View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by CWI's Institutional Repository

On the design of two small batch operating systems 1965 – 1970

F.E.J. Kruseman Aretz

November 14, 2012

Abstract

This paper describes the design considerations and decisions for two small batch operating systems, called MICRO and MILLI, for the Electrologica X8, a Dutch computer delivered from 1965 onwards. Their sole tasks were to run sequences of ALGOL 60 programs, thus transforming the X8 into an ALGOL 60 computer. They were developed in order to increase the efficient use of the hardware (MICRO) and to reduce waiting times for ALGOL 60 programs with small demands (MILLI).

The work described here was carried out mainly by one person, namely the author. The paper is interwoven with some personal history, describing a.o. the background and the context in which this work was carried out.

1998 Computing Science Classification

K.2. History of Computing, D.4.7. Batch Processing Systems

Keywords and Phrases

historical, operating systems, Electrologica X8

1 How I got into the field of operating systems

Educated as a theoretical physicist, I switched to computer science after completing my PhD. I had done quite a lot of numerical calculations for my PhD thesis and gradually I became more and more interested in (programming algorithms for) numerical analysis. Moreover, my interests in physics were directed to its foundations and I feared that my capacities for doing research in that direction were insufficient. So I preferred to enter a young and promising field and got a job at the Mathematical Center in Amsterdam, starting September 1, 1962. Fortunately I followed many classes in mathematics, including logic and numerical analysis, during my study.

My first task at the Mathematical Center was to write an ALGOL 60 program for the reconstruction of particle trajectories from Wilson cloud chamber data. Execution of my programs on the Electrologica X1, using the Dijkstra–Zonneveld ALGOL 60 implementation, produced wrong results that ultimately could only be explained by a bug in that

system. Dijkstra had left the Mathematical Center for a professorship at the Technical University Eindhoven and Zonneveld returned to his work as numerical analyst. With the help of one of my colleagues I repaired the bug in the ALGOL system, but now my interest in system programming was excited.

In the next period I was able to rebuild the ALGOL 60 system for the X1 into a load–and–go system, using the fact that in the period from 1960 to 1962 the X1 store was extended from 4K words to 12 K words [5]. Originally, the ALGOL compiler had to read the program tape twice and punched an object program tape, which afterwards had to be loaded for execution. Also the compiler and the loader had to be read beforehand, since the X1 had no backing store whatsoever. After the reconstruction, running an ALGOL 60 program boiled down to reading the system tape, the program tape (only once) and, if relevant, the data tape. Already at that time I liked to lower the thresholds for users of the computer.

In the course of the year 1963 the Mathematical Center ordered an Electrologica X8, the successor of the X1, and undertook the task to develop an ALGOL 60 implementation for it. Nederkoorn, Van de Laarschot, and I defined the mapping from a source text in ALGOL 60 to the object code for the X8 and coded the subroutines necessary for the support of object code execution (this work was mainly done by Nederkoorn and myself). Barning developed the subroutines for ALGOL's standard numerical functions.

The compiler was written by me in ALGOL 60 first and thereupon hand-coded in ELAN, the assembly language of the X8. The ALGOL load–and–go system was completed in 1965, before the first deliverance of the first X8. Although the X8 was about 12 times as fast as the X1, the ALGOL 60 implementation ran about 60 times faster than that of the X1, due to additions to the instruction set directed to the execution of ALGOL 60 programs [6].

Since it was clear that the operating system, to be written by Electrologica, would not be ready in time, a minimal operating system 'PICO' was developed at the Mathematical Center by Mailloux and Van Berckel. Its only task was to run ALGOL 60 program after ALGOL 60 program.

Before going into the operating systems 'MICRO' and 'MILLI' that I developed as successors of PICO, I first need to discuss some aspects of the X8 computer and the PICO system.

2 PICO

The PICO operating system for the X8 was really minimal. After initializing paper–tape reader, paper–tape punch, the operator's keyboard and typewriter, and, in its second version, also the line–printer, the operator was asked the date and the serial number (to be attached to the first ALGOL 60 program), whereafter it entered its main cycle. In that cycle the text of the next ALGOL 60 program was read, the program was compiled and, if syntactically correct, executed. Source text and data could be spread over several paper tapes. If the system asked, on the typewriter, for the next input tape, the operator had to place a tape in the tape reader. When the handling of a program was completed, the remainder of the current input tape was skipped, after which the cycle was repeated

with an incremented serial number.

The handling of input/output was elementary. For both the paper–tape reader and the paper–tape punch two core buffers (of length 32 and 150 words, respectively) were available, which were filled and emptied in turn. Especially for punch output, this implied long idle times for the cpu in case of much output. For the line printer (in the second release of PICO) 10 buffers of 37 words were available. In a buffer there was room for the 144 characters of a line[1]. But, due to the possibility of having underlinings and bars at each character position of a line, some lines needed three buffers. Also line–printer output could lead to cpu idle time.

The i/o system of the X8 was nicely organized. All i/o devices were equipped with a pushbutton having a green and a red bulb inside. In the red state execution of transput instructions kept pending, but caused the red bulb to flicker in order to draw the operator's attention. Pressing the pushbutton then brought it in the green state, in which transput instructions were honoured immediately. After completion of (the execution of) an ALGOL 60 program PICO skipped the (remaining) paper tape in the paper–tape reader until end–of–tape, which caused the paper–tape reader to pass into its red state. It was the task of the operator to await the end of paper–tape punching and of printing in order to tear off the output.

The operator had few possibilities to interact with the system. By pressing a key of the operator's keyboard he could rerun the last program executed with new data, break off a program's execution, prevent the compiler to be overwritten by the execution of a program or prevent the insertion into an object program of instructions to keep record of line numbers during program execution.

The structure of PICO was rather monolithic. It contained two separate modules: one for number conversion decimal–binary and binary–decimal (because it was clearly a separate subject) and one for the line–printer software (because it was added afterwards). The complete ELAN code of PICO has been documented in [3], together with that of the ALGOL 60 compiler and the run–time support routines[2].

The first X8 was delivered to the University of Utrecht and became operational November 1965. The first PICO system (in its version for a 16 K words core store and no line printer) was installed there that month. The Mathematical Center got its machine May 1966. The hardware included from the start 32 K words core store, a line printer, a teletype, 3 paper–tape readers, 3 paper–tape punches, and a drum store of 512 K words. Of these, PICO used only the teletype, the line printer, one of the paper–tape readers and one of the paper–tape punches.

PICO itself occupied some 3400 words for code, buffers and working space, ALGOL's run–time support system used about 700 words, and a library of standard functions and procedures[3] about 750 words. In a store of 16 K words, roughly 11500 words remained for object programs and their working space. The compiler, about 5000 words long, was

[1] A line on the line printer was 144 characters wide. The hardware required to pack these by 4 characters in a word.

[2] After loading of the system – from paper tape (!) in binary code, constituting approximately 100 m tape, read in about 40 seconds – the execution of the system started at the label DNTST.

[3] Besides those mentioned in the ALGOL 60 revised report the library contained procedures for input and output, which could be used in an ALGOL 60 program without declaration.

placed at the end of store and could be overwritten (if permitted by the operator) during program execution, on the penalty of having to reload the compiler from paper tape at the end of that execution.

The main drawbacks of PICO became clear when the load of the X8 increased, although it was about 60 times faster than its predecessor. These drawbacks were:

- inefficient use of the hardware, due to much idle time of the cpu waiting for i/o completion, and
- long waiting times for users with only a small demand of computing time.

The latter was slightly compensated by daily having certain periods in which only two–minute programs were dealt with, thereby, however, decreasing the overall efficiency.

Soon, the need was felt to replace PICO by some more advanced operating system.

3 MICRO

Not long after November 1965, the moment that the ALGOL 60 system embedded in the PICO operating system became operational, Van Berckel left the Mathematical Center and Mailloux was involved in the design of ALGOL 68. Therefore the development of a more advanced operating system became my responsibility.

I had no previous experience in designing operating systems at all, nor had I knowledge on the subject from literature. I was, however, aware of the troubles with parallel program interaction and synchronization. It was therefore important to keep the system as simple as possible.

Two ideas to improve the performance of PICO arose naturally. They were:

- using the drum store of 512 K words to buffer all input– and output streams, and
- to run programs with short execution times in parallel with programs with larger cpu–time demands by using the drum store to swap complete programs out and in again.

As an exercise the system PICO was changed into the system PICO–PR/DR, operational October 1966, in which all line–printer output was buffered on the drum.

Already in the summer of 1966 ideas were developed into the direction of parallel execution of programs. The need for more efficiency was, however, so high that I gave priority to extend PICO–PR/DR to a system in which all transput streams were buffered on the drum. Moreover, I hoped to learn a lot from its design and to be able to use many of its parts in future systems. The new system, called MICRO, was assembled for the first time in February 1967 and became fully operational in the summer that year. Now it was possible to read the tapes of program after program in advance during the execution of a cpu–demanding program, thus greatly enlarging the efficient use of the cpu greatly. In the meantime the punching of paper–tape output and the printing of line–printer output of preceding programs could continue, if not yet completed.

For MICRO the code for paper–tape reader, paper–tape punch and line–printer had to be largely rewritten. This code was now divided into separate sections, resulting in a more or less modular structure of the system (as opposed to the monolithic structure of PICO).

While the main idea of a cycle in which ALGOL 60 program after ALGOL 60 program was elaborated was kept, the complete system now contained the following 'modules':

- main section,
- number conversion,
- drum section,
- tape–reader section,
- tape–punch section,
- line–printer section,
- object–program run–time support system,
- library section,
- ALGOL 60–compiler section, and
- system–tape punching section[4].

The tape–reader section, the tape–punch section, and the line–printer section contained similar subroutines for dealing with the transport from core buffer to drum and vice versa. The result was an almost doubling of size: MICRO's code, core buffers and working space occupied more than 6100 words of core store.

The main problems to be solved were the efficient use of core store, drum store, and drum channel.

For all three i/o devices it was decided that first the producer of the data had to store them in one of two core buffers (used in turn) available to the producer, next they were transported to the drum, and finally to one of two core buffers available to the consumer. For simplicity reasons this path was never cut, even if the consumer was waiting for the data.

In general a producer of data sent data to the drum only when, and as soon as its current buffer was full. An exception had to be made for the last data belonging to a program: it had to be offered to the drum immediately. This implied that each buffer contained information about the amount of data contained in it.

The drum of the X8 contained 512 tracks of 1024 words each. Its revolution time was 40 msec. A first decision was not to try to keep record of the drum position in time. Therefore the average waiting time for a drum transport was 20 msec and the transport time roughly the number of words $\times$ 40 μsec. This made it advantageous to use a large buffer size. On the other hand, having 3 i/o devices using altogether 12 core buffers asked for a minimal buffer size. As a good compromise a buffer size of 128 words was chosen for the paper–tape reader and punch, with an average transport time of 26 msec (waiting included), whereas for the line–printer buffers of 256 words were adopted, with an average transport time of 31 msec[5].

Again for simplicity reasons each of the three i/o devices was assigned a fixed area on the drum. This area was used cyclically. The tape reader possessed 192 K words (good

[4]activated at the end of system assemblage.

[5]A drum transport could start at an arbitrary drum address and was at most 4096 words long.

for 1536 buffer portions of 128 K), the tape punch 40 K words (320 portions), and the line–printer 128 K (512 portions).

For the sake of efficient use of both drum space and drum channel the information of the tape reader and for the tape punch were packed in three columns of 9 bits width. The lines for the line-printer were already in PICO packed with 4 characters per words as expected by the line–printer hardware. For these, three 39–word buffers were available. When a line was completed these buffers were copied into one of the 256–word buffers. Each of the latter buffers should contain a number of complete lines[6].

The drum was also used to store the whole system. The system could be loaded by reading a rather short paper tape containing merely instructions to load the system from drum. Now it was also possible to reload the ALGOL 60 compiler from drum, whenever it was overwritten during the execution of a program.

Altogether there were, after loading of the system, 7 processes to compete for the use of the drum channel: 3 to–drum processes and 4 from–drum processes. The system should be such that none of these processes could be passed over systematically when in need for a transport. The method chosen was as follows:

For each of these processes one bit was reserved in a word in core store called '*attention*'. Whenever a process wanted to use the drum channel, it raised its bit in *attention* in order to indicate its need for transport. If the drum channel was idle the transport could start immediately, otherwise it had to be postponed. After completing a drum transport the drum channel cleared, in its interrupt program, the bit in *attention* corresponding to the completed transport and inspected whether there were processes waiting for transport. If so, it chose the next one in a circular fashion. By this 'round robin' method, that I learned from Mailloux, every process got its turn and had to let precede at most 5 other processes.

There were a number of conditions to be fulfilled in order to maintain the liveliness of the i/o system (later I learned to call them system invariants). We met already one of them:

- ($attention \neq 0$) $\equiv$ (drum channel active).

Some others read:

- (consumer buffer empty) $\wedge$ (corresponding drum buffer not empty)
  $\equiv$
  (corresponding from–drum process activated[7]).
- (non–empty tape–punch buffer available) $\equiv$ (tape punch is punching).

An interrupt of the tape punch could therefore invalidate three of these invariants:

1. In the first place it inactivated the tape punch, and if not all three columns of the current buffer were punched or if the other tape–punch buffer was filled, the tape punch had to be reactivated.

[6]Between the areas in the buffer occupied by the lines 3 words were reserved for the so–called start instruction for CHARON (the i/o processor of the X8).

[7]i.e., drum transport announced in *attention*.

2. Next, it possibly emptied a tape–punch buffer, and if so and if there was tape–punch stock on the drum, a request for refill had to be submitted to the drum channel.
3. In that case, if the drum channel was passive, it had to be activated.

At the same time Dijkstra was developing the THE system at the Technical University Eindhoven. There he devised the technique of cooperating sequential processes[1], mutually synchronized by semaphores with P and V operations and interacting with one another in critical sections through common variables.

It is not so easy to map our implementation on his method. First of all MICRO does not contain a process scheduler. Essentially one sequential program is running. That (main) program is interrupted from time to time by a peripheral interrupt program. Some pieces of code are executed either by the main program or by one of these interrupt programs. An example is the activation of a from–drum activity, which can be done by the main program while reading an input character, thereby emptying one of its core buffers, or in the drum interrupt program itself, while finding that the other core buffer is empty already. Mutual exclusion is, where necessary, obtained by executing pieces of the main program in 'deafness', i.e. excluding peripheral interrupts temporarily. The only clear example of a semaphore is the use of *attention* by the drum interrupt program. The number of bits '1' in *attention* is comparable to a semaphore value and a kind of P–operation is applied to *attention* to find out whether further activity of the drum is to be postponed or not.

Since the ultimate source of all activity is the sequence of ALGOL 60 programs as read into the system by the paper–tape reader, during system initialization the tape reader is started (by giving it a read instruction). As soon as a paper tape is put into the tape reader (and the push button is pressed in order to activate it) the first core buffer is filled, transported to the drum, transported to core again, and read by the main program. In turn the output generated by the main program started up the other i/o streams.

In order to separate the input tapes belonging to one and the same program from those of its successor, the operator now had to press one key on the command teletype[8]. That key could be pressed during or after the reading of the last tape belonging to a program, something that also required careful implementation.

The MICRO system worked perfectly and was taken over by Philips Research Laboratories in Eindhoven (installed there by myself) and by the Technical University Eindhoven (where is was kept in use along with Dijkstra's THE system after completion of the latter).

4 Extentions of MICRO

During the summer of 1967 the X8 installation of the Mathematical Center was extended and reconfigured. Therefore it was out of operation for several weeks(!). Among the extensions was a third core–store cabinet of 16 K words[9].

[8] either N (for a new data tape for a fresh start of the last program executed) or an E (for announcing a brand new program).

[9] The extension further consisted of 4 teletypes, a punch–card reader, and a punch–card punch. These were not used in MICRO.

The use of the new core store, with addresses running from 32 K to 48 K, required an adaptation of the ALGOL 60 system. This was caused by the fact that X8 instructions have an address part of 15 bits, thus addressing directly core store from 0 to 32 K. Using indirect addressing (with 18 bits addresses) the words of the third cabinet were within reach. By keeping the locations of the compiler, the run–time support system, the library, and all object programs produced by the compiler below the 32 K border and allowing only the run–time stack to grow into the third cabinet only a few changes in the system were required. It became operational in September 1967.

Already in February/March 1967 the library of the ALGOL 60 system was extended with a number of procedures and functions. Especially important were the functions for inner products of vectors and matrix columns (to accelerate matrix operations by avoiding repeated indexing) and the procedures 'todrum' and 'fromdrum' by which users could save and retrieve the contents of arrays to and from a specially reserved area on the drum of 80 K words[10].

Another extra facility (started by the operator command 'C') was the use of the second paper–tape reader and the second paper–tape punch to copy, quasi off–line, paper tapes. This was done in the simplest way: alternately reading and punching one character at a time, completely carried out in interrupt programs.

In the end the operating system MICRO took 6080 words, as compared to PICO's 3400 words. Including run–time support and library about 8080 words were reserved for the system, leaving a 40 000 words for object programs and their execution stacks[11].

5 MILLI

Already in 1967 work started on the development of a new operating system, MILLI. The main purpose was to almost eliminate waiting times for ALGOL 60 programs with small demands, by executing them in parallel with the main stream of programs.

This posed a number of new problems to be solved and decisions to be taken. We mention:

1. If there are several parallel program streams, each with its own limitations, how to assign a program to a certain program stream,
2. How to accommodate several programs concurrently in execution into core store,
3. How to find place on the drum for buffering so many different transput streams,
4. How to restrict the growth of system code (at the expense of the space for ALGOL 60 object programs and their stacks), also if new input and output devices are included such as a second paper–tape reader, a punch–card reader, a punch–card punch, or a plotter.
5. How to keep the system simple, clearly structured, and easily extensible.

The major design decisions were already taken in 1967, in the time that I worked at the Mathematical Center. There I also started programming the essential kernel of the

[10]The ALGOL 60 program could continue its execution, as long as it did not access the array under transport. An untimely access enforced the program to await completion of the transport. This was implemented using the mechanism of a wrong–address interrupt.

[11]For a 48 K core store, with a restriction to object–code length.

system, and I was able to test it at night hours. The completion of MILLI, mainly filling in the details, however, was to be completed after my transfer to the Philips Research Laboratories in Eindhoven.

Simplicity. To keep the system simple it was decided not to try to develop any kind of virtual storage (as was done by Dijkstra for the Technical University of Eindhoven). All user programs were allowed to use the full core store left over by the system and (quasi) parallel execution of programs was realized by swapping a complete program from core store to the drum and another program from drum to core store for giving it a time slice of computing before it, in its turn, had to make room for another.

Also for simplicity reasons it was decided that each ALGOL 60 program and its input data should be read into the system by one input reader, be it one of the paper–tape readers or the punch–card reader. In order to simplify the work for the operators it should be clear at the start of the input to which of the several program streams that input should be assigned.

To simplify the assignment of output devices only one of the program streams was allowed to make use of a paper–tape punch, a punch–card punch, a plotter or of the 80 K words on the drum used by the procedures *todrum* and *fromdrum*, thus avoiding the use of the 'Banker's' algorithm [2]. The only output device shared by all programs had to be the line–printer.

It was expected that the buffer space on the drum would be too small to collect all pages generated during the execution of an ALGOL 60 program before printing them. It should be possible to print these pages in portions, possibly alternated with portions of other programs in execution.

The natural unit to be printed on the line–printer was a full page (consisting of a head line and 60 user lines). To avoid tearing off line–printer portions too frequently, portions contained a number of pages, say 10 to 15. The head line of each page of a portion had the same tear–off number, an increment of the tear–off number indicated the place for the operator where to separate the portions.

Use of the drum store. All these decisions implied a restructuring of the use of the drum. Instead of 3 data streams to be buffered on the drum now at least 11 had to be accommodated[12]: 4 input streams (originating from the paper–tape and punch–card readers), one paper-tape punch stream, one card–punch stream, a plotter stream, and 4 line–printer streams.

In MICRO, 360 of the 512 K words of the drum were available for i/o buffering, and all three data streams got a fixed portion (we never measured how well these were used). In MILLI such a generosity was impossible. First of all many more data streams had to be served and, secondly, less drum space was available since several swap areas on drum (for programs quasi executed in parallel) were needed too.

The main idea was to divide the available drum store into pages of 1 K words and to give each of the 11 data streams a minimal claim of pages. In this way it could be guaranteed

[12]In case of 4 parallel program streams.

that each stream could proceed even if some other streams were filling up buffer space at an enormous rate.

In fact 267 pages were assigned to three portions of pages: 147 pages for the 4 input streams (with individual claim sizes of 5), 100 pages for the 4 printer streams (with individual claim sizes of 19), and 20 pages for the three other output streams (paper–tape punch, card punch, and plotter, with individual claim sizes of 2).

The decision to assign drum space to processes in portions of 1 K words (rather than 128 words as done in MICRO) followed from the requirement to chain successive portions of a process and to keep the administration for the chains in store. There were in fact 3 chains of free portions and 11 chains of assigned portions. Now 267 words of store were sufficient, where otherwise 2136 words would have been needed.

The four input streams, each having in all events 5 drum pages available, were competing for the remaining 127 pages (using, when exhausted, an *attention*–like scheduler to obtain a page when released by one of the input streams). If only one of the input streams required more than its claim, it could get maximally $147 - 3 \times 5 = 132$ drum pages, good for 1056 buffers with the capacity of storing 402 336 paper–tape punchings (i.e., more than 1000 m of punch tape corresponding to at least 7 minutes reading time) or at least 3350 punch cards[13].

Special attention had to be paid to the printer streams. When starting a new page, thereby completing the previous one, the latter was not automatically allowed to be printed. Only when the amount of buffers filled by a number of successive pages had exceeded the lower print limit of 152, a printing permit was given for that set of buffers. A special measure, however, was needed to prevent that a page used too much buffer space before a new page was arrived at. This might occur since the ALGOL 60 programmer had a command *carriage(0)* at his disposal to fill a line of a line–printer page over and over again. After filling more than 190 buffers since the last print permit was given the execution of his program had to be fired with an error report. In normal cases a printing permit was given for each 10 to 15 pages. Of course, printable portions of the 4 program streams had to compete for the line printer, again via an *attention*–like scheduler.

Code–length reduction. When coding the MICRO system I already observed that much code was repeated, with only small variations. Each of the tape–reader, tape–punch, and line–printer sections contained its own *todrum* and its own *fromdrum* routine, which differed by process numbers, drum areas, and core addresses only. In MILLI there were many more processes, partly with even less differences (more than one tape-reader, several input streams, several printer streams), as well as several *attention*–like schedulers. In order to reduce the code I decided to code these parts only once, parametrized by an entry to one of a number of tables containing the necessary specific data. This parametrization made large parts of the code re–entrant.

All these tables contain the variables, constants, instructions, references to other tables, and code addresses relevant for the activities concerned. They are located at the very end of the core store (in the part that could only be addressed indirectly). There are 5 types of tables, for programs, data streams, drum processes, devices, and schedulers. All tables

[13]Two columns per word, but trailing blank columns were not stored at all.

of the same type have a fixed initial part, possibly followed by some additional fields.

The 55 tables of the final version of MILLI are enumerated in Figure 1. They occupied 789 words.

program table	data stream	drum process	device	scheduler
				program base
program 1	restream 1	process 0		batch a
	prstream 1	process 4		pr–chain base
program 2	restream 2	process 1		batch b
	prstream 2	process 5		
program 3	restream 3	process 2		batch c
	prstream 3	process 6		
program 4	restream 4	process 3		batch d
	prstream 4	process 7		
	tpstream 1	process12		out–chain base
	cpstream 1	process14		
	plstream 1	process16		
		process 8	tape reader 1	in–chain base
		process 9	tape reader 2	
		process10	card reader 1	
		process11	printer 1	printer base
		process13	tape punch 1	
		process15	card punch 1	
		process17	plotter 1	
		process18	drum 1	drum base
			teletype 1	
			keyboard 1	
			clock 1	

Figure 1: The 55 tables used in MILLI

Figure 2 reproduces the contents of table ‘tpstream 1’ [14].

address	contents	comment
tpstream 1[0]:	:out–chain base	” address scheduler for drum pages
[1]:	128	” buffer length
[2]:	’skip’ 1	” number of pages - claim
[3]:	2	” claim size
[4]:	’skip’ 1	” level of buffer in drum page
[5]:	’skip’ 1	” address of current drum page
[6]:	’skip’ 1	” number of unsatisfied requests
[7]:	:process12	” base address of the to–drum process
[8]:	:program 4	” address program table
[9]:	2	” bit to request a new drum page
[10]:	’skip’ 1	” number of buffers
[11]:	’skip’ 1	” address of core buffer

Figure 2: Contents of table ’tpstream 1’

The 10 schedulers schedule the activities described in Figure 3. The code to handle these 10 schedulers is given only once.

[14]’skip’ is an assembler command to reserve one word for a variable.

program base:	cpu assignment to a program
batch a:	access to restream 1 for an input device
batch b:	access to restream 2 for an input device
batch c:	access to restream 3 for an input device
batch d:	access to restream 4 for an input device
pr–chain base:	assignment of a drum page to a print stream
out–chain base:	idem to the tape–punch, card–punch or plotter stream
in–chain base:	idem to a tape–reader or the card–reader stream
printer base:	access to the printer for a prstream
drum base:	transport to and from the drum store

Figure 3: The 10 schedulers of MILLI

This arrangement has many advantages. Apart from the reduction of code length it provides great flexibility: adding another paper-tape punch would imply adding some new tables and adapting a few existing ones without any need to change the monitor code. Changing the division of drum space over the several data streams means the adaptation of some table constants.

So far the main decisions were already taken at the Mathematical Center. There I was also able to test large parts of the code. For that purpose I wrote some special main programs, running in 'parallel', that read paper tape from two paper–tape readers and printed the information on the line printer. In order to test the schedulers I used very small drum–page allowances.

As mentioned before, I was not able to complete the MICRO system in my time at the Mathematical Center. Below I will try to explain what was the cause of this and why I left the Mathematical Center for Philips Research Laboratories Eindhoven in August 1969. Only thereafter I will proceed with the review of MICRO.

Intermezzo. August 1, 1965 J.A. Zonneveld, group leader of the Computation Department, left the Mathematical Center to become the group leader of a newly established computing facility at Philips Research Laboratories in Eindhoven. Then Th.J. Dekker and I were appointed as his successors, Dekker for the numerical analysis and I for the programming activities. Alltogether I had to give guidance to about 5 staff members, each working with a different subject, and to a small team around the computer, an Electrologica X1. Since the Mathematical Center also acted as an (informal) computing center for the University of Amsterdam, I also had discussions with researchers from the university over service possibilities. But initially my work as section leader left me ample time to do what I preferred to do, namely to develop software.

But little by little my duties grew. In 1967 I was appointed 'bijzonder hoogleraar' [15] in applied mathematics at the University of Amsterdam for one day a week on account of

[15] In the Netherlands, scientific foundations can get permission to appoint (and pay) professors at a university.

the Stichting voor Hoger Onderwijs in de Toegepaste Wiskunde [16]. For this I developed a course on programming, dealing with compilers and operating systems, but also treating Turing machines and showing the importance of recursion. Around the Electrologica X8, delivered at the Mathematical Center in 1966, grew a (small) computer center, with operators, punching personnel, and programmers. My staff grew to 9 staff members, 4 assistants, 12 programmers, 6 operators and 3 punch typists end 1968. With some staff members we also joined a working group, started by the computer center of the University of Utrecht, to develop a multi–access system[17].

Already in 1968 I felt overworked. My stomach became troublesome and my time at home I mostly spent lying on the couch. Late in the year I discussed my problems with Van Wijngaarden, director of the Mathematical Center, and asked for assistance to lighten my managerial duties. He promised to arrange something but never did. Maybe this was caused by the fact that he had worked too hard himself to bring the conception of the programming language ALGOL 68 in the IFIP WG2–committee on his name (in my opinion he ruined, in this way, his health forever).

One of his remarks was that postponing some activities sometimes helps: that can make them superfluous. One good advice was to take a skiing holidays, which I did in the first two weeks of 1969. But in the course of that year I decided to look after a job outside the Mathematical Center, a job in which I could spend my time in doing research myself and in writing software. A full professorship at the University of Amsterdam (aimed at starting a computing science program), which was offered to me, did not fit my desires. After ample hesitations I accepted a job at the Philips Research Laboratories in Eindhoven. The computer center there, under the supervision of my old sous–chef Zonneveld, had an X8 (later even two of them) at its disposal and gave me the opportunity to complete the MILLI operating system. My condition, never to be 'promoted' into a leading position, was accepted and indeed honoured during the remainder of my professional career. I continued my professorship at the University of Amsterdam for another two years. In 1971 I was appointed extra–ordinary (= part–time) professor at the Technical University Eindhoven.

Before I left the Mathematical Center end July 1969, I introduced the structure of MICRO to a group of my cooperators during 7 sessions. Maybe I had better trained a software group before, but I like to work all by myself and cooperation is not one of my stronger qualities.

After this detour I come back to the further development of MILLI, now at Philips Research Laboratories.

The completion of MILLI. From August 1969 I had ample time to work on the completion of MILLI. The backbone and all major decisions taken already, this came down to filling in lots of details. This was done in close consultation with two cooperators, W.P.J. Fontein and A.J. Dekkers, and with Zonneveld. An important aspect was the interface with both the users and the machine operators. I first deal with the user interface.

[16] Foundation for Higher Education in Applied Mathematics.

[17] For this purpose a magnetic disk of 2 M words(!) and 4 teletypes were purchased. When leaving the Mathematical Center I had not seen them in use.

The user interface. As earlier already indicated implicitly it was decided to have four program streams or batches indeed, called a, b, c, and d, differing in maximal allowed program execution time and output devices. The choice of a stream was left to the user. In an obligatory first line he had to specify the program stream, some user data, and optionally some allotments:

<monitor heading> ::=
  <stream letter> <box number>.<group number>.<user number>,<name> <directions>

<directions> ::= <empty> | ,<direction><directions>

<direction> ::= z | b<int> | k<int> | p<int> | r<int> | t<int> | u1 | u2

An example of a monitor heading is:

d23.86043.5053,p de vries,t300,b100

asking specifically for 300 millihours computer time[18] and 100 000 punchings on the paper–tape punch (in stead of the default portions of 30 millihours and 50 000 punchings); for line–printer (direction letter r), card punch (direction letter k), and plotter (direction letter p) the default portions for batch d of 15 pages, 1000 cards, and 200 millihour plotter time were allowed.

Direction 'z' indicated that the printing of the ALGOL source program should be suppressed, whereas directions 'u1' and 'u2' specified a certain reaction on the occurrence of floating–point underflow, different from the default one.

For introducing the card reader as an input source the representation of the ALGOL 60 alphabet on cards had to be decided upon. This was done together with the computing centers of Utrecht University and the Technical University Eindhoven[19]. Furthermore it was decided that (X8–ALGOL 60) procedure *rehep* would give the user program access to the individual columns of punch cards, whereas procedure *resym* would translate columns to symbols (in internal representation). In the operating system the uninterpreted columns had to be transported to user programs via the drum[20]. The first card of a program should contain the monitor heading. This concludes the discussion of the (new) interface for the users.

The machine operators' interface. Next we turn to the interface for the machine operators. It should also be as simple as possible, minimizing their work (as was already pursued in PICO and MICRO) and also giving a good insight in the status of the machine on the command teletype.

On this teletype each status change was reported by a line. As an example we present a number of successive teletype lines in Figure 4.

Most lines start with the time, followed by 4 numbers, giving the number of programs in the four program batches, (figure 9 meaning 9 or more). An asterisk preceding a figure indicates that the last program still is read on one of the input devices. Then follow two letters, the first one indicating the program batch, the second one on which of the input

[18] Since the X8 use was priced fl 1000 per hour, the millihour was used as unit of computer time.

[19] Among other things, this concerned the representation of '[', ']' and '$_{10}$'.

[20] In compact form: two columns packed into one word, at all invents the first two columns, blank columns at the back not transmitted but regenerated by *rehep*.

event	teletype line
starting paper–tape reader 2 for a new program	1315 2 *2 1 1 BS
the completion of program execution in batch a	1316 1 *2 1 1 AP 035 86123.2213 0024
the (elaboration) start of a program in batch a	1316 1 *2 1 1 AP 038 86043.5053
operator command S on teletype keyboard	**S**
end of tape on tape reader 2	1318 1 2 1 1 BS
starting card reader for a new program	1318 *2 2 1 1 AK
operator command K on teletype keyboard	**K** 0012/0018 0103/0047 0361/0139 0256/0744
termination of a program due to time exhaustion	1320 *1 2 1 1 AP 997 038 86043.5053 0030
the start of next program in batch a	1320 *1 2 1 1 AK 039 86123.2213
after a card with 7-9 in the first two columns	1320 1 2 1 1 AK

Figure 4: Fragment of a command teletype report

devices it is or was read[21].

- In the first line of the table the tape of a new program for program batch b is put into tape reader 2, that device is started and destination b is recognized.
- In the second line the program in batch a is completed; its serial number and its computer time consumption are printed (the latter in millihours).
- Since there is still one program in batch a, that is started; the third line is printed after analyzing its monitor heading and assigning it the next serial number.
- In the fourth line the operator has pressed key 'S', indicating that the paper tape now being read in tape reader 2 is the last tape for the program.
- When that tape is read to its end, the program is administrated as complete.
- Line six reports the introduction of a new program for batch a on the card reader.
- Operator command 'K' gives for all four batches two numbers: the computer time consumed and the computer time left over for the programs in elaboration.
- The execution of the program in batch A is terminated by the system because of exceeding the time limit, indicated by error number 977.
- The next program in batch A, although not yet completely read, is taken into execution.
- A card with 7-9 holes in its first two columns was appended by the operator after each program on punch cards.

By typing a letter 'A', 'B', 'C', or 'D' the operator could terminate the execution of the program in the corresponding batch (to be applied if that program was clearly malfunctioning, e.g. by generating lots of empty line–printer pages).

All output, also that for the paper–tape punch, the card punch, and the plotter, was marked with the serial number, the box number, and the group/user number of the program in a form that was easily readable by both the operator and the user.

The use of MILLI. MILLI was put into operation in February 1970. It made a tremendous difference: for small jobs as well as for syntax checking of new programs users could wait for the results. MILLI was soon adopted by the Mathematical Center. It was also offered to the Technical University of Eindhoven, but there Dijkstra's THE system came into use as a multi–programming system along with the continued use of the MICRO system.

The time slice was chosen to be 15 seconds. This was short enough to suggest parallel execution of the four batches, at the same time long enough to make the time needed for

[21] P(rimo) for tape reader1, S(econdo) for tape reader2, and K(aart) for the card reader.

program swap acceptable: depending on the number of words to be swapped to and from the drum in the order of 1 second.

In its first version the software for the card punch and the plotter still had to be written; these were added gradually.

Another addition was the possibility to store on the drum a set of pre–compiled ALGOL 60 procedures and functions that could be called in a program without further notice or declaration. If in a program such a library routine was called, its code was automatically loaded from drum and added to the object program at the end of compilation. For this facility only small changes to the ALGOL compiler were necessary, both for the pre–compilation of the library and for the insertion in the object programs of the code of called procedures or functions.

A final addition to MILLI was the ability to use ISO code instead of Flexowriter code on paper tape for programs and their input data.

The development of MILLI would not have been possible without the experience and the know–how obtained in the construction of MICRO: the step from PICO to MILLI would have been too large. In retrospect MILLI was, for a single programmer without previous experience in operating–system design, a huge job. Nevertheless no essential mistakes were made and astonishingly little testing was required to come to a truly reliable system.

The final operating system used about 6600 instructions (to be compared with the 6080 words of MICRO). The bulk of it, 5120 instructions, were situated in directly addressable store. Only the plotter section, the code for wrong–parity interrupts, and some conversion tables were positioned in indirectly addressable store. For ALGOL 60 object code and its working space were alltogether 25856 words available.

6 Final remarks.

Apart from the card–punch section all the code was written, punched, tested, and maintained by one man only.

It was written in X8 assembly language; I never experienced that as a problem or a drawback.

Of course the text–editing facilities of today were totally absent. All the code was typed on a FRIDEN Flexowriter, a typewriter which produced both a print on paper and a paper–tape encoding. A Flexowriter could also read and print a paper tape and, if so desired, re–punch the tape; in this way it could be used for corrections. All this was rather labour–intensive. For MILLI allmost all code was rewritten and, therefore, retyped.

Was the work meaningfull in the time that the first time–sharing systems came into existence[22]?

We should not forget that the hardware of the Electrologica X8 lend itself hardly to build time-sharing systems. It had no monitor/user mode, it had, at least initially, no memory protection, it had no virtual memory facilities (allthough Dijkstra's THE system

[22] e.g. CTSS (1961), MULTICS (1969), UNIX (1969)

implemented virtual memory for the X8), and magnetic–disc units were small (8K words) and expensive.

Acknowledgement
I would like Philips Research Eindhoven for giving me the opportunity to write this paper. I am also grateful to Jan Korst who was so kind to read it and to make useful comments.

References

[1] E.W. Dijkstra, "*Cooperating sequential processes*",
EWD 123, 1965.
reprinted in F. Genuys (ed.), "*Programming Languages*",
Academic Press 1968.

[2] E.W. Dijkstra, "*Een algorithme ter voorkoming van dodelijke omarming*",
EWD 108, 1964.
E.W. Dijkstra, *The mathematics behind the Banker's Algorithm*",
EWD 623, 1977.
reprinted in E.W.Dijkstra, "*Selected writings on computing: a personal perspective*",
Springer Verlag 1982, p. 308–312.

[3] F.E.J. Kruseman Aretz and B.J. Mailloux, "*The ELAN source text of the Mathematical Center ALGOL 60 system for the EL X8*",
Mathematical Center report MR 84, Mathematisch Centrum 1966

[4] F.E.J. Kruseman Aretz, "*Doelstellingen en achtergronden van de operating systems PICO, MICRO en MILLI*",
Informatie 16 (1974) 672–678 (in Dutch).

[5] F.E.J. Kruseman Aretz, "*The Dijkstra–Zonneveld ALGOL 60 compiler for the Electrologica X1*",
CWI report SEN–N0301, Centrum Wiskunde en Informatica, June 2003.

[6] F.E.J. Kruseman Aretz, "*A comparison between the ALGOL 60 implementations on the Electrologica X1 and the Electrologica X8*",
CWI report SEN–E0801, Centrum Wiskunde & Informatica, September 2008.