bracket > < set of elements > < close bracket >
< set of elements > ::= < empty >|< element > < set of elements >

We already saw that an IMAGE can be mapped into another IMAGE and then we call it a PICTURE.
So a PICTURE can be an element of an IMAGE (its type is IMAGE, its position is indicated by the map).
We distinguish only one other element type: PRIMITIVE.
A PRIMITIVE may have the following kind:

- dot
- point
- straight line piece
- curved line piece
- text
- axis

Now let us explain some terms more explicitly:

MAP.
A map consists of:
1. scaling
   i.e. \( x \rightarrow y \) such that \( y = \lambda x \) (\( \lambda \) = scaling factor)
2. translation
   i.e. \( x \rightarrow y \) such that \( y = t + x \) (\( t \) is translation vector)
3. rotation
   i.e. \( x \rightarrow y \) such that \( y = Rx \) (\( R \) is rotation matrix).

The map is performed by means of a matrix multiplication. The way this matrix is constructed is very dependent of the implementors point of view.
(See [3], [4])

WINDOW.
A window is defined by specifying of a rectangle in the current space.
All elements, or parts of elements that appear outside that window will never be visualized.
PRIMITIVES.

**Dot**
A dot is indicated by its coordinates.
Its representation is a simple dot.

**Point**
A point is indicated by its coordinates.
Its representation is a special character (*, + etc.).

**Straight line piece**
A straight line piece is indicated by the coordinates of its extreme points.
Its representation may be fully drawn, dotted, dashed or dot-dashed.

**Curved line piece**
A curved line piece is indicated by the coordinates of its extreme points and the derivatives in these points. This holds for 2D. For 3D it probably must be indicated by a parameter form.
For 2D a part of a third degree polynomial is drawn through the indicated points in which the derivatives agree with the indicated values.
Its representation may be fully drawn, dotted, dashed or dot-dashed.

**Text**
A text is indicated by its extreme points and its content.
Its representation may be italicized.

**Axis**
An axis is indicated by its extreme points.
Optionally numbering and comment may be specified.

Remarks:

**Text**
The text was our main problem and actually it still is. Because by mapping an IMAGE into another IMAGE the ultimate visible text may be considerably deformed.
As we consider a text to be comment which should be readable we defined a character to be represented in a square when it is drawn in the real world.

**Axis**
For axis a comparable remark as for text holds.
We consider an axis to be a piece of comment which should be readable in its final representation.

**Curved line piece**
The choice of a third degree polynomial approximation is arbitrary.
Any other approximation should suit as well.
FINAL FORMAL DEFINITION.

< image > ::= < open bracket > < set of elements > < close bracket >
< set of elements > ::= < empty > | < element > < set of elements >
< element > ::= < primitive > | < picture >
< picture > ::= < map > < window > < image >
< window > ::= < empty > < row of coordinates >
< map > ::= < translation > < scaling > < rotation >
< primitive > ::= < dot > < point > < straight line piece > |
< curved line piece > < text > < axis >
< dot > ::= < row of coordinates >
< point > ::= < row of coordinates > < point type >
< straight line piece > ::= < row of coordinates > < row of coordinates >
< representation >
< curved line piece > ::= < row of coordinates > < slope value >
< row of coordinates > < slope value > < representation >
< representation > ::= < fully drawn > < dotted > < dashed > < dot-dashed >
< text > ::= < row of coordinates > < row of coordinates >
< italicty > < string >
< axis > ::= < row of coordinates > < row of coordinates >
< number of intervals > < row of coordinates > < row of coordinates > < number of intervals >
< numbering information > |
< row of coordinates > < row of coordinates > < number of intervals > < numbering information >
< string >

The final appearance of the items, that are not elaborated, is implementor dependent.
MAPPING THE IMAGE ONTO A VISIBLE MEDIUM.

So far the IMAGE is still a virtual thing. By mapping the IMAGE onto the real world (say a piece of paper or a screen) a final map must be performed and the IMAGE can be made visible, i.e. the PHYSICAL PICTURE.

As the real world, onto which the IMAGE is mapped, is limited, this final map has to be slightly different than the MAP described before. That is why we call this final map the WINDOWMAP.

WINDOWMAP

A windowmap consists of:
1. a declaration of an area of the real world that will be visible;
2. a map of the image onto the real world.

ad 1. This declaration may be considered as the piece of paper on which the IMAGE will be visualized. This holds for two dimensions.

For three dimensions there must be a definition of the point of view and the plane on which the IMAGE will be projected as well.

ad 2. The map must describe the relation between the points in the real world and the points of the IMAGE.

The windowmap is performed by indicating a rectangle in the "visible" world and indicating a rectangle in the images world. The content of the rectangle in the IMAGE will be mapped onto the rectangle in the real world.

It is obvious that for 3D a block in the images world must be defined, a viewing point and/or a plane onto which the block content will be mapped.

Conclusion

We didn't mention any implementation yet, nor any data structure in which the image is represented. This is very dependent of the application area. For example the data structure and procedure package for Graphical Displays will be considerably different from those for plotters.

But it must be quite easy to develop a drawing machine that is able to interpret any data structure for any device.
Implementation.

Until now we implemented this concept for plotters, based on a sequential data structure and only for two dimensions.

The implementation consists of two parts:

1. a package of procedures by which a user may generate his FYSICAL PICTURE;
2. a drawing machine which interprets the user-generated FYSICAL PICTURE and plots it on a device.

The FYSICAL PICTURE must be generated from the outer level down to the inner level of IMAGES.

This means the first item of the sequential data structure must be the outer (window) map. The last item must be a close bracket of the outer IMAGE. In between the items must be elements.

The unit that is presented to the drawing machine must be a data file which contains a FYSICAL PICTURE. This means that for such a data file the device kind must be determined already.

Also it is possible to generate an IMAGE. But this cannot be presented to the drawing machine.

It only may appear as an element of another IMAGE.

Devices for which the implementation is suitable now:

CALCOMP C565 drumplotter, 11 inch, 300 steps/sec., 0.1 mm/step.
CALCOMP C1136 drumplotter, 30 inch, 2600 steps/sec., 0.05 mm/step.
Lineprinter, 30 cm., 5 cm/step.
The backup mechanism.
Because there is no need to retrieve picture elements in a plotting system we decided to use a simple sequential data structure with fixed record length.
Each record contains a primitive.
Primitives containing a textstring may cover several records.
Assignment to a drawing device is performed by assigning a specific title to the data file.
For example assignment to C565 is performed by a title which is prefixed by "PLOT11", to C1136 by a title prefixed by "PLOT30".
Because it is not possible to assign windowmap information to data file attributes, the first record must contain that windowmap.
When the file has been generated and locked, it will be drawn. The drawing machine will look for data files which obviously contain a physical picture (identifiable by its prefix) and subsequently the drawing machine will interpret the data file and map it onto the drawing device.

User procedures.
All user procedures are written in BEATHE (Burroughs Extended Algol for EINDHOVEN UNIVERSITY OF TECHNOLOGY).
All procedure headings are described as well as a short description of the function performed by it.
The user must declare one or more files in which he will generate the picture(s).
The picture file is one of the parameters of most of the procedures, so the user is able to generate simultaneously as much pictures as he wants to.
Only the most frequently used and/or necessary procedures are described.

Map and brackets.
PROCEDURE WINDOWMAP(FILE, WINDOWXORIGIN, WINDOWYORIGIN, WINDOWNIGIROX, WINDOWNIGIROY, XORIGIN, YORIGIN, NIGIROX, NIGIROY);
VALUE WINDOWXORIGIN, WINDOWYORIGIN, WINDOWNIGIROX, WINDOWNIGIROY, XORIGIN, YORIGIN, NIGIROX, NIGIROY;
REAL WINDOWXORIGIN, WINDOWYORIGIN, WINDOWNIGIROX, WINDOWNIGIROY, XORIGIN, YORIGIN, NIGIROX, NIGIROY;
FILE FILE;
As already mentioned the windowmap is performed by indicating a rectangle in the real world and a rectangle in the images world. The latter will be mapped onto the former. The indication is performed by two vertexes of the rectangle diagonally to each other.

The matrix of the map is calculated such that:
- \((X_{\text{ORIGIN}}, Y_{\text{ORIGIN}})\) of the image is mapped to \((\text{WINDOW}_{X_{\text{ORIGIN}}}, \text{WINDOW}_{Y_{\text{ORIGIN}}})\);
- \((X_{\text{ORIGIN}}, Y_{\text{ORIGIN}})\) of the image is mapped to \((\text{WINDOW}_{X_{\text{ORIGIN}}}, \text{WINDOW}_{Y_{\text{ORIGIN}}})\);
- \((X_{\text{ORIGIN}}, Y_{\text{ORIGIN}})\) of the image is mapped to \((\text{WINDOW}_{X_{\text{ORIGIN}}}, \text{WINDOW}_{Y_{\text{ORIGIN}}})\).

'PROCEDURE' MAPIMAGE(FILE, X1ORIGINAL, Y1ORIGINAL, X2ORIGINAL, Y2ORIGINAL, X1IMAGE, Y1IMAGE, X2IMAGE, Y2IMAGE, XROTATION, YROTATION, ALPHA);

'VALUE' X1ORIGINAL, Y1ORIGINAL, X2ORIGINAL, Y2ORIGINAL, X1IMAGE, Y1IMAGE, X2IMAGE, Y2IMAGE, XROTATION, YROTATION, ALPHA;

'REAL' X1ORIGINAL, Y1ORIGINAL, X2ORIGINAL, Y2ORIGINAL, X1IMAGE, Y1IMAGE, X2IMAGE, Y2IMAGE, XROTATION, YROTATION, ALPHA;

'FILE' FILE;

As we think the most convenient way to imagine a map is thinking about two rectangles which at least must be mapped onto each other and potentially rotated after that.

The image that is to be mapped we call the original. The image into which the original is mapped, is indicated by image. \((X1_{\text{ORIGINAL}}, Y1_{\text{ORIGINAL}})\) and \((X2_{\text{ORIGINAL}}, Y2_{\text{ORIGINAL}})\) denote two vertexes of the rectangle in the original, diagonally to each other. \((X1_{\text{IMAGE}}, Y1_{\text{IMAGE}})\) and \((X2_{\text{IMAGE}}, Y2_{\text{IMAGE}})\) denote in the same way a rectangle in the image into which \((X1_{\text{ORIGINAL}}, Y1_{\text{ORIGINAL}})\), \((X2_{\text{ORIGINAL}}, Y2_{\text{ORIGINAL}})\) will be mapped.

To be able to calculate uniquely the mapping matrix it is assumed that \((X1_{\text{ORIGINAL}}, Y2_{\text{ORIGINAL}})\) is mapped to \((X1_{\text{IMAGE}}, Y2_{\text{IMAGE}})\).

After that the image of the original is rotated around a certain point \((X\text{ROTATION}, Y\text{ROTATION})\) in the current image over ALPHA degrees.
'PROCEDURE' CLOSEBRACKET(FILE);
'FILE' FILE;
This procedure writes the close bracket of the last opened image into the file.

'PROCEDURE' WINDOW(FILE, WINDOWXMIN, WINDOWYMIN, WINDOWXMAX, WINDOWYMAX);
'VALUE' WINDOWXMIN, WINDOWYMIN, WINDOWXMAX, WINDOWYMAX;
'REAL' WINDOWXMIN, WINDOWYMIN, WINDOWXMAX, WINDOWYMAX;
'FILE' FILE;
This procedure defines a window in the current space. If it appears, it must appear immediately after a call of MAPIMAGE.
This procedure only is handy to insert an already existing image, stored in the file titled by IMAGESFILETITLE into an image. The file must contain a complete image, the close bracket inclusive. The variables that contain "original" denote a rectangle in the image to be mapped. The variables that contain "image" or "rotation" denote points in the image into which is mapped. See also the procedure MAPIMAGE.

Primitives.

'PROCEDURE' DOT(FILE, X, Y);
'VALUE' X, Y;
'FILE' FILE;
'REAL' X, Y;
The point (X, Y) is dotted.

'PROCEDURE' POINT(FILE, X, Y, POINTTYPE, POINTHEIGHT);
'VALUE' X, Y, POINTTYPE, POINTHEIGHT;
'FILE' FILE;
'REAL' X, Y, POINTHEIGHT;
'INTEGER' POINTTYPE;
The point (X, Y) is marked with a special symbol. There are 16 different marking symbols, numbered from 0 up to 15 inclusive. The value of POINTTYPE determines which symbol is placed on (X, Y) on the output device. POINTHEIGHT indicates the height of the special symbol expressed in the current data units in Y direction.
'PROCEDURE' TEXT(FILE, XFROM, YFROM, XTO, YTO, ITALICITY, STRING);
'VALUE' XFROM, YFROM, XTO, YTO, ITALICITY;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO, ITALICITY;
'STRING' STRING;

This procedure defines a text as an image element. The position of the text is fixed by its starting-point and end-point, that is: the lower left point of the first character and the lower right point of the last character of the text. Each character is written in a parallelogram ABCD. The base AB lies in the direction of writing, the height AE of the character is (on paper) equal to the length of the base of one character. The italicty is given by ITALICITY, the angle α, in degrees, measured clockwise, between the normal on the base (AE) and AD (see figure 3).

Figure 3. This figure shows how a character will be drawn.
'PROCEDURE' STRAIGHTLINEPIECE(FILE, XFROM, YFROM, XTO, YTO, LINEMODE);
'VALUE' XFROM, YFROM, XTO, YTO, LINEMODE;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO;
'INTEGER' LINEMODE;

This procedure defines a straight line piece between (XFROM, YFROM) and (XTO, YTO). The representation of the straight line piece is given by the value of LINEMODE. Possible representations are: traced, dashed, dotted and dashed-dotted.

'PROCEDURE' CURVEDLINEPIECE(FILE, XFROM, YFROM, SLOPEFROM, XTO, YTO, SLOPETO, LINEMODE);
'VALUE' XFROM, YFROM, SLOPEFROM, XTO, YTO, SLOPETO; LINEMODE;
'FILE' FILE;
'REAL' XFROM, YFROM, SLOPEFROM, XTO, YTO, SLOPETO;
'INTEGER' LINEMODE;

This procedure defines a curved line piece between (XFROM, YFROM) and (XTO, YTO) with the boundary conditions

\[
\frac{dy}{dx} (XFROM, YFROM) = SLOPEFROM \quad \text{and} \quad \frac{dy}{dx} (XTO, YTO) = SLOPETO
\]

The parameter LINEMODE has the meaning as described in STRAIGHTLINEPIECE. Possible representations are: traced, dashed, dotted and dashed-dotted.
This procedure defines the axis, complete with tickmarks, numbering and a comment string. The axis from \((X_{FROM}, Y_{FROM})\) to \((X_{TO}, Y_{TO})\) has a specified number of intervals. At each intersection point tickmarks are situated perpendicular to the axis. There are three possible kinds of situating these tickmarks, dependent of the value of \(MARKTYPE\):

- \(MARKTYPE = 0\) : tickmarks on the upperside of the axis;
- \(MARKTYPE = 1\) : tickmarks on the lowerside of the axis;
- \(MARKTYPE = 2\) : tickmarks through the axis.

If \(MARKTYPE = 0\) or \(MARKTYPE = 2\) then numbering of the axis is performed on the lowerside of the axis else on the upperside. \(VALUEFROM\) contains the value belonging to \((X_{FROM}, Y_{FROM})\) and \(VALUETO\) contains the value belonging to \((X_{TO}, Y_{TO})\). The drawing machine determines which tickmarks can be numbered. The direction of writing these values is given by the parameters \(PARALLELNUMBERING\) and \(NUMBERING\) \(TO\) \(AXIS\).
The possibilities are shown in the table below:

<table>
<thead>
<tr>
<th>NUMBERING TO AXIS</th>
<th>'TRUE'</th>
<th>'FALSE'</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARALLEL NUMBERING</td>
<td>'TRUE'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the drawn number is parallel to the axis, the top of the digits is turned to the axis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parallel to the axis, the base of the digits is turned to the axis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>perpendicular to the axis, the direction of writing is to the axis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>perpendicular to the axis, the direction of writing is from the axis</td>
<td></td>
</tr>
</tbody>
</table>

If necessary a scaling factor (with representation: * @ < exponent >) is drawn, by which every number has to be multiplied. Finally a string can be added to the axis as comment. The direction of writing of the contingent scaling factor and the string is parallel to the axis. The base of the characters is turned to or turned from the axis dependent of NUMBERING TO AXIS.

SYMBOLHEIGHT indicates the height of the characters, used to comment the axis and is expressed in current data units.
Procedures based on straight line pieces and points.

'PROCEDURE' POLYGON(FILE, I, II, I2, XI, YI, LINEMODE);
'VALUE' II, I2, LINEMODE;
'FILE' FILE;
'INTEGER' I, II, I2, LINEMODE;
'REAL' XI, YI;

The polygon is defined by the sequence of points (XI, YI) for II ≤ I ≤ I2, i.e. the points (XI, YI) will be connected by straight line pieces.

'PROCEDURE' QUEUE OF POINTS(FILE, I, II, I2, XI, YI, POINTTYPE, POINHEIGHT);
'VALUE' II, I2, POINTTYPE, POINHEIGHT;
'FILE' FILE;
'INTEGER' I, II, I2, POINTTYPE;
'REAL' XI, YI, POINHEIGHT;

Every point in the sequence (XI, YI) for II ≤ I ≤ I2 is marked with a special marking symbol (see procedure POINT).

Basic procedures to draw a curve.

It is possible to define a curve by initializing the curve by two points and a contingent slope in the first point, a certain number of points and a termination of the curve, with the final slope at the last point given. It is essential that, while generating the curve, information about the last generated curved line piece is preserved. Therefore an array STATUS is used with bounds[0:5]. Initialization of the curve performs initialization of this array.

The total number of points has to be greater than or equal to 3.

'PROCEDURE' INITCURVE POINTS DERIV(X1, Y1, X2, Y2, DERIV, STATUS);
'VALUE' X1, Y1, X2, Y2, DERIV;
'REAL' X1, Y1, X2, Y2, DERIV;
'REAL' 'ARRAY' STATUS[0];

The curve is initialized by the two points (X1, Y1) and (X2, Y2) and the initial value of the slope \( \frac{dy}{dx} \) in the point (X1, Y1), given in DERIV.

The two points must be different.
The curve is initialized by the two points \((X_1, Y_1)\) and \((X_2, Y_2)\). These points must be different.

This procedure adds the point \((X_{\text{NEXT}}, Y_{\text{NEXT}})\) to the current curve. If we call \((X_{\text{NEXT}}, Y_{\text{NEXT}})\) point \(p_i\), \((i \geq 3)\) then the effect of the procedure is as follows:

1. the connection with the previous curved line piece is continuous in the first derivative;
2. if \(i = 3\), then the initial slope is equal to the given slope by \(\text{INITCURVE POINTS DERIV}\) or the initial slope is equal to the slope in point 1 of the circle, passing through \(p_1, p_2\) and \(p_3\) (initialization with \(\text{INITCURVE POINTS}\));
3. the slope in \(p_{i-1}\) is equal to the slope in \(p_{i-1}\) on the circle passing through \(p_{i-2}, p_{i-1}\) and \(p_i\).

The representation of the drawn curve is determined by the parameter \(\text{LINEMODE}\) (see \(\text{CURVEDLINEPIECE}\)).

The last curved line piece is generated. \(p_{i-1}\), the slope in \(p_{i-1}\) and \(p_i\) are already known (in \(\text{STATUS}\)).

The slope \(\frac{dy}{dx}\) in the last point is given by \(\text{DERIV}\).
'PROCEDURE' FINISH CURVE(FILE, LINEMODE, STATUS);
'VALUE' LINEMODE;
'FILE' FILE;
'INTEGER' LINEMODE;
'REAL' 'ARRAY' STATUS[0];

The last curved line piece is generated. \( p_{i-1} \), the slope in \( p_{i-1} \) and \( p_i \) are already known (in STATUS).

The final slope is the one on the circle, fixed by \( p_{i-1}, p_i \) and the slope in \( p_{i-1} \).

Procedures for drawing complete curves.

'PROCEDURE' MARKEDCURVE1(FILE, I1, I2, XI, YI, MARKING1, POINTTYPE, POINTHEIGHT, LINEMODE);
'VALUE' I1, I2, POINTTYPE, POINTHEIGHT, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, POINTTYPE, LINEMODE;
'REAL' XI, YI, POINTHEIGHT;
'BOOLEAN' MARKING1;

A complete curve without initial slope and final slope is generated, passing through the points \((XI, YI)\) for \( I1 \leq I \leq I2 \). It is necessary that there are at least 3 different points \((XI, YI)\).

The value of MARKING1 determines whether the point \((XI, YI)\) is marked with a special marking symbol, represented by POINTTYPE and POINTHEIGHT (see procedure POINT). The representation of the curve depends on the value of LINEMODE (see procedure CURVEDLINEPIECE).

'PROCEDURE' MARKEDCURVE2(FILE, I, I1, I2, XI, YI, STARTDERIV, MARKING1, POINTTYPE, POINTHEIGHT, LINEMODE);
'VALUE' I1, I2, STARTDERIV, POINTTYPE, POINTHEIGHT, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, POINTTYPE, LINEMODE;
'REAL' XI, YI, STARTDERIV, POINTHEIGHT;
'BOOLEAN' MARKING1;

The same as MARKEDCURVE1 except: the initial slope is given by STARTDERIV.
'PROCEDURE' MARKEDCURVE3(FILE, I, I1, I2, XI, YI, FINDERIV, MARKINGI, POINTTYPE, POINTHEIGHT, LINEMODE);

'VALUE' I1, I2, FINDERIV, POINTTYPE, POINTHEIGHT, LINEMODE;

'FILE' FILE;

'INTEGER' I, I1, I2, POINTTYPE, LINEMODE;

'REAL' XI, YI, FINDERIV, POINTHEIGHT;

'BOOLEAN' MARKINGI;

The same as MARKEDCURVE1 except: the final slope is given by FINDERIV.

'PROCEDURE' MARKEDCURVE4(FILE, I, I1, I2, XI, YI, STARTDERIV, FINDERIV, MARKINGI, POINTTYPE, POINTHEIGHT, LINEMODE);

'VALUE' I1, I2, FINDERIV, POINTTYPE, POINTHEIGHT, LINEMODE;

'FILE' FILE;

'INTEGER' I1, I2, POINTTYPE, LINEMODE;

'REAL' XI, YI, STARTDERIV, FINDERIV, POINTHEIGHT;

'BOOLEAN' MARKINGI;

The same as MARKEDCURVE1 except: the initial slope is given by STARTDERIV and the final slope is FINDERIV.

'PROCEDURE' CURVE1(FILE, I, I1, I2, XI, YI, LINEMODE);

'VALUE' I1, I2, LINEMODE;

'FILE' FILE;

'INTEGER' I, I1, I2, LINEMODE;

'REAL' XI, YI;

The same as MARKEDCURVE1 except marking.

'PROCEDURE' CURVE2(FILE, I, I1, I2, XI, YI, STARTDERIV, LINEMODE);

'VALUE' I1, I2, STARTDERIV, LINEMODE;

'FILE' FILE;

'INTEGER' I, I1, I2, LINEMODE;

'REAL' XI, YI, STARTDERIV;

The same as MARKEDCURVE2 except marking.
'PROCEDURE' CURVE3(FILE, I, II, I2, XI, YI, FINDERIV, LINEMODE);
'VALUE' II, I2, FINDERIV, LINEMODE;
'FILE' FILE;
'INTEGER' I, II, I2, LINEMODE;
'REAL' XI, YI, FINDERIV;

The same as MARKEDCURVE3 except marking.

'PROCEDURE' CURVE4(FILE, I, II, I2, XI, YI, STARTDERIV, FINDERIV, LINEMODE);
'VALUE' II, I2, STARTDERIV, FINDERIV, LINEMODE;
'FILE' FILE;
'INTEGER' I, II, I2, LINEMODE;
'REAL' XI, YI, STARTDERIV, FINDERIV;

The same as MARKEDCURVE4 except marking.

Procedures based on TEXT.

'PROCEDURE' NUMBER(FILE, XFROM, YFROM, XTO, YTO, ITALICITY, FORMATSTRING, VALUE);
'VALUE' XFROM, YFROM, XTO, YTO, ITALICITY, VALUE;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO, ITALICITY;
'STRING' FORMATSTRING;
'DOUBLE' VALUE;

With this procedure, the user is able to draw formatted numbers.

The string FORMATSTRING contains the information for formatting the number VALUE.

The value of FORMATSTRING must be conform to the syntax of < format string >.

\[
< \text{format string} > :: = < \text{free-field format} > | \\
< \text{fieldprecision letter} > < \text{field width} > | < \text{decimal places} > | < \text{field letter} > < \text{field width} > \\
< \text{free-field format} > :: = < \text{slash} > \\
< \text{fieldprecision letter} > :: = F | R | D | E \\
< \text{field letter} > :: = A | C | H | I | J | K \\
< \text{field width} > :: = < \text{unsigned integer} > \\
< \text{decimal places} > :: = < \text{unsigned integer} >
\]
'PROCEDURE' BOOLEAN VALUE(FILE, XFROM, YFROM, XTO, YTO, ITALICITY, WIDTH, VALUE);

'VALUE' XFROM, YFROM, XTO, YTO, ITALICITY, WIDTH, VALUE;

'FILE' FILE;

'REAL' XFROM, YFROM, XTO, YTO, ITALICITY;

'INTEGER' WIDTH;

'BOOLEAN' VALUE;

The value of the boolean VALUE is drawn with WIDTH characters. In fact the field precision letter L is used with field width = WIDTH.

Procedures based on AXIS COMPLETE.

'PROCEDURE' AXIS(FILE, XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS, MARKTYPE, VALUEFROM, VALUETO, PARALLELNUMBERING, NUMBERING TO AXIS, SYMBOLHEIGHT;

'VALUE' XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS, MARKTYPE, VALUEFROM, VALUETO, PARALLELNUMBERING, NUMBERING TO AXIS, SYMBOLHEIGHT;

'FILE' FILE;

'REAL' XFROM, YFROM, XTO, YTO, VALUEFROM, VALUETO, SYMBOLHEIGHT;

'INTEGER' NUMBER OF INTERVALS, MARKTYPE;

'BOOLEAN' PARALLELNUMBERING, NUMBERING TO AXIS;

The procedure defines an axis with tickmarks, numbering and a contingent scaling factor. Its effect is equal to that of the procedure AXIS COMPLETE, with exception of the comment string. The meaning of the parameters is the same as the corresponding ones in AXIS COMPLETE.

'PROCEDURE' BASIC AXIS(FILE, XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS, MARKTYPE, SYMBOLHEIGHT);

'VALUE' XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS, MARKTYPE, SYMBOLHEIGHT;

'FILE' FILE;

'REAL' XFROM, YFROM, XTO, YTO, SYMBOLHEIGHT;

'INTEGER' NUMBER OF INTERVALS;

The procedure defines an axis with tickmarks. The meaning of the parameters is the same as the corresponding ones in AXIS COMPLETE.
Other procedures.

'INTEGER' 'PROCEDURE' SCALE(TI, I, I1, I2, NUMBER OF INTERVALS, MODE, MIN, MAX);

'VALUE' I1, I2, NUMBER OF INTERVALS, MODE;

'REAL' TI, MIN, MAX;

'INTEGER' I, I1, I2, NUMBER OF INTERVALS, MODE;

If we call the set (1, 1.25, 2, 2.5, 4, 5, 8) a set of "basic round numbers", \( r_j, j = 1, 2, \ldots, 7 \), then we will call any number \( x = r_j \times 10^k \), for all integers \( k \), a "round number", and any integer multiple of a round number is called a "nice number".

For optimal subdivision of axes it is required:
1. the value of the length of an interval must be a round number;
2. each subdivision point must be indicated by a nice number;
3. the data must fit into the allotted space;
4. the data must occupy as much as possible of the allotted space.

The procedure calculates the values of MIN and MAX such that both MIN and MAX are nice numbers as defined above, satisfying:

\[
\text{MIN} \leq \text{TI} \leq \text{MAX} \quad \text{for} \quad \text{I1} \leq I \leq \text{I2}
\]

MODE may have the values 0, 1 or 2 with the following meaning:

- \( \text{MODE} = 0 \) the interval (MIN, MAX) is subdivided into precisely \( \text{NUMBER OF INTERVALS} \) subintervals;
- \( \text{MODE} = 1 \) the actual number of subintervals is between \( 0.625 \times \text{NUMBER OF INTERVALS} \) and \( \text{NUMBER OF INTERVALS} \);
- \( \text{MODE} = 2 \) the actual number of subintervals is between \( 0.625 \times \text{NUMBER OF INTERVALS} \) and \( 1.6 \times \text{NUMBER OF INTERVALS} \).

The value delivered by SCALE is the number of actual subintervals.

(See [1], [2])
The drawing machine.
The drawing machine is built out of two parts:
1. a control part; this part looks for data files suitable for plotting and hands then over to the drawing part;
2. a drawing part which does the actual drawing.

The control part.
The total drawing machine is a MCS, because it must be able to reach an online plotting device (Calcomp C565), so written in DCALGOL.
All data files that are presented to the drawing machine are identified by a prefix in the file title.
Prefixes:

PLOT11 → Calcomp 565
PLOT30 → Calcomp 1136
PLOTPR → PRINTER

WHILE NOT DISASTER
DO
BEGIN
IF THEREISAPLOT11DATAFILE
THEN BEGIN PUTFILEINTOQUEUEFORC565DEVICES;
    IF NUMBEROFACTIVEC565 < NUMBEROFC565PLOTTERS
    THEN STARTANOTHERC565PLOTTER
END;
IF THEREISAPLOT30DATAFILE
THEN BEGIN PUTFILEINTOQUEUEFORC1136DEVICE;
    IF NUMBEROFACTIVEC1136PLOTTERS < NUMBEROFC1136PLOTTERS
    THEN STARTANOTHERC1136PLOTTER
END;
IF THEREISAPLOTPRDATAFILE
THEN BEGIN PUTFILEINTOQUEUEFORPRINTERPLOTTER;
    IF NOT PRINTERPLOTTERBUSY
    THEN STARTPRINTERPLOTTER
END;
IF THEREISSOMEDATACOMFUNCTION
THEN PERFORMDATACOMFUNCTION;
END
This is a simplified version of the mainloop of the MCS.

For each plotter device a process is fired up whenever there is a data file to plot.

If another data file is found, its title is put into a queue to be interpreted by the plot process.

The plot process dies if and only if there is no other data file to plot. This approach has been chosen, because there might be several identical devices (for example we have two C565 plotters).

The processes attached to these devices share the same queue in order to prevent the plotting of a data file twice.

In order to prevent that a data file which title was already put into the queue will be identified again, its title prefix is changed (for example PLOT11 is changed to PLIT11).

The drawing part.

The simplified version of the drawing part (procedure AUTOPLLOT) is represented by the following program text.

For each device the same procedure is used.

WHILE SOMETHINGINMYDEVICEQUEUE
DO
BEGIN TAKENEXTFILETITLE;
IF FILEISPRESENT
THEN IF PLOTIT(FILETITLE) % THIS IS THE ACTUAL PLOTTING
THEN REMOVFILE(FILETITLE)
ELSE % SOMETHING WENT WRONG
PUTFILETITLEBACKINTOQUEUE
ELSE % PITY, FILE DISAPPEARED
END

The actual plotting is performed by PLOTIT.

The value of PLOTIT may be false which indicates that something went wrong while plotting (for example the DCP died).
Simplified text of PLOTIT.

INITIALISE;
PLOTIDENTIFICATION;
WHILE FILENOTEMPTY AND NOT ABORTED DO
BEGIN GETNEXTITEM;
   IF ITEMTYPE NEQ WINDOWMAP OR ITEMTYPE NEQ CLOSEBRACKET
      THEN ABORT(NOFSICALDRAWING)
   ELSE IF ITEMTYPE = WINDOWMAP
      THEN BEGIN CREATEWINDOW;
         BUILDMAPMATRIX;
         DRAWER(MAPMATRIX) % MAPS IMAGE TO PAPER
      END
   ELSE
   END;
PLOTIT := NOT ABORTED;

Simplified text of DRAWER.
% ALL INDICATED POINTS WILL BE TRANSFORMED
% ACCORDING TO THE MATRIX
WHILE FILENOTEMPTY AND NOT ABORTED AND NOT ENDOFIMAGE DO
BEGIN GETNEXTITEM;
   IF NOT VALIDITEM
      THEN ABORT(INVALIDITEM)
   ELSE
      BEGIN CASE ITEMTYPE OF
          BEGIN
             ENDOFIMAGE := TRUE; % CLOSE BRACKET
             BEGIN
                 CASE ELEMENTTYPE OF
                     % IMAGE ELEMENT
BEGIN

DRAWDOT;
DRAWPOINT;
DRAWSTRAIGHTLINE;
DRAWCURVEDLINE;
DRAWTEXT;
DRAWAXIS;
BEGIN

BUILDMAPMATRIX;
CREATEMATRIX(MAPMATRIX, MATRIX, NEWMATRIX);
DRAWER(NEWMATRIX)
END
END
END
END
END

This description of the drawing machine is a very brief one, but we hope it gives an idea about its structure. The actual drawing machine obviously looks quite different, although the global structure is maintained. The distinction between the several devices is made on the lowest possible level.
References.

J.A.Th.M. van Berckel, B.J. Mailloux

Stichting Mathematisch Centrum Amsterdam
January 1970.

[2] Prof.dr. F.E.J. Kruseman Aretz
Het plottersysteem in MILLI op EL X8.
Philips research laboratories, computernote nr. 1972/2.

[3] Dr.ing. José Encarnação
Untersuchungen zum Problem der rechnergesteuerten räumlichen
Darstellung auf ebene Bildschirmen.
Dissertation Technischen Universität Berlin.
Juli 1970.

Principles of interactive computer graphics.

4 april 1974.
Note: We want cassette tapes for the B6700.

At the Research Establishment Risø, many scientists log data from experiments. Some have large facilities with minicomputers, others have small arrangements that do not warrant expensive instrumentation. A paper tape punch has been their most cost-effective choice.

With the advent of cassette tapes (CT) this has changed. So we need a way to process the data on these tapes on our B6700. The most straight-forward method is to attach a CT station as a peripheral unit alongside the paper tape reader. We have been told that peripheral controls for CT exist for e.g. B4700. It can hardly be a problem for Burroughs to build one for B6700.

It has been said that CT are too slow for B6700. Well, we don't imagine using them for the system tape; damn it, but as faster (and hopefully more reliable) paper tape.

You may connect a CT station to an RJE terminal. Yes, but that is rather breakin a butterfly on a wheel, from the point of view of cost if not of efficiency.

We imagine that many other B6700 installations could utilize a CT station as a direct peripheral.

We imagine that Burroughs will build one if a few customers want it. So we urge you to tell your local Burroughs representative that he should ask Detroit to market CT stations on B6700.

Thank you!

Leif Hansson

COMPUTER INSTALLATION
DANISH ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT RISØ
DK-4000 ROSKILDE - DENMARK
Note: B6700 REMOTE JOB ENTRY INTERFACE FOR I.B.M. TERMINALS.

Philco Ford Corporation is pleased to announce the availability of a software patch for the B6700 N.D.L. which allows users to connect I.B.M. 2780 compatible terminals to the B6700 through an M.C.S. The most obvious advantage of this software is to broaden the spectrum of terminals that can be considered for an R.J.E. installation.

For additional information and pricing, please call Mr. Len Wisniewski collect at (215) 646-8600 ×4182.

As in each serious revolution, it will provoke violent opposition and one can ask oneself where to expect the conservative forces trying to counteract such a development. I don't expect them primarily in big business, not even in the computer business; I expect them rather in the educational institutions that provide today's training and in those conservative groups of computer users that think their old programs so important that they don't think it worthwhile to rewrite and improve them. Even if we know how to educate tomorrow's professional programmer, it is not certain that the society we are living in will allow us to do so. The first effect of teaching a methodology - rather than disseminating knowledge - is that of enhancing the capacities of the already capable, thus magnifying the difference in intelligence. In a society in which the educational system is used as an instrument for the establishment of a homogenized culture, in which the cream is prevented from rising to the top, the education of competent programmers could be politically unpalatable.

Edsger W. Dijkstra,
1972 ACM Turing Lecture.
Mailing list of JUB6700

corrections and additions.

Institutes, additions

Comissão Nacional de Energia Nuclear, Centro de Informações Nucleares
Rua General Severiano, 90 sala 316 - Botafogo
20.000 Rio de Janeiro - GB, Brasil

c/o Ivano H. Marchesi

Department of Agricultural Technical Services
Private Bag X116
Pretoria, 0001, South Africa

c/o Gert J. Smit

Fundacao Universidade de Brasilia
Brasilia, DF, Brasil

c/o ........

Heriot-Watt University
37-39 Grassmarket
Edinburgh EH1 2HW, Scotland

c/o A. Balfour

Instituto de Pesquisas Espaciais
São José dos Campos, SP, Brasil

c/o ........

Norges Kooperative Landsforening
Revierstredet 2
Oslo 1, Norway

c/o Reino Knaapila

Universidade Federal do Rio de Janeiro
Rio de Janeiro, GB, Brasil

c/o ........

Universidade de São Paulo
São Paulo, SP, Brasil

c/o ........

University of Illinois at Urbana - Champaign
Center for Advanced Computation
207 Advanced Computation Building
Urbana, Illinois 61801, USA

c/o Dave Subject
Institutes, corrections

Victoria University of Wellington,
page JUB 1-29:

Victoria University of Wellington, Computing Services Centre
c/o Robert H. Gordon

Private Bag
Wellington, New Zealand

Mailing list of Burroughs-representatives, additions

* Mr. Ulrich Gebhard, Large Systems Project Manager,
  Burroughs Machines Limited, P.O. Box 1958,
  Pretoria, 0001, South Africa.

* Mr. José Francisco Guimarães Costa, Manager Large Systems Support,
  Burroughs Electronico LTDA, R. Araujo Porte Alegre, 36,
  Rio de Janeiro, GB, Brasil.

* Mr. Bo Svensson, System Support Manager,
  Berzeliiig 12-14,
  S-41 253 Göteborg, Sweden.

* Mr. Donald H. Thurnau, Manager Applications Research
  Burroughs Corporation, Burroughs Place,
  Detroit, Michigan 48232, USA.

* will receive 1 copy.