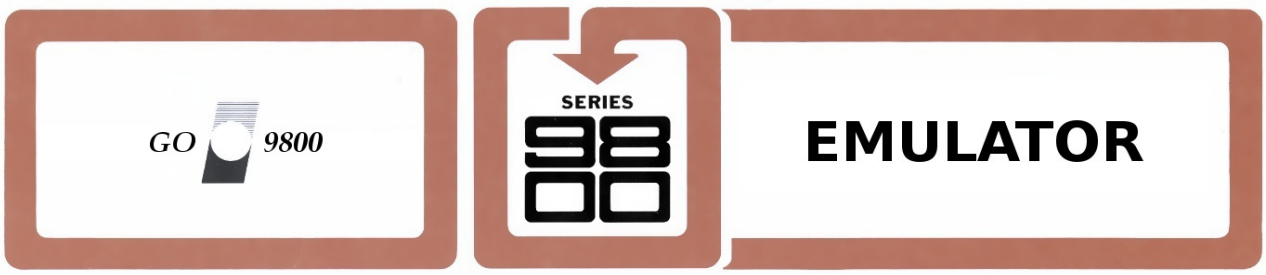


GO9800 – Series 9800 Emulator

Release 2.0

User Manual

Contents

This book is dedicated to my friend Jon Johnston, who tragically died in an accident in April 2016.

0. Preface

GO9800 is intended as a complete emulation of the hardware of the famous first family of Hewlett-Packard desktop calculators, i.e. HP9810A, HP9820A, HP9821A, and HP9830A/B.

The emulator is able to run the complete ROM-based firmware of the above models and the application programs written in the model specific programming language. Moreover the user interfaces (display, keyboard, printer), mass memory devices (magnetic card reader, tape drive, disc drive), and other external devices can be emulated to resemble the complete functionality of the original machine.

To enhance the realism of the emulation part of the outside appearance is graphically displayed and some of the mechanical sounds (beeper, cooling fan, drive motors, etc.) can be output via a sound card.

The emulator is completely written in Java 2 and platform independent.

1. General Features

The GO9800 kernel is an emulation of the CPU which was common to all calculators of the 9800-family. All CPU instructions are implemented, including I/O and floating point. The instruction set is briefly described in the U.S. patent 3,859,635. Since release 2.0 the CPU is emulated on micro-code level, which means, that decoding and execution of the instructions is done using the micro-program by the CPU itself. In prior releases the instruction set was implemented in JAVA code, based on the (incomplete) description in the mentioned patent document. The micro-code emulation is based on the ROM dumps, printed in the patent document, and also comprises the arithmetic logic unit (ALU). To achive this, many hardware elements of the original machines had to be simulated, such as all CPU shift-registers, flip-flops, decoders and last but not least the eight micro-code and ALU ROMs. The micro-code and CPU are further described in appendix D.

Part of the CPU is also the 'I/O-register and gate interface', which decodes and executes the CPU I/O-instructions.

The CPU communicates with the memory unit, consisting of read only (ROM) and read/write memory (RWM, nowadays called 'RAM'). Via the I/O-bus the IO-register is connected to all internal and external devices (display, keyboard, printer, etc.). The complete device I/O protocols are implemented, including service request (interrupt). The peripheral devices work in separate program threads, so that parallel and real time operation of the calculator mainframe and peripheral devices is possible. The machine firmware, originally residing in ROMs, was extracted from the original hardware and is executed without any modification by the CPU emulation. The ROM areas of the memory are write protected in the emulator and behave as the original. The complete memory equipment is configurable to resemble several memory expansion options. The presence of plug-in ROM blocks can also be configured and ROM blocks can be exchanged at runtime.

The emulator also contains a complete disassembler, which, when activated, disassembles each CPU instruction prior to execution, dumps the contents of the CPU registers A, B, and E, the IO register, various flags, and the pseudo registers AR1 and AR2, used for floating point operations. The disassembler can be activated either interactively or by setting breakpoints or watchpoints in the calculator memory. The

disassembler output is displayed in a separate, scrollable window.

In a special trace mode, activated either manually in the disassembler window or by a breakpoint, the cpu instructions can be executed step by step or in a 'slow motion' with variable time intervals.

Since release 2.0 the disassembler is independent of the instruction execution, which is now done by micro-code, and comprises a mode for decoding of all micro-instructions, which make up a CPU instruction. In this mode the contents of all internal CPU registers A, B, E, M, T, Q, and P as well as the state of the control flip-flops are displayed together with the micro-code.

1.1 Machine Configuration Files

The emulated machines can be customised by means of a configuration file. The configuration file defines the layout and size of the calculator memory areas, type and position of ROM modules, and external devices which are connected to the machine. For one specific calculator model there can be several configuration files for different combinations of RWM, ROM, and peripheral devices.

The configuration file is a plain text file and contains one configuration item per line. Each configuration item consists of a type, name, address or select code fields, and an internal slot identification.

The configuration file may contain comment lines, which are not evaluated. Comment lines have to start with a semicolon.

Example:

```
; This is a comment line
```

Since release 1.50 the emulator loads the Java classes for each model, peripheral device, and device interface dynamically using the Reflection API. This has the advantage that additional model and device classes may be added without changing the source code of the emulator. The names of calculator models and devices must correspond to an existing Java class. If in a configuration file a model or device name is used for which no matching Java class is found an appropriate error message is created and the emulator is terminated.

1.1.1 Peripheral Device Configuration Files

Since release 1.50 each peripheral device has its own configuration file, in which the appropriate calculator interface, the possible select codes, and the suitable calculator models are defined.

The configuration file comprises the following configuration items:

```
Model <List of suitable calculator models>
Name <Java class name for the device (usually the HP-number)>
Title <Name which is displayed in the startup log>
Interface <Java class name for the appropriate interface (usally
the HP-number)>
Selectcode <List of allowed select codes>
RWM <Optional starting address and length of extended memory>
Parameters <Optional device specific parameters>
```

The select code defined in a calculator configuration file are checked against the allowed select codes in the device configuration file.

Here is an example configuration file for the HP11305A disk controller with HP11273A interface containing 256 words of extended memory, connected to one HP9880A disk drive with two disks (Parameters 1 2), which may only be used in conjunction with the HP9830A:

```
Model HP9830A
Name HP11305A
Title MASS_MEMORY_CONTROLLER
Interface HP11273A
Selectcode 11
RWM 077000 000400
Parameters 1 2
```

1.1.2 Search Strategy for Configuration Files

Every configuration file is searched first in the current working directory (normally the directory where the emulator is started from).

If there is no customized configuration file found in this place, the emulator looks for the standard configuration file in the installation directory, where the file GO9800.jar is located.

If this is also not present, the configuration is loaded from the directory /config/ in the GO9800.jar itself.

If the configuration file not found in any of these locations the emulator is terminated.

1.1.3 Calculator Model Configuration

The first line of the configuration file must define the calculator model and an optional version. In this release the version information is used only for the HP9810A which exists in version 1 and 2. If there is only one version the version number has to be omitted.

Examples:

```
Model HP9830A
Model HP9810A 2
```

In the current release the following Java classes (calculator models) exist:

```
HP9810A
HP9810A2
HP9820A
HP9821A
HP9830A
```

The following lines define memory blocks of different type and sizes. There are two types of memory: RWM (Read-Write-Memory) and ROM (Read-Only-Memory). The areas where RWM and ROM are located are specific and fixed for each calculator model.

1.1.4 ROM configuration

The ROM areas may be divided in several sections, e.g. for build-in system ROM and

additional ROM modules. Each ROM block has a 16bit start address and block size. Both values have to be given as octal values. Each ROM block has a name which references a file which contains the ROM data. At startup of the emulator the referenced ROM files are read into the calculator memory.

Examples:

```
ROM 016000 001000 HP9830A_System8 Block16
ROM 040000 000400 HP9830A_System9 Block40
ROM 020000 002000 HP11274B Slot1
ROM 022000 002000 HP11271B Slot2
```

A system ROM block has a specific start address and can only be used at this position. The block name has the form <Calculator Model>_System#. Each memory block (ROM or RWM) needs a unique block-id. The block-id for system ROM block is arbitrary but has to be unique amongst all memory blocks, conventionally the block-id contains the octal starting address of the block. For plug-in ROMs the block-id must be in the form Slot#, where # is the position of the calculators ROM slot. In the real calculator each slot corresponds to a fixed address range (e.g. in the HP9810A Slot1 uses addresses from 002000 to 003777 octal). In the emulator one can assign a different address range although this will probably lead to malfunction or hangup of the emulation.

Add-on ROM blocks may be positioned at different start addresses although not every position is useful. E.g. the HP9810A Mathematics ROM block may be plugged in every of the three ROM slots of the calculator but will only be functional in the first slot (address 002000). The block name is identical to the HP product number of the original ROM, e.g. HP11220A.

Since release 1.40 each ROM has an additional description file which contains the valid slots and/or address positions for the particular ROM and calculator. During startup each ROM in the calculator configuration is validated against the corresponding ROM description. If a conflict is detected, the cause is logged in the command console and the emulator startup is aborted. If a ROM is later plugged in manually the same check is performed and the ROM is activated only if no conflict is detected (see chapter 1.2).

The assigned address range for ROM blocks is marked read-only for the calculator CPU, so the contents may not be changed by any program.

1.1.5 Empty ROM Slots

If any ROM slot should be left empty at startup, then the corresponding configuration items have to be set to a dummy ROM block HP11XXXX, otherwise the related memory range will not be initialized correctly. E.g. if the ROM slots 2 and 3 of the HP9810A should be empty at startup, the configuration file must contain:

```
ROM 006000 002000 HP11XXXX Slot2
ROM 010000 002000 HP11XXXX Slot3
```

The HP11XXXX dummy ROM simply contains 0 (zero) values of each memory location.
See also chapter 1.2.

1.1.6 RWM configuration

Areas of Read-Write-Memory are defined by RWM blocks. A RWM block is defined similar

to a ROM block with start address and block size (in octal values), a block name, and a slot-id. The block name is arbitrary and for information purpose only. RWM block names are typically System or HP product numbers of memory expansions. The slot-id is also arbitrary but has to be unique between all memory blocks (RWM and ROM). Memory blocks with the same block-id 'overwrite' each other in the internal configuration.

Examples:

```
RWM 001400 000400 HP9830A_System Block1
RWM 040400 007400 HP9830A_User Block40.4
RWM 050000 010000 HP11275F Block50
```

1.1.7 Peripheral Device Configuration

Peripheral devices which shall be attached to the calculator are defined by configuration items of type DEV. The name field contains the HP product number of the device (e.g. HP9862A for a plotter). The last entry of the configuration line is a optional select code. Some devices have fixed select code which can not be changed (e.g. the HP9862A has fixed select code 14). In such cases the select code entry is ignored. Other devices need a specific select code which has to be unique between all internal and external devices. Refer to the original documentation of the specific peripheral device and calculator for possible select codes.

Examples:

```
DEV HP9862A
DEV HP9865A 5
```

1.1.8 Breakpoint Configuration

The emulator contains a run-time disassembler for inspection and analysis of machine programs (appendix C). Normally the disassembler is inactive and can be enabled manually in the disassembler dialog window.

In the configuration file breakpoint addresses can be defined at machine instruction level. As soon as the CPU program counter reaches a breakpoint address the execution of the machine instructions is halted and the disassembler dialog window is automatically opened. The instruction at the breakpoint address is shown in the disassembler list but not yet executed. The execution can be continued by either using the single step mode or resuming to the normal run mode.

Breakpoint addresses have to be given as octal values from 0 to 77777. There can be a arbitrary number of breakpoints defined without significant performance drawback. Breakpoints are only evaluated when the associated memory address is accessed.

Example:

```
Breakpoint 05733
```

Since release 2.0 the CPU is emulated on micro-code level, which means, that machine instructions are decoded and executed under control of the CPU micro-programm. In a kind of endless loop the micro-program reads machine instructions from memory, decodes them and branches in the appropriate execution paths of the micro-programm. There is

only one address in the micro-program where it comes by for each new instruction, this is micro-adress 0616 (hex 6E). So breakpoints are checked every time when the micro-program reaches this address. If micro-decoding mode is anabled in the console window, the next micro-instructions that can bee seen are for decoding of the machine instruction. See appendix D for more details on micro-code.

1.1.9 Watchpoint Configuration

In the configuration file memory watchpoint addresses can be defined. A watchpoint is a memory location which is checked every time it is accessed by a CPU instruction either in read or write mode. A watchpoint can be conditional or unconditional. A conditional watchpoint is further characterised by a watch condition and a test value. The content of the watched memory location is compared with the test value by a comparison operator. If the result is true, the watchpoint condition is met. In an unconditional watchpoint the memory content is ignored.

As soon as a CPU (micro-) instruction accesses the memory location identified by the watchpoint address, the contents of this memory location is tested against the watchpoint condition. If the condition is met, the current instruction is finished, then the machine program execution is halted and the disassembler dialog window is automatically opened. The next CPU instruction is shown in the disassembler list but not yet executed. The execution can be continued by either using the single step mode or resuming to the normal run mode.

Watchpoint addresses have to be given as octal values from 0 to 77777. For conditional watchpoints a test value and a watch condition may be defined. The test value can be any unsigned 16 bit octal value between 0 and 177777. The watch condition is defined by one of the comparison operators (=, <, >).

There can be a arbitrary number of watchpoints defined without significant performance drawback. Watchpoints are only evaluated when the associated memory address is accessed.

Examples:

```
Watchpoint 01000 07777 =
Watchpoint 05600
```

Although memory access happens as part of the micro-program execution watchpoints are effective only when micro-address 0616 (hex 6E) is reached (see also chap. 1.1.8 and appendix D.2.3) and program execution is halted just before decoding of the next machine instruction.

1.1.10 Example Configurations

This is an example of a complete configuration file for the HP9830A calculator:

```
Model HP9830A
ROM 000000 001400 HP9830A_System1 Block0
ROM 002000 002000 HP9830A_System2 Block2
ROM 004000 002000 HP9830A_System3 Block4
ROM 006000 002000 HP9830A_System4 Block6
ROM 010000 002000 HP9830A_System5 Block10
```

```
ROM 012000 002000 HP9830A_System6 Block12
ROM 014000 002000 HP9830A_System7 Block14
ROM 016000 001000 HP9830A_System8 Block16
ROM 040000 000400 HP9830A_System9 Block40.0
RWM 001400 000400 HP9830A_System Block1
RWM 040400 007400 HP9830A_User Block40.4
RWM 050000 010000 HP11275F Block50
RWM 060000 016000 HP11276F Block60
ROM 017000 001000 INFOTEK_FB2 Int0
ROM 024000 002000 INFOTEK_FB1 Int1
ROM 030000 002000 HP11270B Int2
ROM 032000 002000 HP11272B Int3
ROM 036000 002000 HP11273B Slot1
ROM 034000 002000 HP11274B Slot2
ROM 020000 002000 HP11289B Slot3
ROM 022000 002000 HP11279B Slot4
ROM 026000 002000 HP11271B Slot5
DEV HP9860A
DEV HP9861A 8
DEV HP9862A
DEV HP9865A 5
DEV HP9866B 15
DEV HP9880B
DEV HP11202A 1
```

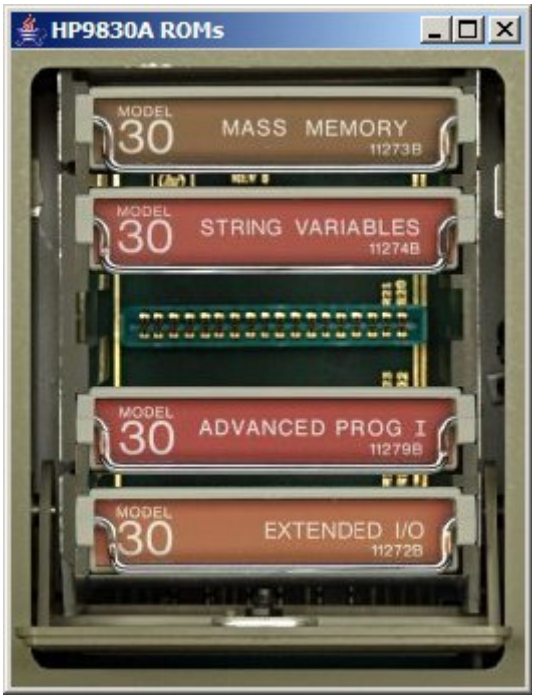
1.2 Exchanging of ROM blocks

Plug-in ROM blocks may be exchanged in all emulated calculators at run-time. To achieve this in the HP9810A or HP9820A click with the left mouse-button on the module to be exchanged (on the calculator left side above the LED display). Then a dialog window is opened in which all available plug-in ROMs are visible. Click on one of the symbolic ROMs to select it for use in the same slot as the previous. If no ROM has to be used (the slot should be empty) click on the first symbol showing the slot cover. In this case a dummy ROM block HP11XXXX with 0 (zero) byte values will be used in the address range of the slot.

Since release 1.4 the chosen ROM slot is validated against the ROM description file (see chapter 1.1.4). If the ROM is not compatible with this slot it is unloaded and the previous ROM in this slot is reloaded. This may occur with several HP9810A ROMs. E.g. The HP11211A PRINTER ALPHA ROM can only be plugged into slot 3.

ROM exchange in the HP9830A is somewhat tricky since the plug-in ROMs are covered in the left side of the calculator housing and normally not visible. One has to click somewhere in an area in the LED display, above the function keys. Then a dialog window is opened which shows the ROM slots. Now click on the ROM position to be exchanged. A Second dialog window is opened which shows the available plug-in ROMs. Click on one of the symbolic ROMs to select it for use in the same slot as the previous. If no ROM has to be used click on the first symbol showing an empty slot. The newly select ROM is now displayed in position. Choose another ROM to exchange or close the slot window by clicking on the upper right corner.

HP9830A ROM selector dialog



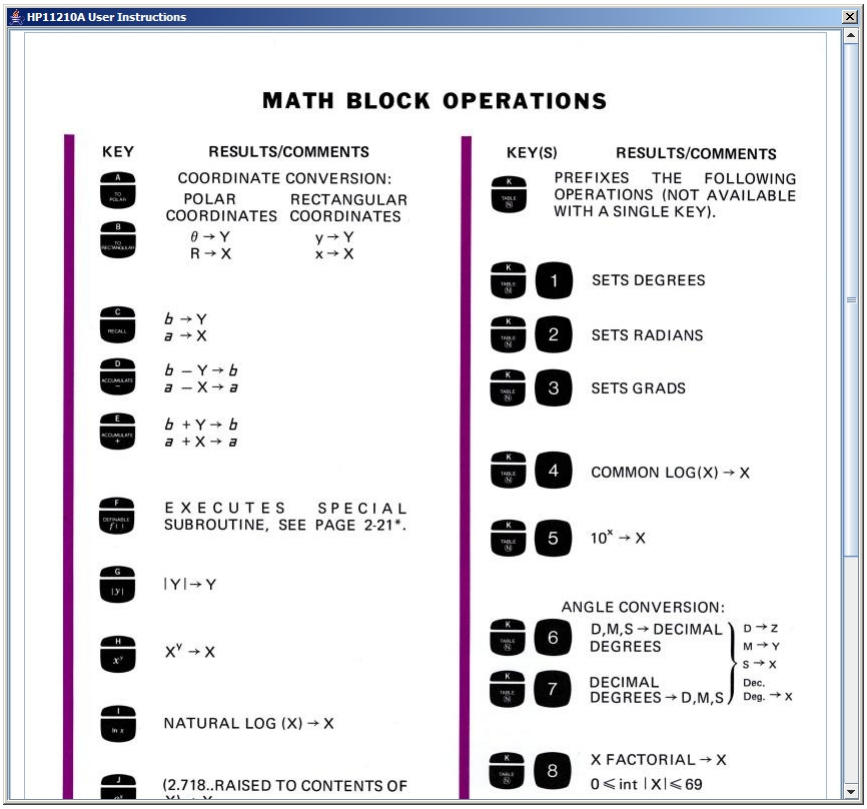
HP9830A ROM housing with one empty slot

After exchange of a ROM block the calculator should be restarted by the key combination Ctrl+Alt+R. This has the same effect as switching the real calculator off and on. For certain ROMs this procedure is necessary to prevent the calculator from erratic behaviour or hang-up.

1.3 Display of User Instructions

There is a facility to display one ore more pages of user instructions for each plug-in ROM as well as for the calculator itself. The user instructions where extracted from the original operating manuals as far as these had been available and contained instruction summary pages or error codes. If there is a keyboard overlay for a ROM (only available for the HP9810A and HP9820/21A half-keys), this overlay will also be shown enlarged.

The ROM specific user instructions may be displayed by clicking with the *right* mouse-button on the plugged ROM module. On the HP9810A, HP9820A, and HP9821A a right-click anywhere on the keyboard overlay area also shows the user instructions. On the HP9830A the ROM slot window has to be opened before the instructions can be displayed (see above ch. 1.2).



User Instructions for the HP9810A Mathematics ROM

Then a popup window opens and shows the first instructions page. If there are more pages clicking with the left mouse-button on the instruction page or pressing the space-bar loads the next one. If the last page was displayed the cycle begins again with page one.

Calculator specific user instructions for the HP9810A and HP9820A may be displayed by clicking with the right mouse-button on the handle of the top cover (above the printer and magnetic card reader). For the HP9830A right-clicking on the 'hot spot' for the ROM window does the job (see above ch. 1.2).

1.4 Keyboard Configuration Files

The assignment of keys on the host-PC keyboard to key codes of the emulated machines is controlled by a calculator specific configuration file. By that means user- or language-specific keyboard customizations can be achieved.

The name of the configuration file is

```
<Machine>-keyb.cfg
```

where <Machine> is the name of the machine configuration file. E.g. if the machine configuration file is MyHP9830A.cfg then the corresponding keyboard configuration file is MyHP9830A-keyb.cfg.

The keyboard configuration file is a plain text file and contains one key code assignment per line. Each assignment consists of the octal calculator key code, the decimal PC key code, and an optional modifier key, each separated by space or tabulator.

The octal key code may be found in the operation manual of the individual calculator or can be determined using the GO9800 Console key-log function (see appendix C.3).

The host-PC key code may be found elsewhere or also using the key-log function as described above.

The modifier key defines one to three PC-keys which have to be pressed simultaneously with the function key. The possible modifier keys are

- S = Shift
- A = Alt
- C = Control

Example: If the calculator key code is 77 (octal) and on the PC the keys Alt+Sift+X have to be pressed to activate that key, the configuration line is:

```
77   88   AS
```

The configuration file may contain comment lines, which are not evaluated. Comment lines have to start with a semicolon. Comments can also be used as last part of a assignment line.

Example:

```
; This is a comment line
```

To ease the creation of keyboard configurations for the HP9830A, most alphanumeric keys are inherited from the host PC. PC keys which are not listed in the configuration file are treated as follows:

If the corresponding octal scan code is between 40 and 172 (inclusive) it is used as the calculator key code. For compatibility with the HP9830A lower case characters are converted to upper case and upper case characters are or-ed with the shift-bit (octal 200).

If the scan code is outside this range, the key is ignored. So normally only function keys and language specific keys have to be configured.

1.4.1 Display of the Keyboard Mapping

The current mapping of PC keys to calculator keys can be temporary displayed as a

keyboard overlay by pressing Ctrl+K. For each calculator key the configured PC key combination is displayed beneath that key. If no special PC key is assigned, the calculator key code converted to the corresponding ASCII character is displayed. Modifiers Alt, Ctrl, and Shift are displayed as A+, C+, and S+ respectively, followed by the normal key.

In order to have a readable representation of special PC keys like function or cursor keys there is an editable file

`keynames.cfg`

which contains the translation of JAVA key codes to display character strings.

The configuration file of the distribution looks as follows:

```
8    Bsp
10   Ent
19   Pau
27   Esc
32   Spc
33   PgUp
34   PgDn
35   End
36   Home
37   Left
38   Up
39   Right
40   Dn
96   K0
97   K1
98   K2
99   K3
100  K4
101  K5
102  K6
103  K7
104  K8
105  K9
106  K*
107  K+
109  K-
110  K.
111  K/
112  F1
113  F2
114  F3
115  F4
116  F5
117  F6
118  F7
119  F8
120  F9
121  F10
```

```
122   F11
123   F12
127   Del
153   <
155   Ins
520   #
521   ~
```

The key map also shows the 'hot areas' for each calculator key in which it can be actuated by a mouse-click.

HP9830A with key map showing 'hot areas' and PC key combinations

1.4.2 Predefined Keys

Some functions of the emulator itself may be controlled by the PC keyboard. The key codes used for these functions are hard coded and can not be changed or used in keyboard configuration files. These functions and corresponding keys are as follows.

PC keys	Function
Ctrl + C	Open the HP2116 style control panel
Ctrl + D	Opens the disassembler console.
Ctrl + F	Toggles the sound output of the calculator cooling fan.
Ctrl + K	Toggles display of the PC to HP calculator key map.
Ctrl + P	Generates a hardcopy of the output of the build-in printer.
Shift + Ctrl + P	Opens the page format dialog for the hardcopy function.
Alt + Ctrl + R	Resets the calculator to power-on condition.
Ctrl + S	Toggles the complete sound output of the calculator on and off.
Ctrl + T	Starts and stops the CPU instruction execution timer.
Ctrl + Page↑	Moves the paper of the built-in printer one page up.
Ctrl + Page↓	Moves the paper of the built-in printer one page up.
Ctrl + Del	Clears the output of the built-in printer.
Ctrl + Home	Advances the paper of the built-in printer.

1.5 General Limitations

When executed on a modern PC hardware (typically about 3 GHz) the emulated calculator is faster than the original machine. Programs which use loops to generate time delays may not run correctly. For internal timing of the emulator the JAVA class Thread is used. On most platforms the method Thread.sleep(milliseconds) is inaccurate, esp. for small values of 1-3 ms. This may lead to slower display and printer output and other timing deviations.

Since release 2.0 the CPU is emulated on micro-code level. This results in a significant slower execution than in previous releases (up to 1.61). The speed is still about 6 to 7 times faster than the real hardware but 20 to 50 times slower than release 1.61, depending on the program code. This is due to the complete simulation of the one-bit-serial buses and micro-instructions. One may consider this slowness as contribution to a more realistic emulation.

Some emulated peripheral devices are timing-critical because they deliver a continuous data stream to the calculator. Amongst these devices are the built-in magnetic card reader, the HP9860A marked card reader, and the HP9865A cassette memory. The emulator accounts for the behaviour of the real devices where asynchronous mechanical transports determine the rate of incoming data. If for any reason the calculator is not able to receive data at this given rate, information maybe lost and the data unusable. With the HP9865A this may result in check-sum errors. So care should be taken that the host PC is not too

busy with other programs when using the peripheral devices mentioned.

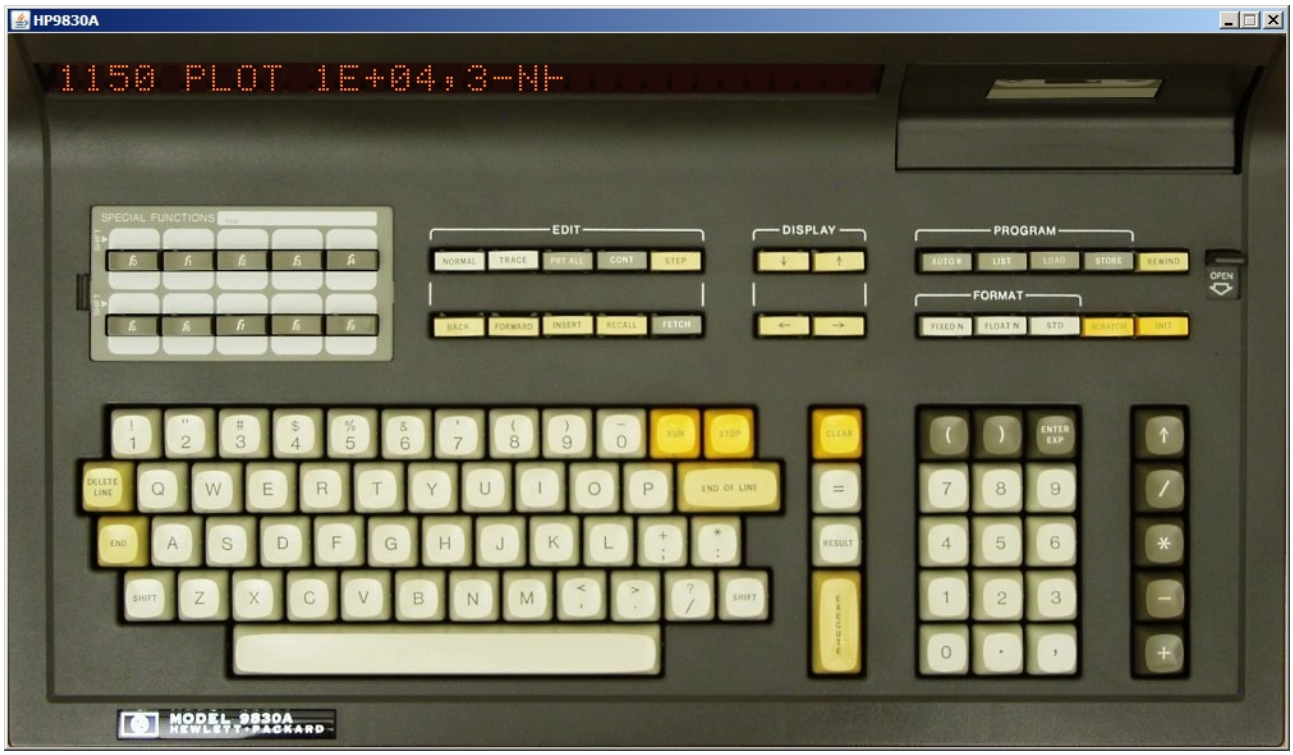
From release 1.41 there is an automatic timing calibration running at startup of the emulator. During this calibration the critical timing constants are “measured” and corrected if necessary. As a result the timing of peripheral devices should be more accurate and platform independent.

On some platforms, esp. Linux-Systems, the sound output maybe distorted or may exhibit some crackling. The is due to characteristics of the Java sound engine and some sound cards. If this is perturbing the sound may be completely switched off using the key combination Ctrl+S.

2. Implemented Machines

The HP9800 emulator is able to run with different personalities which resemble the individual hardware features of the real models. In this release the personalities of the HP9810A, HP9820A, HP9821A, and HP9830A are implemented. The HP9810A is available in two versions: version one with single-digit LED display modules, and version two with five-digit LED modules. Which calculator personality is used by emulator is defined in the Model item of the configuration file. The configuration file is given the the emulator as start parameter.

2.1 HP9830A Personality



2.1.1 Implemented Features

2.1.1.1 ROM

For the HP9830A the following system ROM blocks must be defined in the configuration file:

HP9830A_System1 to HP9830A_System9

The following add-on ROMs are optional:

HP11270B Matrix Operations
HP11271B Plotter Control
HP11272B Extended I/O
HP11273B Mass Memory
HP11274B String Variables

HP11277B Terminal I
HP11278B Batch Basic
HP11279B Advanced Programming I
HP11283B Printer Control
HP11289B Advanced Programming II
HP11296B Data Comm. I
Infotek Fast Basic I
Infotek Fast Basic II
Infotek Fast Basic III
Infotek Fast Basic IV

2.1.1.2 RWM

The minimal RWM blocks which have to be configured are:

HP9830A_System
HP9830A_User (3840 words)

The following memory expansions are optional:

HP11275F (additional 4096 words)
HP11276F (additional 8192 words)

2.1.1.3 I/O Devices

The following internal devices are implemented:

Display
Sound (for output of BEEP command and error messages)
Keyboard (select code 13)
Tape drive (select code 10)

The following external devices can be configured:

HP9860A Marked Card Reader (no select code)
HP9861A Typewriter (select code 1 to 14)
HP9862A Plotter (fixed select code 14)
HP9865A Cassette Tape Drive (select code 1 to 9)
HP9866A/B Thermoelectric Line Printer (fixed select code 15)
HP9880B Mass Memory (fixed select code 11)
HP11202A Interface for file I/O on host PC (select code 1 to 9)

2.1.2 Limitations

When executed on a modern PC hardware (some GHz) the emulated HP9830A is much faster than the original machine (typically 30 times). Programs which use loops to generate time delays may not run correctly. Instead the WAIT instruction generates a correct delay in milliseconds. See also General Limitations (chapter 1.5).

Special attention should be payed when a HP9880A/B Mass Memory is connected to the HP9830A. Refer to chapter 3.6.4 for futher informations.

2.1.3 User Interfaces

2.1.3.1 Display

The original 5*7 dot matrix LED display is simulated. In the original machine the display is multiplexed completely by the CPU. The calculator controls the display by the signal DEN (Display Enable). When the calculator is busy, DEN is false and the display is dark.

In the emulator the display multiplexing would lead to flickering and high load of the host PCs CPU. Therefore the display content is buffered and refreshed only if it has changed. When DEN is false for a period of more than 200ms (which means that the calculator is busy with other things) the emulated display will get dark.

The output timing is nearly like the original, visible delays in display output are intended.

2.1.3.2 Keyboard

The representation of the HP9830 keyboard on the host PC keyboard can be configured. The configuration is stored in the text file HP9830A-keyb.cfg and can be changed with any text editor. More on keyboard configuration can be found in chapter 1.4.

Besides the predefined emulator keyboard functions (see chapter 1.4.2) the distributed HP9830A keyboard configuration is as follows.

Main keyboard, unshifted:

PC Key	**HP9830 Key**
A - Z	A - Z
0 - 9	0 - 9
space	space
symbols (+-*, ...)	symbols (+-*, ...)
backspace	END OF LINE
return	EXECUTE
insert	INSERT
delete	CLEAR
home	RESULT
end	END
page ↑	↑ (power-of operator)
pause	STOP
← ↑ → ↓	DISPLAY ← ↑ → ↓
F1-F9	f1-f9
F10	f0

Main keyboard, shifted:

PC Key	HP9830 Key
A - Z	a - z (displayed as A - Z)
0	=
1	!
2	"
3	
4	$
5	%
6	&
7	/
8	(
9	)
symbols (+-*, ...)	symbols (+-*, ...)
del	DELETE LINE
F1-F9	f11-f19
F10	f10

Numpad keys:

PC Key	HP9830 Key
0 - 9	0 - 9
+ - * /	+ - * /
enter	EXECUTE

Alt + Key:

PC Key	HP9830 Key
←	BACK
→	FORWARD
↓	STEP
↑	CONT
return	RUN
backspace	RECALL
del	SCRATCH
A	AUTO#
D	STD
F	FLOAT

PC Key	HP9830 Key
I	LIST
L	LOAD
N	NORMAL
P	PRT ALL
S	STORE
T	TRACE
X	FIXED

2.1.4 Internal Devices

2.1.4.1 Tape Drive

The internal tape drive is functional identical to the HP9865A external tape drive, so the emulator uses the same Java classes for internal and external drives. See the description of the HP9865A (chapter 3.4) for more detailed information.

The original tape cassettes are emulated by files in the host file system. To prepare a new host tape-file one needs a 'blank' tape. A original blank tape is not really empty, but contains at least a Begin-Of-File marker (hex 3C + control bit). Without that BOF no new files can be MARKed on that tape. In the GO9800 distribution there is file named 'blank.tape' which should be kept as origin for new tapes. To create new tape make a copy of the file 'blank.tape' and give it a name of your choice. For convention it should have the extension .tape.

To load a tape in the HP9830A emulation, mouse-click on the 'OPEN' door lever. The cassette door will be opened and a file selector dialog is displayed in which you can select your new empty tape. If you dismiss this dialog with the Cancel button, the cassette door stays open. After successful selection of a tape the door will be closed again and a loaded cassette is visible in the door window.

Now the tape is ready for the usual HP9830 tape commands, like MARK, STORE, and LOAD. See the original HP9830A manual for reference.

2.2 HP9820A Personality

2.2.1 Implemented Features

2.2.1.1 ROM

For the HP9820A the following system ROM blocks must be defined in the configuration file:

HP9820A_System1 to HP9820A_System6

The following add-on ROMs are optional:

HP11221A Mathematics
HP11222A User Defined Functions
HP11223A Cassette Memory / Special Programs
HP11224A Peripheral Control I

2.2.1.2 RWM

The minimal RWM blocks which have to be configured are:

HP9820A_System
HP9820A_User (768 words = 192 registers)

The following memory expansions are optional:

HP11228A (additional 1024 words = 256 registers)

2.2.1.3 I/O Devices

The following internal devices are implemented:

Display
Magnetic Card Reader
Printer
Keyboard (select code 13)

The following external devices can be configured:

HP9860A Marked Card Reader (no select code)
HP9861A Typewriter (fixed select code 15)
HP9862A Plotter (fixed select code 14)
HP9865A Cassette Tape Drive (select code 1 to 9)
HP9866A/B Thermoelectric Line Printer (fixed select code 15)
HP11202A Interface for file I/O on host PC (select code 1 to 9)

2.2.2 Limitations

When executed on a modern PC hardware (some GHz) the emulated HP9820 is much faster than the original machine (typically 30 times). Programs which use loops to generate time delays may not run correctly. See also general limitations (chapter 1.5).

2.2.3 User Interfaces

2.2.3.1 Display

The original 5*7 dot matrix LED display is simulated. In the original machine the display is multiplexed completely by the CPU. The calculator controls the display by the signal DEN (Display Enable). When the calculator is busy, DEN is false and the display is dark.

In the emulator the display multiplexing would lead to flickering and high load of the host PCs CPU. Therefore the display content is buffered and refreshed only if it has changed. When DEN is false for a period of more than 200ms (which means that the calculator is busy with other things) the emulated display will get dark.

2.2.3.2 Keyboard

The representation of the HP9820 keyboard on the host PC keyboard can be configured. The configuration is stored in the text file HP9820A-keyb.cfg and can be changed with any text editor. More on keyboard configuration can be found in chapter 1.4.

Besides the predefined emulator keyboard functions (see chapter 1.4.2) the distributed HP9820A keyboard configuration is as follows.

Main keyboard:

PC Key	HP9820 Key
A - C	A - C
D - W	D - W (half keys)
X - Z	X - Z
0 - 9	0 - 9
$ % & ' ?	$ % & ' ?
,	,
;	;
~	√x
#	≠
=	=
<	≤
>	>
^	GOTO SUB
backspace	STORE
space	SPACE

Numpad keys:

PC Key	HP9820 Key
0 - 9	0 - 9
+ - * /	+ - * /
.	.
enter	EXECUTE

Special and Function Keys:

PC Key	HP9820 Key
insert	INSERT
delete	DELETE
end	END
backspace	STORE
pause	STOP
←	BACK
→	FORWARD

PC Key	HP9820 Key
F1	NORMAL
F2	TRACE
F3	FIXED
F4	FLOAT
F5	ENTER
F6	DISPLAY
F7	PRINT
F8	SPACE
F9	LIST
F10	LOAD
F11	RECORD
F12	

Alt + Key:

PC Key	HP9820 Key
C	SET / CLEAR FLAG N
E	END
F	FLAG N
G	GO TO
R	R()
T	RETURN
U	GO TO SUB
X	ENTER EXP
<	≠
→	→
return	RUN PROGRAM
backspace	RECALL
del	CLEAR

Alt + Shift + Key:

PC Key	HP9820 Key
del	ERASE

2.2.4 Internal Devices

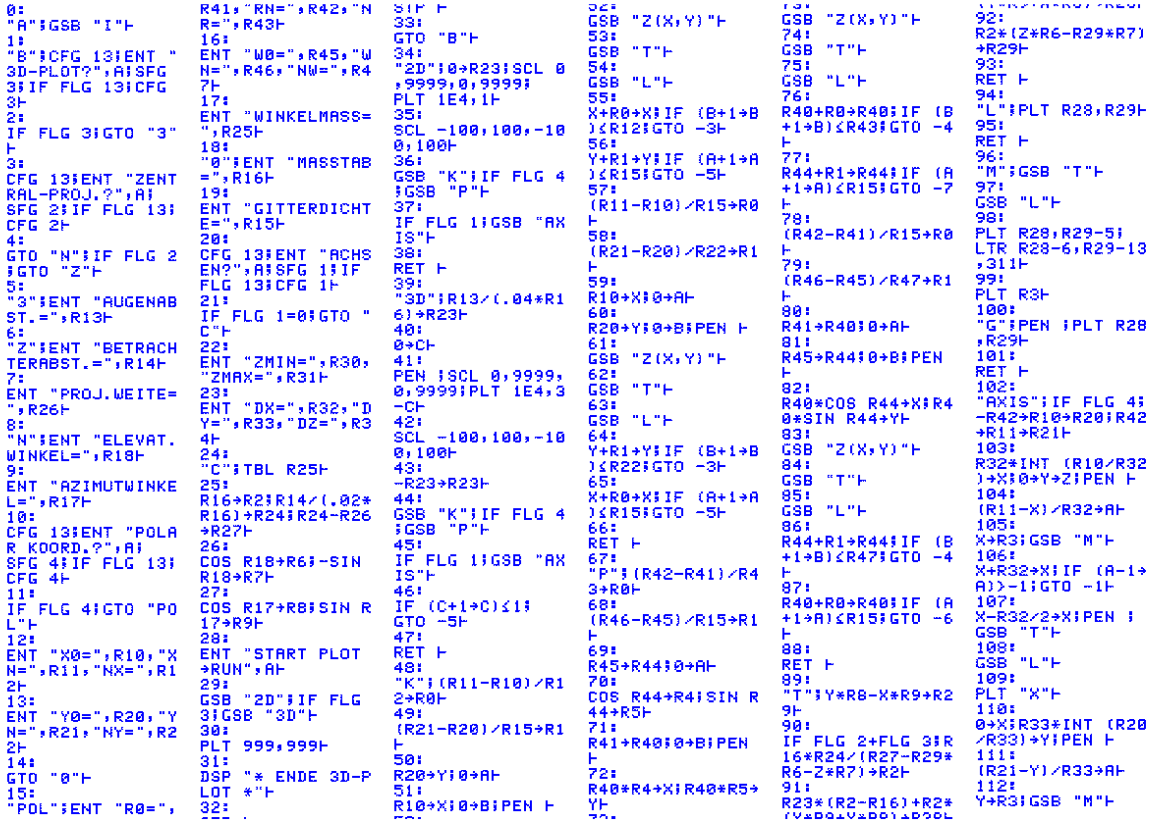
2.2.4.1 Printer

The HP9820A was equipped with a 16 character wide thermo electric printer. Use of the printer is identical to the original. The printer output is graphically emulated and the graphics are internally buffered. The 'paper' advances to the top of the displayed window area. Since the HP9820 generates the output as printer dots, not as characters, it is impossible to store it in a text file.

The output can be scrolled back and forth with the PC keys Ctrl+Page↑ and Ctrl+Page↓. The complete output can be erased by Ctrl+Del.

The PAPER advance key can be operated by mouse click or Ctrl+Home keys. Click on the PAPER key and hold the mouse button down to achieve a continous paper feed.

Since release 1.50 it is possible to send a the printer output to an arbitrary printer of the host PC. Pressing the PC keys Shift+Ctrl+P opens the standard page-setup dialog of the host system. Pressing Ctrl+P opens the standard print dialog of the host system, where the printer may be selected and the hardcopy started. The output is automatically arranged in multiple columns per page, the number depends of the page size and orientation.



Example of a printer hardcopy

2.2.4.2 Magnetic Card Reader

Like the HP9865A the internal card reader emulates the original magnetic media by files in the host file system. To prepare a new host card-file one needs a 'blank' card. This can be any simple empty (0 byte length) file, e.g. created with a text editor. Give the file a name of your choice. For convention it should have the extension .mcard.

To write a program in HP9820 main memory to a magnetic card simply click on the RECORD key. Then the display goes out and the sound of the transport motor is audible. A file selector dialog is displayed in which you can select your card file. If you dismiss this dialog with the Cancel button, the motor keeps running and has to be stopped with the STOP key. After successful selection of a card the program will be stored in the card file. As in the original every previously stored contents will be overwritten. Finally the motor sound stops.

Unlike the original a program of any size can be stored on one single card.

Reading of a program stored in a magnetic card is straight forward. Also writing and reading of data registers. See the original HP9820A manual for reference.

2.3 HP9821A Personality

The HP9821A is an advancement of the HP9820A and is identical to it in the most functions. The main differences are:

- A cassette tape drive instead of a magnetic card reader
- Extended memory (up to 1400 registers)
- Different design of the chassis

Most user programs for the HP9820A will also run on the HP9821A.

2.3.1 Implemented Features

2.3.1.1 ROM

For the HP9821A the following system ROM blocks must be defined in the configuration file:

HP9821A_System1 to HP9821A_System6

The following add-on ROMs are optional:

HP11221A Mathematics
HP11222A User Defined Functions
HP11224A Peripheral Control I

The HP11223A Cassette Memory / Special Programs ROM can not be used with the

HP9821A since the functionality is already contained in the system ROM.

2.3.1.2 RWM

The minimal RWM blocks which have to be configured are:

HP9821A_System
HP9821A_User (1792 words = 448 registers)

Up to two of the following memory expansions are optional:

HP11255A (additional 2048 words = 512 registers)

2.3.1.3 I/O Devices

The following internal devices are implemented:

Display
Printer
Sound (for output of ♦BEL command and error messages)
Keyboard (select code 13)
Cassette Tape Drive (select code 10)

The following external devices can be configured:

HP9860A Marked Card Reader (no select code)
HP9861A Typewriter (fixed select code 15)
HP9862A Plotter (fixed select code 14)
HP9865A Cassette Tape Drive (select code 1 to 9)
HP9866A/B Thermoelectric Line Printer (fixed select code 15)
HP11202A Interface for file I/O on host PC (select code 1 to 9)

2.3.2 Limitations

When executed on a modern PC hardware (some GHz) the emulated HP9821 is much faster than the original machine (typically 30 times). Programs which use loops to generate time delays may not run correctly. See also general limitations (chapter 1.5).

2.3.3 User Interfaces

2.3.3.1 Display

The original 5*7 dot matrix LED display is simulated. In the original machine the display is multiplexed completely by the CPU. The calculator controls the display by the signal DEN (Display Enable). When the calculator is busy, DEN is false and the display is dark.

In the emulator the display multiplexing would lead to flickering and high load of the host PCs CPU. Therefore the display content is buffered and refreshed only if it has changed. When DEN is false for a period of more than 200ms (which means that the calculator is busy with other things) the emulated display will get dark.

2.3.3.2 Keyboard

The representation of the HP9821 keyboard on the host PC keyboard can be configured. The configuration is stored in the text file HP9821A-keyb.cfg and can be changed with any text editor. More on keyboard configuration can be found in chapter 1.4.

The distributed PC keyboard configuration is the same as for the HP9820A.

2.3.4 Internal Devices

2.3.4.1 Printer

The HP9821A was equipped with a 16 character wide thermo electric printer. Use of the printer is identical to the original. The printer output is graphically emulated and the graphics are internally buffered. The 'paper' advances to the top of the displayed window area. Since the HP9821A generates the output as printer dots, not as characters, it is impossible to store it in a text file.

The output can be scrolled back and forth with the PC keys Ctrl+Page↑ and Ctrl+Page↓. The complete output can be erased by Ctrl+Del.

The PAPER advance key can be operated by mouse click or Ctrl+Home keys. Click on the PAPER key and hold the mouse button down to achieve a continuous paper feed.

Since release 1.50 it is possible to send a the printer output to an arbitrary printer of the host PC. Pressing the PC keys Shift+Ctrl+P opens the standard page-setup dialog of the host system. Pressing Ctrl+P opens the standard print dialog of the host system, where the printer may be selected and the hardcopy started. The output is automatically arranged in multiple columns per page, the number depends of the page size and orientation.

2.3.4.2 Tape Drive

The internal tape drive is functional identical to the HP9865A external tape drive, so the emulator uses the same Java classes for internal and external drives. See the description of the HP9865A (chapter 3.4) for more detailed information.

The original tape cassettes are emulated by files in the host file system. To prepare a new host tape-file one needs a 'blank' tape. A original blank tape is not really empty, but contains at least a Begin-Of-File marker (hex 3C + control bit). Without that BOF no new files can be MARKed on that tape. In the GO9800 distribution there is file named 'blank.tape' which should be kept as origin for new tapes. To create new tape make a copy of the file 'blank.tape' and give it a name of your choice. For convention it should have the extension .tape.

To load a tape in the HP9821A emulation, mouse-click on the 'OPEN' key. A file selector dialog is displayed in which you can select your new empty tape. After successful selection of a tape a loaded cassette is visible in the door window.

Now the tape is ready for the usual HP9821A tape commands, like MRK, STF, and LDF. See the original HP9821A manual for reference.

Most tape functions and some special functions are obtained by pressing the prefix key ♦ followed by another key. On the original HP9821A these secondary functions are printed on the front side of the key caps but are not visible in the emulator. For the ease of use here is a list of the secondary functions and their corresponding key sequences.

Function	Mnemonic	Key Sequence
Load File	LDF	LDF
Record File	RCF	♦ LDF
Find File	FDF	♦ >
Rewind	REW	♦ JUMP
Back Space	BKS	♦ ≤
Mark	MRK	♦ IF
Set Select Code	SSC	♦ =
Identify File	IDF	♦ FLAG N
Call Special Program	CSP	♦ ≠
Initialize Special Program	ISP	♦ SET|CLEAR FLAG N
Bell	BEL	♦ SPACE N

2.4 HP9810A Personality

2.4.1 Implemented Features

The HP9810 comes in two slightly different versions. The first version was produced between 1971 and (perhaps) 1973 and used a single LED display module for each digit. The second version used the same 5-digit modules (HP1990-7405 or HP5082-7405) as the classic pocket calculators, like the HP-35. The digits of this display are significantly smaller then those of the first version. Also the second version has a bug fix in the system ROM affecting the square root function[1].

HP9810A version 1 real display

HP9810A version 1 emulator display

1 In the first version the square root of arguments near to squares of integers, e.g. √36.0000000005, causes an arbitrary change of the exponent of the Z-register.

HP9810A version 2 real display

HP9810A version 2 emulator display

Which version is emulated is controlled by the configuration file (chapter 1.1). To emulate the earlier version the first configuration item has to be

```
Model HP9810A 1
```

To emulate the second version the first configuration item has to be

```
Model HP9810A 2
```

2.4.1.1 ROM

For the HP9810A the following system ROM blocks must be defined in the configuration file:

HP9810A_System1 to HP9810A_System3

Alternatively the never system ROMs of the second version can be used:

HP9810A2_System1 to HP9810A2_System3

The following add-on ROMs are optional:

HP11210A Mathematics
HP11211A Printer Alpha
HP11213A User Definable Functions
HP11214A Statistics
HP11215A Plotter
HP11252A Peripheral Control II
HP11261A Plotter / Printer Alpha Combo
HP11262A Peripheral Control / Cassette Memory Combo
HP11266A Peripheral Control / Printer Alpha Combo
HP11267A Typewriter / Cassette Memory Combo

Note 1: The Printer Alpha ROM (HP11211A) and all combo ROMs with this can only be used in ROM slot 3. Use in another slot will lead to erratic output of the printer.

Note 2: The ROMs which require the left block of the so called half-keys and a corresponding keyboard overlay will work in ROM slot 1 only.

2.4.1.2 RWM

The minimal RWM blocks which have to be configured are:

HP9810A_Data (109 registers)
HP9810A_Program (500 program steps)

The following memory expansions are optional:

HP11217A (512 additional program steps)
HP11218A (1024 additional program steps)

2.4.1.3 I/O Devices

The following internal devices are implemented:

Display
Magnetic Card Reader
Printer
Keyboard

The following external devices can be configured:

HP9860A Marked Card Reader (no select code)
HP9861A Typewriter (select code 15)
HP9862A Plotter (fixed select code 14)
HP9865A Cassette Tape Drive (select code 1 to 9)
HP9866A/B Thermal Line Printer (select code 15)
HP11202A Interface for file I/O on host PC (select code 1 to 9)

2.4.2 Limitations

When executed on a modern PC hardware (some GHz) the emulated HP9810 is much faster than the original machine (typically 30 times). Programs which use loops to generate time delays may not run correctly. See also general limitations (chapter 1.5).

2.4.3 User Interfaces

2.4.3.1 Display

The original 7 segment LED display is simulated. There are two different versions implemented: the older version 1 uses single digit display modules, the newer version 2 5-digit modules. In the original machine the display is multiplexed completely by the CPU. The calculator controls the display by the signal DEN (Display Enable). When the calculator is busy, DEN is false and the display is dark.

In the emulator the display multiplexing would lead to flickering and high load of the host PCs CPU. Therefore the display content is buffered and refreshed only if it has changed. When DEN is false for a period of more than 200ms (which means that the calculator is busy with other things) the emulated display will get dark.

2.4.3.2 Keyboard

The representation of the HP9810 keyboard on the host PC keyboard can be configured. The configuration is stored in the text file HP9810A-keyb.cfg and can be changed with any text editor. More on keyboard configuration can be found in chapter 1.4.

Besides the predefined emulator keyboard functions (see chapter 1.4.2) the distributed HP9810A keyboard configuration is as follows.

Main keyboard:

PC Key	HP9810 Key
A - O	A - O (half keys)
P	π
Q	b
R	a
S	y→()
T	x→()
U	1/x
V	int x
W	INDIRECT
X	y←→()
Y	x←()
Z	x^2
0 - 9	0 - 9
-	CHG SIGN
<	IF x<y
>	IF x>y
=	IF x=y
#	ENTER EXP
~	√x
^	SUB/RETURN
backspace	CLEAR x
return	CONTINUE

Numpad keys:

PC Key	HP9810 Key
0 - 9	0 - 9
+ - * /	+ - * /
enter	CONTINUE

Special and Function Keys:

PC Key	HP9810 Key
del	CLEAR
page ↑	ROLL↑
page ↓	x←→y
pause	STOP
↑	↑
↓	↓
←	BACK STEP
→	STEP PRGM
F5	FLOAT
F6	FIX
F7	RUN
F8	PRGM
F9	KEY LOG
F10	LIST
F11	LOAD
F12	RECORD

Alt + Key:

PC Key	HP9810 Key
A	a
B	b
E	END
F	FORMAT
G	GOTO
I	IF FLAG
L	LABEL
P	PRINT
R	SUB/RETURN
S	SET FLAG
U	PAUSE
X	x→()
Y	y→()

2.4.4 Internal Devices

2.4.4.1 Printer

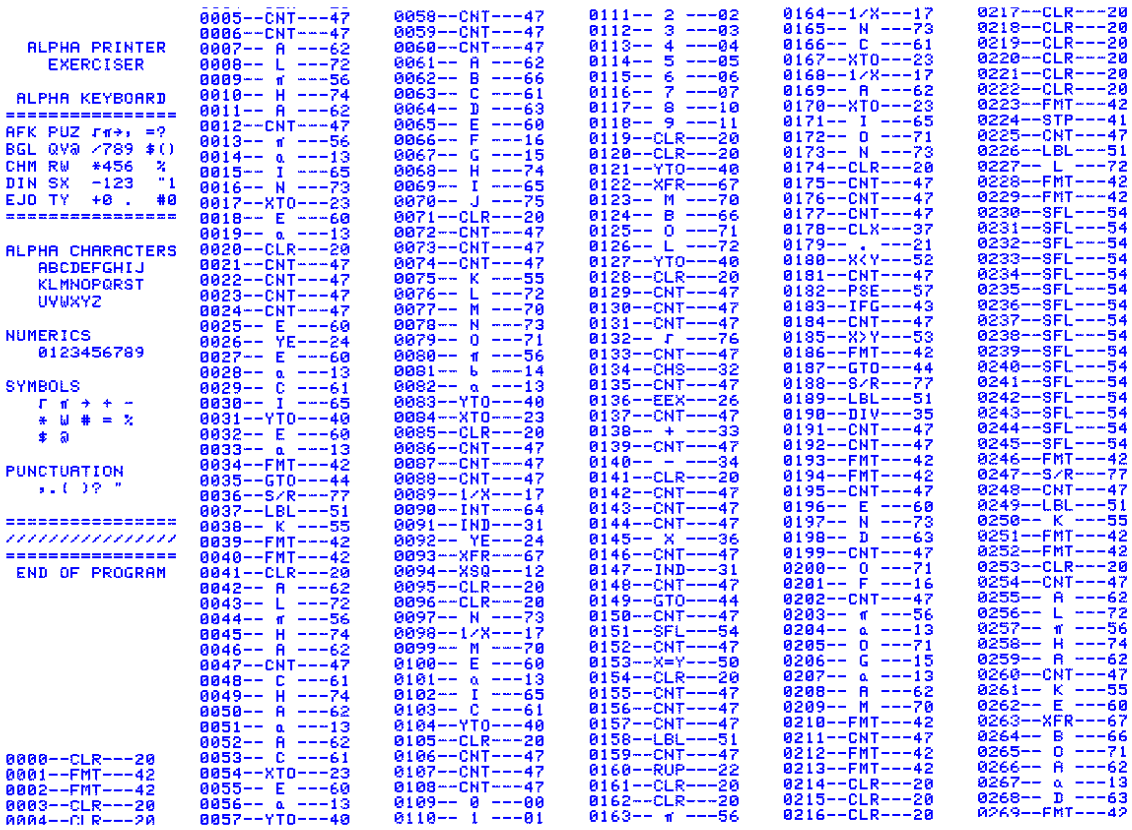
For the HP9810A the thermo electric printer was optional (Option 004). The emulated HP9810A is equipped with this printer. Use of the printer is identical to the original. The printer output is graphically emulated and the graphics are internally buffered. The 'paper' advances to the top of the displayed window area. Since the 9810 generates the output as printer dots, not as characters, it is impossible to store it in a text file.

The output can be scrolled back and forth with the PC keys Ctrl+Page↑ and Ctrl+↓. The complete output can be erased by Ctrl+Del.

The PAPER advance key can be operated by mouse click or Ctrl+Home keys. Click on the PAPER key and hold the mouse button down to achieve a continuous paper feed.

HP9810A sample printer output

Since release 1.50 it is possible to send a the printer output to an arbitrary printer of the host PC. Pressing the PC keys Shift+Ctrl+P opens the standard page-setup dialog of the host system. Pressing Ctrl+P opens the standard print dialog of the host system, where the printer may be selected and the hardcopy started. The output is automatically arranged in multiple columns per page, the number depends of the page size and orientation.



Example of a printer hardcopy

2.4.4.2 Magnetic Card Reader

Like the HP9865A the internal card reader emulates the original magnetic media by files in the host file system. To prepare a new host card-file one needs a 'blank' card. This can be any simple empty (0 byte length) file, e.g. created with a text editor. Give the file a name of your choice. For convention it should have the extension .mcard.

To write a program in the 9810 memory to a magnetic card simply click on the RECORD key. Then the LED 'INSERT CARD' lights and the sound of the transport motor is audible. A file selector dialog is displayed in which you can select your card file. If you dismiss this dialog with the Cancel button, the motor keeps running and has to be stopped with the STOP key. After successful selection of a card the program will be stored in the card file. As in the original every previously stored contents will be overwritten. Finally the 'INSERT CARD' LED goes out and the motor sound stops.

Unlike the original a program of any size can be stored on one single card.

Reading of a program stored in a magnetic card is straight forward. Also writing and reading of data registers. See the original HP9810A manual for reference.

2.4.5 Notes on use of the HP9865A tape drive

In the meantime the original manual for the Cassette Memory ROM became available to the author. In this manual the functions and FMT-key sequences for the HP9810A are

described in detail. A copy of the manual is available from the author on request.

The functions and FMT key sequences are also described in the ROM instructions page implemented in the emulator (see chapter 1.3).

3. External Devices

3.1 HP9860A Marked Card Reader

3.1.1 General Information

The HP9860A Marked Card Reader is an input device for programs and data. In fact it is a keyboard-like device which reads octal key codes from optical markings on a paper card and sends them to the calculator (using interrupts). For the original device there where cards with 30 and 50 lines (key codes) available, so one needed many cards to store a long program. The codes had to be marked by hand with a pencil.

In GO9800 the cards are emulated by a single file in the host PC file system. The card-file has to be written by hand using a simple text editor. The HP9860A can be used in conjunction with all calculators models. With the HP9810A and HP9820A the key codes have to be entered as octal characters, one key code per text line. The first line of the card file has to be 'OCT', e.g.

```
OCT
44
01
00
00
46
```

The octal key codes can be found in the original calculator manuals. For convention the octal card files should have the extension .oct.

In addition to the normal calculator key codes there is a pseudo code 200 (octal) for the Skip marker. On the original cards there was a special skip-column which caused the whole character to be ignored (skipped). This feature could be used to insert a pause in the reading process to allow for completion of somewhat lengthy operations (like STORE on HP9820). This feature is also available on the emulated HP9860A with one difference: the pseudo code 200 creates a wait time of 300ms instead of 30ms like in the original. When skip is combined with the key code 177 (octal) the pseudo key code 377 may be used as end-of-file marker. As on the original all codes behind skip+177 are ignored.

For the HP9830 key codes may also be octal coded but the emulated 9860 has a special 'convenience mode' which accepts BASIC programs stored as normal ASCII text. Such a program file has to begin with the first line 'BAS', e.g.

```
BAS
10 FOR I=1 TO 10
20 DISP I
30 NEXT I
40 END
```

For convention the card files containing BASIC programs should have the extension .bas.

3.1.1.1 Calculator Configuration

The HP9860A has no specific select code and the configuration item is:

```
DEV HP9860A
```

3.1.1.2 Device Configuration File

The device configuration file for the HP9860A is:

```
Model HP9810A HP9820A HP9821A HP9830A
Name HP9860A
Title MARKED_CARD_READER
Interface HP11200A
Selectcode 12
```

3.1.2 Usage of the HP9860A

In GO9800 the marked card reader has its own window and runs in a parallel program thread.

To read a card-file into the calculator bring the HP9860A to the foreground and mouse-click on the card input slot (the silver metallic area below the type label). Alternatively the Return or Enter key on the host PC may be pressed. A file selector dialog is displayed in which you can select the prepared card file. After successful selection of a file a marked card is displayed in the reader output area and the transport motor sound is audible. After the complete reading of the card-file the motor sound stops and the marked card is removed from the display window. The reading process may be stopped at any time by mouse-click on the marked card in the output area or by pressing Pause on the host PC keyboard.

Note: Pressing STOP on the calculator doesn't affect the reader operation but may lead to loss of data.

3.1.3 Reading of HP9810A programs

When reading programs into the HP9810A one has to change to PRGM mode before reading the card file. Alternatively one can store the mode switch in the beginning of the card file, e.g.

```
OCT
107
46
106
```

This sequence executes RUN, END, PRGM before reading the program.

3.1.4 Reading of HP9820/21A programs

HP9820A programs have to be octal coded. The octal key codes may be found in the Appendix II of the original HP9820A manual. To avoid loss of characters each STORE command (code 002) has to followed by one or more Skips (code 200) to give the calculator time to execute the STORE.

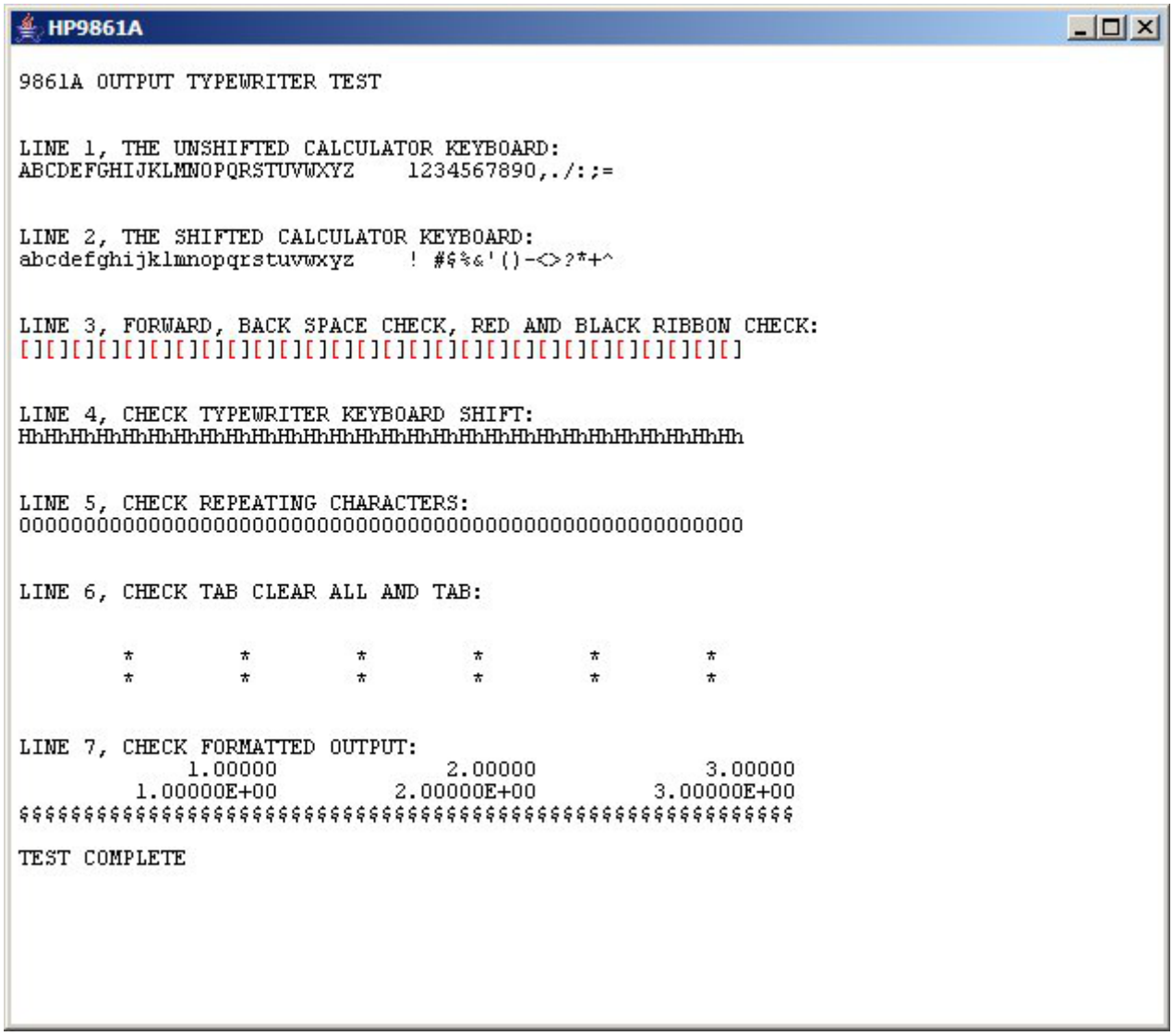
3.1.5 Reading of HP9830A programs

HP9830A programs can be BASIC files coded in ASCII. The emulated HP9860A automatically inserts a 300ms pause after each program line to allow for execution of the END OF LINE key.

3.2 HP9861A Output Typewriter

3.2.1 General Information

The original HP9861A was a ASCII text impact printer based on a Facit 3841 electric typewriter. The maximum line width was 162 characters. For use with the HP9810A the Typewriter or Peripheral Control ROM is necessary. The Peripheral Control ROM is also required for use with the HP9820/21A.



HP9861A output of HP9830A printer test program. Note the red ribbon color below LINE 3.

3.2.1.1 Calculator Configuration

In the configuration file the select code has to be set to an unused value between 1 and 15. The standard value which is assumed in the HP9810A and HP9820A ROMs is 15:

```
DEV HP9861A 15
```

For use with the HP9810A or HP9820A an additional Typewriter or Peripheral Control ROM is necessary and has to be listed in the configuration file.

3.2.1.2 Device Configuration File

The device configuration file for the HP9861A is:

```
Model HP9810A HP9820A HP9821A HP9830A
Name HP9861A
Title TYPEWRITER
Interface HP11201A
Selectcode 1 2 3 4 5 6 7 8 9 15
```

3.2.2 Usage of the HP9861A

The typewriter is controlled by the HP9810A using the FORMAT key and by the HP9820/21A using the TYPE key. Refer to the original manuals of the Typewriter and Peripheral Control ROMs for further informations.

On the HP9830A the typewriter is controlled by the WRITE command. Also the PRINT # and LIST # commands with the proper select code may be used.

The HP9861A uses its own window for output. For text output the JAVA Monospaced font is used, so some characters may look different from the original. Red and black ribbon colors are supported as well as backspace / overtype and tabulator functions. The font size may be increased and decreased in integer steps between 8 and 40. The output is automatically scrolled line by line in the window and is internally buffered. The window may be resized manually.

The buffered output maybe controlled by the following PC keys:

PC key	**Function**
page ↑	Scroll output one window page up
page ↓	Scroll output one window page down
home	Position to first (top) window page
end	Position to last window page
delete	Delete complete output
shift + delete	Delete complete output, reset to default font and window size
insert	Generates a hardcopy of the output
shift + insert	Opens the page format dialog for the hardcopy function.
+	Increase font size (max. 40)
-	Decrease font size (min. 8)
S	Toggle high-speed mode

It is possible to send a the output to an arbitrary printer of the host PC. Pressing the PC keys Shift+Ctrl+P opens the standard page-setup dialog of the host system. Pressing Ctrl+P opens the standard print dialog of the host system, where the printer may be selected and the hardcopy started. The hardcopy is automatically scaled to fill the available page size.

The emulated HP9861A has a special high-speed output mode, where all artificial delays are eliminated and the print output is generated at the maximum possible speed. In normal output mode the delay resembles the timing of the real device. In high-speed mode the

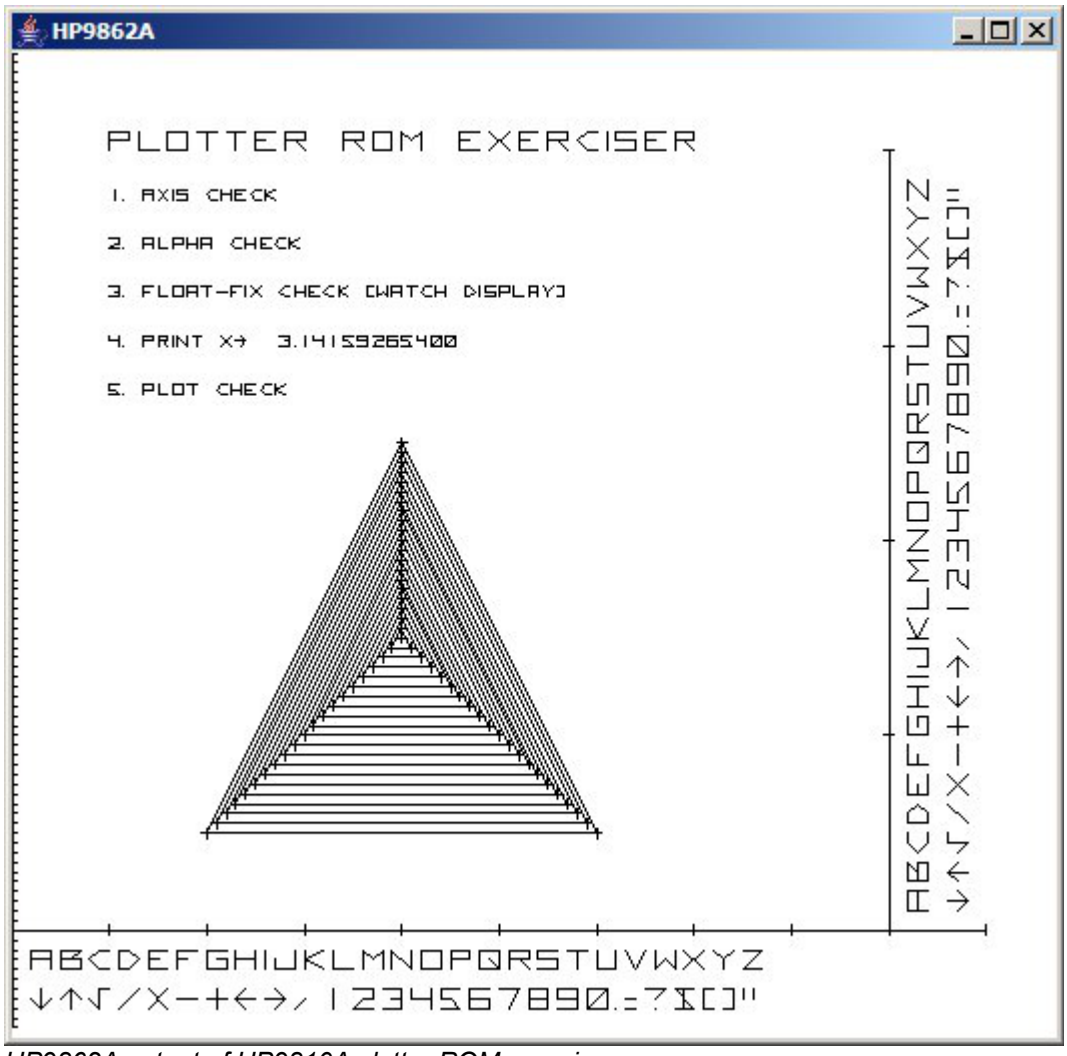
printing sound is also disabled.

When the HP9861A window has the focus pressing the key S toggles between high-speed and normal speed output. The high-speed mode is visualized in the window title bar.

3.3 HP9862A Plotter

3.3.1 General Information

The original HP9862A is a single pen flatbed x/y plotter with a resolution of 10000 by 10000 units. The emulation uses its own window for drawing. The graphic output is internally buffered so the window can be resized as desired and the graphics will be redrawn.



HP9862A output of HP9810A plotter ROM exerciser program

Different from the original the emulated HP9862A is able to change the pen color. This is achieved by driving the pen position beyond the right border and 'grab' one of the 15 virtual color pens. To change the pen color to N (1 <= N <= 15) output a plot command with x,y coordinates of (10000,N), e.g. on the HP9830A:

```
SCALE 0,9999,0,9999
PLOT 10000,N.
```

The x-value of the right border of course depends on the scaling factor.

The colors are defined as follows:

1	black (default color)
2	green
3	red
4	blue
5	cyan
6	magenta
7	yellow
8	orange
9-15	black

3.3.1.1 Calculator Configuration

The select code of the HP9862A is fixed to 14 by the interface. In the configuration file the select code has to be omitted:

```
DEV HP9862A
```

3.3.1.2 Device Configuration File

The device configuration file for the HP9862A is:

```
Model HP9810A HP9820A HP9821A HP9830A
Name HP9862A
Title PLOTTER
Interface HP9862Interface
Selectcode 14
```

3.3.2 Usage of the HP9862A

In GO9800 the plotter has its own window and runs in a parallel program thread. Plot commands can only be issued by the calculator. Every calculator needs a special add-on ROM to control the plotter, these are delivered with the GO9800 emulator. See the appropriate original manual for further informations.

The buffered output maybe controlled by the following PC keys:

PC key	**Function**
delete	Delete complete output
shift + delete	Delete complete output, reset to default window size
insert	Generates a hardcopy of the output
shift + insert	Opens the page format dialog for the hardcopy function.
S	Toggle high-speed mode

The plot output may be cleared by the Del key on the host PC. When the Shift key is pressed simultaneously, the output window will be resized to the default size of 500x500.

It is possible to send a the output to an arbitrary printer of the host PC. Pressing the PC keys Shift+Ctrl+P opens the standard page-setup dialog of the host system. Pressing Ctrl+P opens the standard print dialog of the host system, where the printer may be selected and the hardcopy started.

The emulated HP9862A has a special high-speed output mode, where all artificial delays are eliminated and the plot output is generated at the maximum possible speed. In normal output mode the delay resembles the timing of the real device. In high-speed mode the plotting sound is also disabled.

When the HP9862A window has the focus pressing the key S toggles between high-speed and normal speed output. The high-speed mode is visualized in the window title bar.

3.4 HP9865A Cassette Memory

3.4.1 General Information

The original HP9865A used modified audio compact cassettes for storage of programs and data. The original tape cassettes are emulated by files in the host PC file system. One host tape-file can contain a unlimited number of HP calculator files. Like on a real tape these files have to be created by the calculator using a MARK command. The movement of the tape is emulated by a read/write position in the host file. When a tape is loaded in the HP9865A the r/w position is 0 (zero). The position moves forward when reading or writing HP files on the tape. It moves backward when a REWIND or FIND is executed. When the r/w position reaches the begin or end of the host file a 'clear leader' is signalled to the calculator, which automatically stops the HP9865A. So the emulated device behaves like the real one.

To prepare a new host tape-file one needs a 'blank' tape. A original blank tape is not really empty, but contains at least eight 0-value bytes and a Begin-Of-File marker (hex 3C + control bit). Without that initial BOF no new files can be MARKed on that tape. The zero value bytes before BOF are necessary to prevent the drive running into a 'clear leader' position during FIND or LOAD, which would result into an error message on the HP9830A. In the GO9800 distribution there is file named 'blank.tape' which should be kept as source for new tapes. To create new tape make a copy of the file 'blank.tape' and give it a name of your choice. For convention it should have the extension .tape.

3.4.1.1 Calculator Configuration

In the configuration file the select code has to be set to an unused value between 1 and 9. The standard value which is assumed in the HP9810A and HP9820A cassette memory ROMs is 5:

```
DEV HP9865A 5
```

3.4.1.2 Device Configuration File

The device configuration file for the HP9865A is:

```
Model HP9810A HP9820A HP9821A HP9830A
Name HP9865A
Title CASSETTE_MEMORY
Interface HP9865Interface
Selectcode 5 1 2 3 4 6 7 8 9 10
```

3.4.2 Usage of the HP9865A

In GO9800 the cassette memory has its own window and runs in a parallel program thread.

To load a tape in the HP9865A bring the HP9865A to the foreground and mouse-click on the 'OPEN' door lever. The cassette door will be opened and a file selector dialog is displayed in which you can select your new empty tape. After successful selection of a tape the door will be closed again and a loaded cassette is visible in the door window.

The tape can only be read or written by commands issued on the calculator. On the HP9830A these commands are build-in, on the HP9810A and HP9820A one needs a

'Cassette Memory' ROM block. See the original manuals for reference.

The only tape operation which can be directly executed by the HP9865A is rewind. Mouse-click on the REWIND button places the read/write position to the beginning of the tape.

In the emulated drive the tape activity is not obvious since there are no moving reels in the emulator and the only feedback is acoustical by the simulated motor sounds. Therefore a tape activity indicator was added which shows if the tape drive is busy by using colored chevrons.

The tape activity indicators have the following meaning:

>	move tape forward slow
<	move tape backward slow
>>	move tape forward fast
<<	move tape backward fast

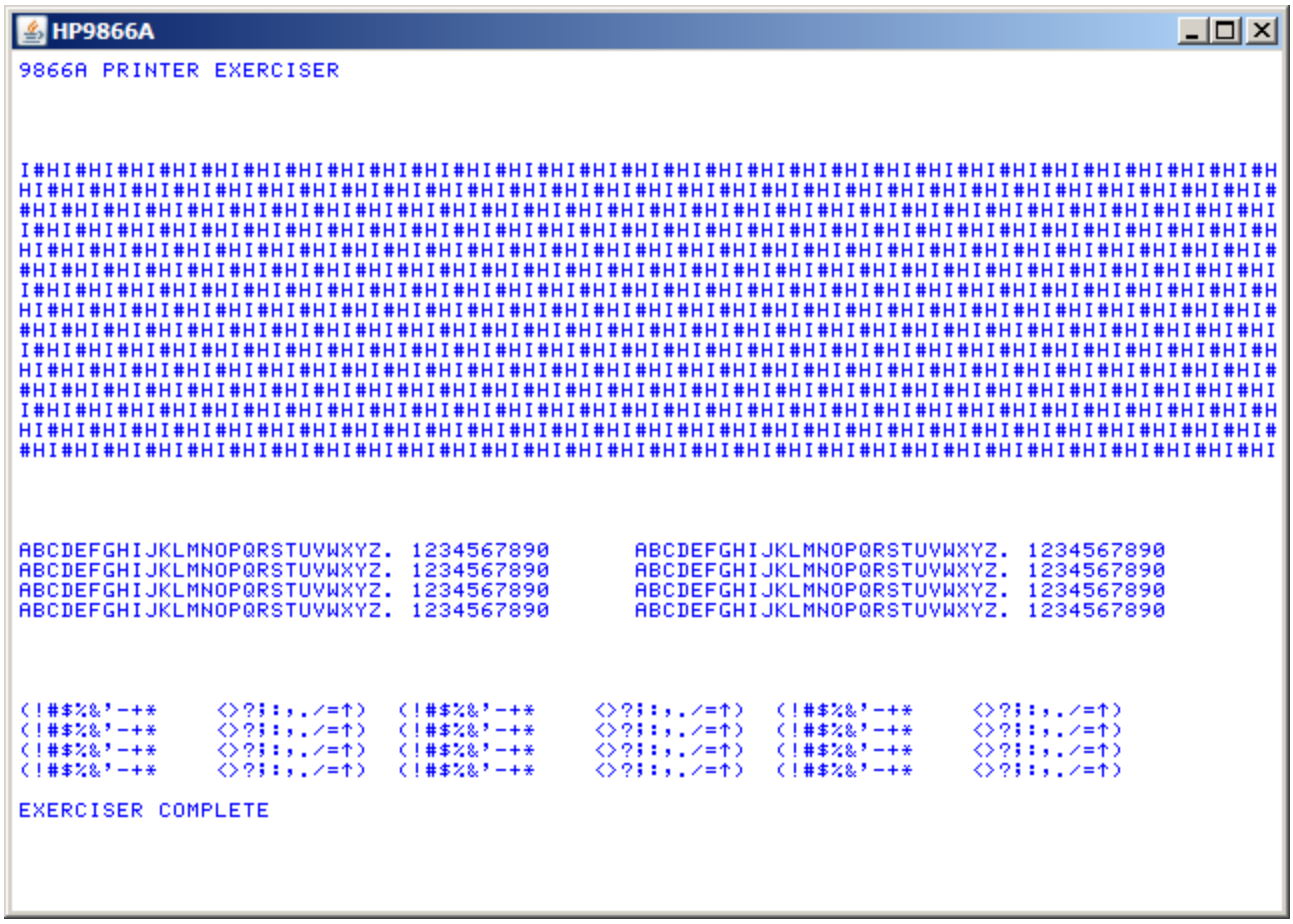
The color coding is:

yellow	search for control word (begin-of-file)
green	read from tape
red	write to tape

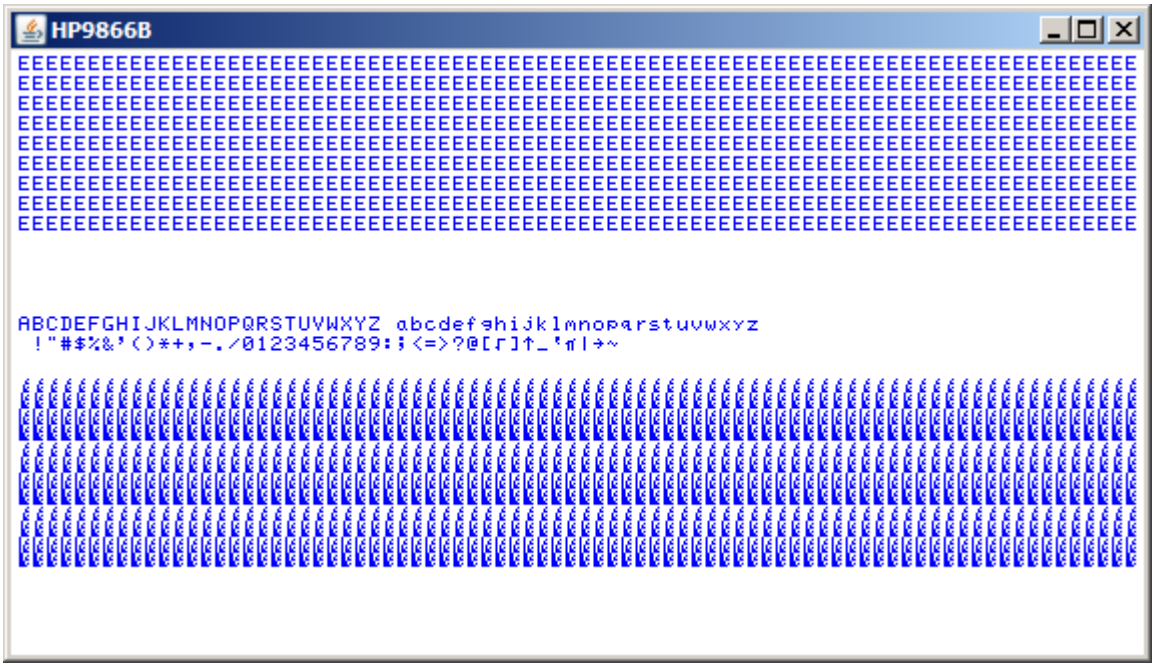
3.5 HP9866A/B Thermal Line Printer

3.5.1 General Information

The original HP9866A was a ASCII text printer which prints one complete line at a time on thermo sensitive roll paper. It was the primary printer for the HP9830A which had no internal printer like the HP9810 and HP9820. The HP9830A had a build-in interface for direct connection of the printer. For use with other calculators a special interface card was necessary. The HP9866B was the successor of the HP9866A with extended character set and graphics capability.



HP9866A output of HP9830A printer test program



HP9866B output of HP9830A printer test program

3.5.1.1 Calculator Configuration

In the GO9800 the HP9866A/B can be used with the HP9830A as well as with the other calculators. For use with the HP9830A/B the select code must be set to 15.

Typical configuration item:

```
DEV HP9866A 15
```

For use with the HP9810A or HP9820A an additional Peripheral Control ROM is necessary and has to be listed in the configuration file. The select code has to be set to an unused value between 1 and 9.

Example:

```
DEV HP9866B 8
```

The HP9866A/B may also be controlled by the HP9810A Typewriter ROM. The select code must then be set to 15.

3.5.1.2 Device Configuration File

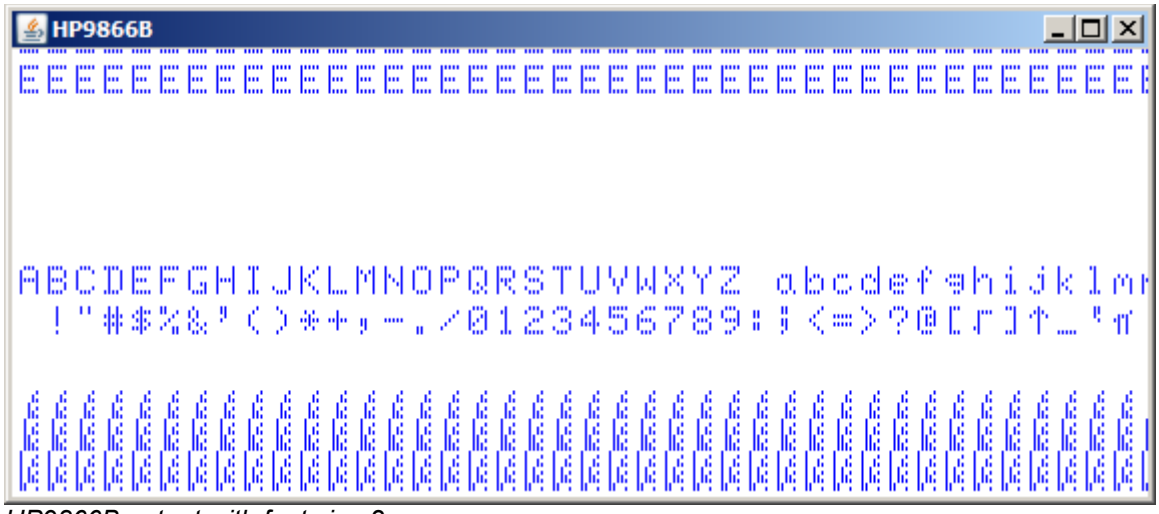
The device configuration file for the HP9866A is:

```
Model HP9810A HP9820A HP9821A HP9830A
Name HP9866A
Title LINE_PRINTER
Interface HP9866Interface
Selectcode 1 2 3 4 5 6 7 8 9 15
```

3.5.2 Usage of the HP9866A/B

The HP9830A commands PRINT, LIST, and TLIST can be used like on the original machine.

The HP9866A uses its own window for output. For text output the internal dot matrix font is emulated. The font size may be changed in fixed steps, each step increases the hight of a print dot by one display pixel. The original printer dots where slightly elliptic, this is only visible with font sizes larger than 7. The HP9866B has an additional graphics printing mode which is enabled for one text line by output of a DC1 character (decimal code 17). The lower five bits of each character in the line are printed as dot pattern. Detailed description of the HP9866B graphics may be found in the “HP9866A/B Options 10, 20, 21 and 30 Operating Manual” on pages 3-3 ff. A copy of this manual is available from the author on request. The output is automatically scrolled line by line in the window and is internally buffered. The window may be resized manually.



HP9866B output with font size 2

The buffered output maybe controlled by the following PC keys:

PC key	**Function**
page ↑	Scroll output one window page up
page ↓	Scroll output one window page down
home	Position to first (top) window page
end	Position to last window page
delete	Delete complete output
shift + delete	Delete complete output, reset to default font and window size
insert	Generates a hardcopy of the output
shift + insert	Opens the page format dialog for the hardcopy function.
+	Increase font size (max. 10)
-	Decrease font size (min. 1)
S	Toggle high-speed mode

It is possible to send the output to an arbitrary printer of the host PC. Pressing the PC keys Shift+Ctrl+P opens the standard page-setup dialog of the host system. Pressing Ctrl+P opens the standard print dialog of the host system, where the printer may be selected and the hardcopy started. The hardcopy is automatically scaled to fill the available page size. The scaling accounts for the selected font size.

The emulated HP9866A/B has a special high-speed output mode, where all artificial delays are eliminated and the print output is generated at the maximum possible speed. In normal output mode the delay resembles the timing of the real device. In high-speed mode the printing sound is also disabled.

When the HP9866A window has the focus pressing the key S toggles between high-speed and normal speed output. The high-speed mode is visualized in the window title bar.

3.6 HP9880A/B Mass Memory System

3.6.1 General Information

The original HP9880A/B was a storage system with fixed or removable magnetic discs with a capacity of 2.5 Mbytes each. It comprised of a HP11273B interface, a HP11305A disc controller unit and one or more HP9867A/B disc drives. Each HP11305A controller was able to connect up to four disc units, either up to four HP9867A single platter or two HP9867B dual platter drives. In the emulator both the HP9880A (with HP9867A drives) and the HP9880B (with HP9867B drives) is available.

The HP9867A has a single removable disc, while the HP9867B has one fixed (lower) and one removable (upper) disc. Each disc can be accessed by a unit number between 0 and 3. In the HP9867B the upper disc has the unit number 0 or 2, and the lower, fixed disc the unit number 1 or 3.

The physical disc is emulated by a binary file in the host PC file system. Before it can be used a new disc has to be initialized. The procedure is described below. Alternatively an initialized empty disc may be used. In the GO9800 distribution there is file named 'empty.disc' which should be kept as source for new discs. To create new disc make a copy of the file 'empty.disc' and give it a name of your choice. For convention it should have the extension .disc.

3.6.1.1 Calculator Configuration

In the GO9800 the HP9880A/B can be used with the HP9830A only. The select code is internally fixed to 11. There are 5 device configuration files for different combinations of drive and disk numbers.

```
HP9880A   (one HP9880A drive with one disk)
HP9880A_2 (two HP9880A drives with one disk each)
HP9880A_4 (four HP9880A drives with one disk each)
HP9880B   (one HP9880B drives with two disks)
HP9880B_2 (two HP9880B drives with two disks each)
```

Each of these device configurations may be referred in the calculator configuration.

Typical configuration items:

```
DEV HP9880A_4
DEV HP9880B
```

For use with the HP9830A an additional Mass Memory ROM is necessary and has to be listed in the configuration file. The ROM has to be placed in the uppermost slot in order to be functional. The corresponding configuration item is:

```
ROM 036000 002000 HP11273B Slot1
```

The HP11273B interface connects the HP11305A controller to the calculator and contains a 256-word cache memory which is mapped in the main memory at address 77000-77377. This cache is automatically configured in the HP9880A configuration file.

3.6.1.2 Device Configuration File

The device configuration file for the HP9880A is:

```
Model HP9830A
Name HP11305A
Title MASS_MEMORY_CONTROLLER
Interface HP11273A
Selectcode 11
RWM 077000 000400
Parameters 1 1
```

3.6.2 Usage of the HP9880A/B

In the emulator the HP9880 has a user interface showing the front panel of each HP9867 disc drive in its own window.

At startup each configured disc unit is loaded with a standard disc named

'HP9880-UNIT*n*.disc'

where *n* is the unit number.

A disc can be exchanged by mouse-clicking on the 'LOAD' switch on the right side of the HP9867. The 'DRIVE READY' lamp will go out and the 'DOOR UNLOCKED' lamp will be lighted. Then a file selector dialog is displayed in which a new disc-file can be selected. After successful selection of a disc the 'DRIVE READY' lamp will be lighted again and the other lamp will go out. If the dialog is dismissed without selecting a disc-file the drive will stay in the not-ready state and can not be operated until another disc is loaded.

The HP9867A/B drives have a write-protect mechanism which can be activated separately for each disc. Mouse-click on the 'PROTECT U.D.' or 'PROTECT L.D.' area of the HP9867 front panel and the corresponding disc will be write protected. In the HP9867A only the upper disc is present and can be protected.

In the emulated drive the disc activity is not obvious, therefore a disc activity indicator was added which shows which which disc record is currently accessed. The physical records have numbers between 0 and 9743 (corresponding to 2 surfaces with 203 cylinders and 24 sectors each).

The disc activity indicators have the following meaning:

Unnnn	HP9867B upper disc record nnnn
Lnnnn	HP9867B lower disc record nnnn
nnnn	HP9867A disc record nnnn

The color coding is:

yellow	Record being initialized
green	Record being read from disc
red	Record being written to disc

The emulated HP9867A/B has a special high-speed output mode, where all artificial delays are eliminated and the disc access is performed at the maximum possible speed. In

normal output mode the delay approximates the timing of the real device.

When the HP9867A/B window has the focus pressing the key S toggles between high-speed and normal speed output. The high-speed mode is visualized in the window title bar.

3.6.3 Initializing discs

Before a new disc can be used for data storage it has to be initialized (formatted). This can be accomplished by a utility program provided on the HP9880 Systems Tape. This tape is included in the GO9800 distribution as file

```
applications/HP9830/HP11273-60004 SYSTEMS TAPE.tape
```

The initialization procedure is described in the HP9880 operation manual and shortly outlined here. Load the systems tape into the HP9830 tape drive and execute

LOAD BIN 60

When the file is loaded execute

INITIALIZE

and follow the instructions printed on the HP9866A printer. When you are requested to set the mode switch to INITIALIZE on the controller, simply ignore this and continue.

3.6.4 Note on Usage of the Infotek Fast Basic II ROM

The Infotek Fast Basic II ROM contained in the GO9800 distribution has a feature which automatically executes the command

GET “↑”,10,10

when the HP9830 calculator is switched on. This command searches for a BASIC program file named “↑” on disc unit 0. If present, this file is loaded and started from line 10.

If at startup of the HP9830 the disc unit 0 is not ready or not present, the calculator beeps periodically and waits for unit 0 to become available. This can be abandoned pressing the STOP key.

If the file “↑” is not present on the disc ERROR 94 is displayed. If the file is not executable ERROR 98 occurs.

If the Infotek Fast Basic II ROM is not necessary and the above behavior is bothersome the ROM should be removed from the HP9830A configuration file. The following configured line should be deleted:

```
ROM 017000 001000 INFOTEK_FB2 Int0
```

3.7 HP11202A I/O Interface

3.7.1 General Information

The original HP11202A interface was a bidirectional 8 bit TTL interface. Like other peripheral interfaces it was plugged into one of the four slots on the rear side of the calculator. It could be connected by a cable to output devices like printers (e.g. HP9866A, HP9871A) or input devices like a card reader (HP9869A).

For control of the HP11202A and devices connected to it by the calculator functions from special add-on ROM blocks (Peripheral Control Block I or II for the HP9810A, HP9820A, and Extended I/O Block for the HP9830A) had to be used. Some simple functions where also implemented in the basic machines. The basic HP9830A was able to read programs from a HP9869A (PTAPE#) or output to a connected printer (LIST#).

3.7.1.1 Calculator Configuration

In the configuration file the select code has to be set to an unused value between 1 and 9. A standard value which is assumed for HP9869A paper tape reader and other devices is 1:

```
DEV HP11202A 1
```

3.7.1.2 Device Configuration File

The device configuration file for the HPA is:

```
Model HP9810A HP9820A HP9821A HP9830A
Name HP11202HostFileIO
Title Host_File_IO
Interface HP11202A
Selectcode 1 2 3 4 5 6 7 8 9
Parameters 16
```

The HP11202A can handle I/O data in different formats: as binary bytes or number strings in an arbitrary radix, e.g. as decimal, octal or hexadecimal numbers. The radix is configurable as an additional parameter.

`Parameters Bin`
configures binary format. Data is input or output as binary bytes.

`Parameters <`*n*`>`
configures input and output as number string in radix *n*. Each value is separated by a newline character. E.g.

`Parameters 8`
configures input and output as octal number string (e.g. 77). This is also the default format if the parameter is omitted.

3.7.2 Usage of the HP11202A

From the beginning there is no special user dialog for the HP11202A. Instead when one of the input or output commands is executed for the first time with the interface select code, a file dialog box is displayed. Then the location and name of an input or output file has to be entered. This file will be used for the current and following input resp. output commands.

There are two different files possible: one for input and another for output operations. So if both input and output operations occur over the same interface, two file selector dialogues are displayed.

Once an output or input file is opened an audit trail window is drawn in which every output byte will be displayed. The audit trail may be scrolled, formatted and printed in the same way as in the HP9861A (see chap. 3.2.2). Manual closing of the audit trail window causes any output or input file also to be closed.

When the end-of-file condition of an input file is met during an input operation, another file selector box will be displayed. If this dialog is dismissed by Cancel or no file is selected, a pause of 5 seconds is carried out. During this pause the input process can be cancelled by pressing the calculator STOP key. After the 5 second pause the file dialog is displayed again and the user gets a second chance.

3.7.3 Saving and Loading of HP9810A programs

Using the integrated Peripheral Control ROM HP9810A programs can be saved to a file in the host PC and later reloaded without using slow magnetic cards. To save a program press the following keys:

FORMAT 4 0 1 RECORD

To reload a saved program press:

FORMAT 3 0 1 GOTO

After the complete program is loaded the reading doesn't stop automatically but the file selector is displayed again. Dismiss this dialog by pressing Cancel and press STOP on the HP9810A emulation.

3.7.4 Saving and Loading of HP9830A programs

HP9830A programs can be saved to a host PC file using the LIST command:

LIST #1

The resulting text file may be opened by a text editor and printed.

To load a program from a text file enter:

PTAPE #1

After the complete program is loaded the reading doesn't stop automatically but the file selector is displayed again. Dismiss this dialog by pressing Cancel and press STOP on the HP9830 emulation.

4. Installation and Running

The emulator requires a JAVA runtime environment 1.7.0 or higher. The JAVA runtime may be downloaded from

https://www.java.com/de.

The emulator consists of a single JAR-archive which can be placed anywhere in the file system.

To run specific calculator a configuration file is needed (see chapter 1.1), which contains informations about the calculator model, the memory, and attached peripheral devices. The name of the configuration file is arbitrary.

Run the emulator by entering the command line

```
java -jar GO9800.jar [-d] <configuration-file>
```

During startup of the emulator a copyright note and various messages about configured components and devices are output to the command console. At the beginning a timing calibration is performend to evaluate critical timing constants which may be influenced by the host platform.

A typical console output looks like this:

```
HP Series 9800 Emulator Release 2.02, Copyright (C) 2006-2016 Achim Buerger

GO9800 comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to
redistribute it under certain conditions.

GO9800 is in no way associated with the Hewlett-Packard Company.
Hewlett-Packard, HP, and the HP logos are all trademarks of the Hewlett-Packard
Company.
This software is intended solely for research and education purposes.


HP9800 Microcode ROM loaded.
HP9800 ALUcode ROM loaded.
HP9800 BCDcode ROM loaded.
HP9800 ALU test:
0+0123456789
1+0123456789
2+0123456789
3+0123456789
4+0123456789
5+0123456789
6+0123456789
7+0123456789
8+0123456789
9+0123456789
0-0123456789
1-0123456789
2-0123456789
3-0123456789
```

```
4-0123456789
5-0123456789
6-0123456789
7-0123456789
8-0123456789
9-0123456789
HP9800 CPU loaded.
Timing calibration 1/1 3/3 7/7 8/8 10/10 16/16 20/20 30/30 32/32 100/100 done.
HP9800 I/O unit loaded.
Custom configuration file HP9810A.cfg loaded.
HP9810A_System1 ROM at 0-777, Block0 (SYSTEM1) loaded.
HP9810A_System2 ROM at 4000-5777, Block4 (SYSTEM2) loaded.
HP9810A_System3 ROM at 16000-16777, Block16 (SYSTEM3) loaded.
HP9810A_Data RWM at 1000-1777, Block1 initialized.
HP9810A_Program RWM at 12000-12777, Block12 initialized.
HP11217A RWM at 13000-13777, Block13 initialized.
HP11218A RWM at 14000-15777, Block14 initialized.
HP11210A ROM at 2000-3777, Slot1 (MATHEMATICS) loaded.
HP11267A ROM at 6000-7777, Slot2 (TYPEWRITER/CASSETTE_MEMORY) loaded.
HP11261A ROM at 10000-11777, Slot3 (PLOTTER/PRINTER_ALPHA) loaded.
HP9860A MARKED_CARD_READER HP11200A, select code 12 loaded.
HP9861A TYPEWRITER HP11201A, select code 15 loaded.
HP9862A PLOTTER HP9862Interface, select code 14 loaded.
HP9865A CASSETTE_MEMORY HP9865Interface, select code 5 loaded.
HP9866A LINE_PRINTER HP9866Interface, select code 8 loaded.
HP11202HostFileIO Host_File_IO HP11202A, select code 1  mode=16 loaded.
HP11202HostFileIO Host_File_IO HP11202A, select code 2  mode=Bin loaded.
Default configuration file C:\Users\Achim\workspace\9800 Emulator
2.02/config/keynames.cfg loaded.
Default configuration file C:\Users\Achim\workspace\9800 Emulator 2.02/config/HP9810A-
keyb.cfg loaded.
HP2116 Panel loaded.
HP9800 Printer loaded.
HP9800 Magnetic Card Reader loaded.
HP9810 Display loaded.
HP9810 KeyLEDs loaded.
HP9810 Keyboard loaded.
HP9810 Mainframe loaded.
HP9800 Emulator started.
```

For Windows and Linux / Unix / MacOS host operation systems the GO9800 distribution contains shell scripts for each of the calculator models to start the emulator. On Linux hosts the scripts have to be given execution permissions.

The GO9800 emulator is known to be working on the following host OS with Java runtime 1.7:

- Windows XP and later
- Linux
- Raspian on Raspberry Pi 3
- MacOS X

On Raspberry PI there is a known problem with the sound output and the emulator release 2.0 runs very slow. It is recommended to use release 1.61 instead.

5. Known Issues

5.1 HP9830A

Writing and reading of files in the internal or external HP9865A tape device is timing critical and may hang up on some machines. This due to timing and CPU load problems on the host machine.

5.2 Peripheral Devices

The devices with critical timing and operation in background, like HP9865A and the internal card reader, may not work properly on slow host machines. Symptoms are error messages, loss of data, and hangup of the emulated machine.

5.3 Java Issues

5.3.1 Direct3D Problem

In the Java runtime there is an issue with Microsoft DirectX output of graphics which may lead to rather slow drawing or flickering of bitmaps. This is mainly visible with the output of the LED display on the HP9820A. There is a workaround in deactivating the usage of Direct3D acceleration by the Java runtime. This is accomplished by adding the option -Dsun.java2d.d3d=false when starting the Java runtime.

This is already done in the startup CMD files for each calculator contained in the distribution. A typical Java call looks like this:

```
java -Dsun.java2d.d3d=false -jar GO9800.jar HP9810A
```

You may try if this option has an effect on drawing speed on your particular host system.

5.3.2 Window Resize Problem

In Java implementations for certain operating systems, e.g. MS Windows, resizing of an application window may lead to irregular or incomplete repaint of the window contents. This occurs especially with the resizable windows of the HP9862A and HP9866A/B devices. When the window size is enlarged by dragging the window corner with the mouse, the paint() method of the Frame is called very frequently. But obviously only the new exposed areas of the window are part of a system generated clipping area, so only these are painted according to the new window size. Since the scaling and position of the graphics (plot or print) depends on the actual window size, the new painted area doesn't fit in most cases to the existing graphics.

This problem is partially solved by painting the complete window once again after the window resize is finished by releasing the mouse button.

5.3.3 Window Redraw Problem

There is another problem related to the incomplete repaint described above. When moving the emulator main window the LED display and the calculator keyboard me be drawn incomplete or distorted. This also may happen sporadically without touching the window.

6. The Project

The objectives of this project where to understand and preserve the firmware of the Hewlett-Packard series 9800 calculators and to create a fully functional real-time simulation of the complete hardware.

Of course there where some legal concerns which had to be examined. As of April 2007 all patents are long since expired and there are no registered trademarks affected except the HP logo. The firmware ROMs, software tapes and magnetic cards don't carry any copyright notice. Also the contents of the most ROMs are disclosed as source code in the appropriate patents. Nevertheless all HP firmware and software is still regarded as copyrighted and used as is, without any changes.

The project began with acquisition of the ROM contents of the HP9810A and HP9820A by means of a logic analyzser. The next step was programming a disassembler in Java to get a deeper understanding of the 9800 CPU instructions. Furthermore by analyszing the disassembled firmware it was possible to understand the I/O procedure and interrupt structure.

One more result was the ability to create my own assembler programs for the 9800 CPU and run them on the real hardware. I discovered undocumented features in the 9810 Peripheral Control ROM which allowed to store and run binary machine programs. The first one was a kind of PEEK function for the HP9810A. By means of this the complete memory contents could be dumped. It was a good verification of the correctness of the logic analyzer read out. The next assembler program was a PEEK function for the HP9830A, whose firmware contents was not available so far. This also succeeded in a complete dump of the system ROM and additional plug-in ROMs.

The last phase of the project was the Java programming of a 9800 CPU emulator. Because of the properties of the display and keyboard I/O devices the HP9830A seemed to be the easiest candidate for emulation of the whole hardware. There where several risks in this phase as it was not sure whether the CPU instructions where completely documented in the various U.S. patents of the series 9800 machines. Also it was difficult to estimate the execution speed of the emulator. The machines should at least run in real time speed. Moreover the I/O principles where very poor documented and difficult to understand in detail.

In fact during the first tests of the emulator it showed that some CPU instructions had to behave different from the available 'documentation'. Esp. the floating point macro instructions took some weeks of debugging and step-by-step analyzing until they produced correct results. Also the stack handling of JSM /RET and the interrupt service routine gave some unexpected surprises. Late in the project, when testing the HP9820A together with the HP9865A, there showed one more anomaly in the IO interface. After many hours of testing, debugging, and re-programming it was only one possible solution left: the handling of the CEO handshake in the instruction STF 1 was wrong. There was only one constellation in the firmware of all three calculators where this special situation occurred.

With the core functionalities of the HP9830 and getting the machine up running with a very simple output and input interface, the proof of concept was done. After that I added a simulated keyboard and LED matrix display. Photographs and sound recordings of the original machine where used to resemble a most naturalistic simulation.

The display simulation is somewhat tricky: in the original machine the display output is controlled and multiplexed by the CPU. That means the CPU outputs every displayed

character many times a second to give to the human eye a quasi steady display. When implemented 1:1 in software, this resulted in heavy flickering and high load of the host CPU. So I decided to update the display output only if a character changed, i.e. when a new output was generated by DISP command, user entry and the like. This has only one disadvantage: the display keeps visible even if the calculator is busy. This issue finally is solved by clearing the simulated display if the DEN (display enable) signal is false for more than a certain amount of executed CPU instructions.

In the next steps several peripheral devices where added: the HP9860A marked card reader, the HP9866A thermal printer, and the HP9862A plotter. The device which caused the most work was the HP9865A cassette tape drive, which is available build-in to the HP9830A and as external device. The I/O protocol is very complicated and the timing crucial, since the 9865 reads and writes asynchronously to the main machine. Some tape operations run completely in background and send interrupts to the calculator. JAVA multi threading techniques had to be used intensely. It took several weeks until the 9865 worked satisfactory.

When the HP9830 and the peripherals where completed the next machine to be implemented was the HP9810A. Reusing or extending most of the classes of the HP9830, the 9810 was basically brought up in a few days. Again the display simulation required a special timing logic. Most work had to be spend in the magnetic card reader. It has a similar I/O protocol as the HP9865, but the timing even more critical. Again it took more than two weeks to get a stable solution.

The last one was the HP9820A. With the experience of the first two machines it would have take only a few days to complete the 9820. If there didn't occur this annoying problem with the cassette memory ROM and the 9865 (see above).

In several releases of the emulator more add-on ROM blocks and application programs where added to the distribution. Some of them where borrowed by contributors (see chapter 7) and read out using the mentioned assembler programs. Some ROM blocks are obviously very rare and could not be acquired until now. Amongst these are the HP9810A User Defined Functions ROM and the Typewriter ROM. Since their contents are completely listed in the HP9810A patent I decided to re-create them from scratch and type the octal values from the patent listings. By disassembling and cross-checking the instructions all typing errors could be eliminated. The graphics of the ROM blocks and keyboard overlays where also created artificially using GIMP on basis of other existing ROMs.

In 2007 the HP9880A/B mass memory system was added to the project, in 2008 the HP9861A impact printer, and in 2011 the HP9866B graphics capable thermal printer. At this time the complete printer and plotter output was reworked for scalability and hardcopy to a real printing device. The last major addition and intended finish of the project was the HP9821A in 2011. This was made possible by a acquisition of the original HP9821A system ROMs.

As of May 2012 and release 1.61 the project took more than 1000 hours of time for research, design, programming, and testing. Writing this manual took another 100+ hours.

In 2016 I reopened the project to implement execution of the CPU micro-code. Up to release 1.6 the arithmetic and logic functions and the instruction decoding where programmed in plain java code. This gave a rather fast execution but led to the problems described above. In the real hardware the main logic of the CPU is coded in a micro-program contained in seven ROMs. This micro-program is responsible of decoding the

CPU instructions (on assembler code level) and executing them. Two additional ROMs contain the logic of the binary and decimal (BCD) arithmetic operations.The ROM contents are well known as they are printed as binary values in the patent documents. There where three main issues which where unsolved until 2016: first, the ROM contents had to be dumped or alternatively the binary values had to be retyped from the patent prints, which is very unpleasant and error prone. Second, the binary code had to be decoded to micro-instructions, and third the micro-program had to be analyzed and understood.

In 2016 I stumbled over the work of Brent Hilpert (see link below), who had typed the contents of all CPU ROMs from the patent print and decoded it to a complete listing of the micro-program. He also scrutinized the CPU hardware to understand the micro-code and its control functions. By using this work I was able to implement the micro-code execution in GO9800 in about 100 hours of work. The implementation required the detailed simulation of major parts of the CPU hardware, such as shift-registers, buses, flip-flops, and several logic gates. Also it was neccessary to rework the complete I/O logic as it is entangled with the CPU and bus structure. Up to release 1.6 the functionality of the 'I/O-register and gate interface' where also programmed based on the describtion in the patent documents. Now also parts of the hardware are simulated in Java code, e.g. the I/O state machine and the I/O shift-register.

After another 100 hours of testing and debugging the release 2.0 was running, but as expected, much slower (about 25-30 times) than the previous releases. This is mainly due to the simulation of the bit-serial CPU buses and ALU functions. Nevertheless the advantage of this implementation is, that no assumptions on the CPU functions have to be made, such as, how the BCD arithmetic is working. It simply works by executing the micro-program. But this 'simplicity' was not as easy to realize as I thought. During testing several problems arose, as the micro-code execution has a somewhat complicate 'timing' which required an in-depth understanding of the state machine and clock cycles. To accomplish this, the running emulator release 1.6 was extremely helpful, as I was able to compare the results on instruction level step-by-step. As the last work the still Java-programmed 1-bit arithmetic functions where replaced by readout of the ALU and BCD ROMs. Now the emulator didn't need any more 'calculus knowledge'. But switching to the ALU and BCD ROMs slowed down the emulation again by nearly a factor of two. Also, testing revealed a few typing errors in Brents ROM data and also in the patent prints. There is one error in the printed binary code of the BCD ROM which, if this was manufactured 1:1 in the ROM hardware, would lead to erroneous results in BCD subtraction of certain values (e.g. 3 – 8 + carry). Strange enough, this error could never occur in the real calculators, as BCD subtraction is only used in two CPU instructions, the complement CMX and CMY. Both subtract values from 0 (zero) and the special constellation of operators, which leads to the error, never happens.

6.1 Literature and Links

The architecture of the HP9800 series is most completely described in the following patents available from the U.S. patent office (www.uspto.gov):

HP9810A: 3,859,635 Programmable Calculator
HP9820A: 3,839,630 Programmable Calculator Employing Algebraic Language
HP9830A: 4,012,725 Programmable Calculator

There are also german patents available from Deutsches Patent- und Markenamt (www.dpma.de):

HP9810A: 2264920, 2264923, 2264871
HP9820A: 2262725, 2264896, 2264897, 2264898
HP9830A: 2333908, 2365567, 2365568, 2365570

Further information about the history and technology of HP calculators can be found at

www.hpmuseum.org

www.hpmuseum.net

www.hp9825.com

www.hp9831.com

www.hp9845.net

http://www.classiccmp.org/hp/9800.htm

Most calculator and peripheral manuals are available for download from

www.hpmuseum.net

In-depth informations on the HP9830A hardware, the CPU architecture and micro-code and many more can be found on Brent Hilperts site

http://www.cs.ubc.ca/~hilpert/e/HP9830/

A nice video with the real HP2116 control panel in action

https://www.youtube.com/watch?v=9NWvNHGFRI8

7. Contributions to the Project

The author doesn't own all option ROMS, interfaces and peripheral devices. So all users are invited to contribute by borrowing or donating items which are still missing. These can be analyzed or read out in the authors lab.

Also wanted is original software for all models. There are methods to transfer program listings or file dumps to a PC, these programs could be made available for use with the GO9800.

You may also join as a developer. The IDE used for this project is Eclipse.

7.1 Contributors

Jon Johnston (hpmuseum.net)

provided several ROM blocks (HP11252A, HP11262A, HP11222A, HP11283B). The ROM contents where read out and made available in the emulator.

He also provided photographs of an original HP9821A, and the components of the HP9880B mass memory system.

Moreover he borrowed several original HP tape cassettes and magnetic cards containing applications, diagnostics, and system programs for the HP9810, 9820, and 9830. The programs where extracted and included in the applications folder of the emulator.

Tim Eliseo

has created several add-on software packages for the HP9830A and has provided several bug fixes and performance improvements for the emulator.

David Bryan

was the initiator of the keyboard configuration file and provided several hints for functional improvement and internationalization. He also tested pre-releases of the emulator and localized a lot of bugs. Dave also created internationalized keyboad configurations for the HP9820A and HP9830A which especially matches US keyboards.

Philippe Sonnet

borrowed several magnetic cards with programs for the HP9810A. The contents where extracted and are included in the applications folder of the emulator.

Appendix

A Creation and Execution of Assembler Programs

Originally the calculators of the 9800 series where not designed to execute user programs in assembler or machine language. The machine instructions of the 9800 CPU where not well documented (only as part of some U.S. patents describing the machines) and there was no assembler to generate machine instructions from an assembler source code.

Nevertheless there where so called special programs available from HP, which served for certain applications esp. in controlling of peripheral devices. These special programs where created outside of the calculators and sold by HP.

During realization of the HP9800 emulator most of the system and plug-in ROMs where disassembled and analyzed. Several ways where found to insert user created machine programs into the calculator memory and execute them. Of course detailed knowledge of the assembler language, which is common to all calculators, and of the machine instructions is necessary. There is no assembler available to generate machine instructions from an assembler source. The coding has to be done by hand. It is beyond the scope of this manual to describe the 9800 CPU instructions in detail.

Starting from an assembler source the first step is to obtain the instruction opcodes, a 16-bit code of each CPU instruction.

A.1 HP9810A

To execute assembler programs on a HP9810A the presence of a Peripheral Control ROM 1 or 2 is necessary. These ROMs contain undocumented features to store a machine language program in the user RWM and execute it.

Machine programs are stored in the user RWM, precisely in data registers, beginning from the highest available register (108) and continuing with register 107, 106, etc. The data registers are located at the octal memory addresses 001000 (register 108), 001004 (register 107), and so on. So the machine instructions are stored from address 001000 upward. In fact the first word at 001000 is used as an internal program length counter, so the user program starts at address 001001.

A.1.1 Coding the Program

For preparation all opcodes of the machine program have to be represented as 4 digit (16 bit) hexadecimal values. This can conveniently done using OpenOffice Calc. Next every hexadecimal value has to be represented by HP9810A keys with matching key codes by the following scheme:

Hex digit	HP9810A key
0 – 9	0 - 9
A	x^2
B	a
C	b
D	G
E	F
F	1/x

Example:

Instruction	Oct. opcode	Hex. opcode	HP9810A keys			
LDB B,I	074537	795F	7	9	5	1/x
STB 01717	035717	3BCF	3	a	b	1/x
LDA 016	020016	200E	2	0	0	F
CLR	170000	F000	1/x	0	0	0

A.1.2 Entering the Program

Before entering the coded program the length counter at address 001000 has to be cleared. This is done by storing 0 to register 108.

Press: 0 x→() 1 0 8

Then each opcode has to be entered using the undocumented format sequence

FORMAT 3 b key1 key2 key3 key4

Each time this sequence is entered the original 16 bit value is reconstructed by the calculator and stored in a memory position given by the said length counter plus 01001. After that the length counter is incremented by one.

For the example from above the keys to be pressed would be:

FORMAT 3 b 7 9 5 1/x
FORMAT 3 b 3 a b 1/x
FORMAT 3 b 2 0 0 F
FORMAT 3 b 1/x 0 0 0

This procedure is of course lengthy and error prone. Mistakes can not be corrected. So the best way is to enter the whole sequence in the program memory and save it on a magnetic card. The code then can be corrected re-entered as necessary.

A.1.3 Executing the Program

The stored machine program can be executed by the format sequence

FORMAT 3 CONTINUE

The machine program is executed beginning at address 001001. The last executed instruction should be RET (opcode 170402).

A.1.4 Example Program

Here is an example assembler program which implements a 'PEEK' function. The program takes the value of the X register, converts it to a 16 bit address value, reads the memory location pointed to by this address, and returns the value in the X register.

The resulting format sequences are stored on a magnetic card, which is contained in the GO9800 distribution (applications/HP9810/HP9810A PEEK.mcard).

```
01001 020004            LDA D4      ; A = addr of X
01002 024015            LDB D15     ; B = addr of AR1
01003 170004            XFR         ; transfer X->AR1
01004 074742            SBR 16      ; B = 0
01005 035717            STB D1717   ; (BTMP) = B
01006 021744            LDA D1744   ; A = Exponent word of AR1
01007 070113            SAP *+2     ; A>=0?
01010 067024            JMP J1024   ; If not: address=0
01011 070342      J1011 SAR 8       ; A = Exponent byte
01012 024063            LDB D63     ; B = 1
01013 070037            ADB A       ; B = A+1
01014 000023            ADA D23     ; A = A-5
01015 070112            SAM *+2     ; Exponent >4 ?
01016 025065            LDB D1065   ; B = 5           ; max. 5 digits
01017 035716      J1017 STB D1716   ; (addrC) = B     ; Address counter
01020 175400      J1020 DLS         ; get highest digit of AR1 and shift AR1
left
01021 063056            JSM J1056   ; find binary equivalent
01022 055716            DSZ D1716   ; all digits processed?
01023 067020            JMP J1020   ; no, repeat
;
01024 074537      J1024 LDB B,I     ; B = (B)   ; load contents of memory
location in B
01025 035717            STB D1717   ; (BTEMP) = B
01026 020016            LDA D16     ; A = addr of AR2
01027 170000            CLR         ; Clear AR2
01030 021066            LDA D1066   ; A = 16    ; process 16 bits
01031 031716            STA D1716   ; initialize counter
01032 025717      J1032 LDB D1717   ; B = (BTEMP)
01033 074133            SBP *+2,C   ; Is bit15 of B zero?
01034 072075            SEC *+1,S   ; Set E to 1111 binary
01035 074744            SBL 1       ; B = B<<1
01036 035717            STB D1717   ; (BTEMP) = B
01037 020016            LDA D16     ; A = addr of AR2
01040 024015            LDB D15     ; B = addr of AR1
01041 170004            XFR         ; transfer AR2->AR1
01042 170560            FXA         ; AR2 = AR2+AR1+E = 2*AR2 + bit15(BTEMP)
01043 055716            DSZ D1716   ; all bits processed?
01044 067032            JMP J1032   ; no, continue
01045 171450            NRM         ; normalize AR2
01046 074056            CMB         ; B = !B = -B-1
01047 004106            ADB D106    ; B = 12-B-1 = 11-B
01050 074404            SBL 8       ; B = B<<8  ; exponent byte
01051 035754            STB D1754   ; Exponent word of AR2 = B
01052 020016            LDA D16     ; A=addr of AR2
```

```
01053 024004            LDB D4      ; B=addr of X
01054 170004            XFR         ; transfer AR2->X
01055 170402            RET
;
01056 025717      J1056 LDB D1717   ; B = (BTEMP)
01057 074704            SBL 2       ; B = 4*B
01060 005717            ADB D1717   ; B = B+BTEMP = 5*BTEMP
01061 074037            ADB B       ; B = 10*BTEMP
01062 070037            ADB A       ; B = B+A = 10*BTEMP+A
01063 035717            STB D1717   ; (BTEMP) = B
01064 170402            RET
;
01065 000005      D1065 OCT 00005   ; 5
01066 000020      D1066 OCT 00020   ; 16
```

A.2 HP9830A

To create assembler programs on a HP9830A and store them permanently on tape the presence of a Infotek Fast Basic ROM II is necessary. This ROM can be used to directly read and write memory locations by means of two undocumented functions. Additionally with the FDATA command it is possible to create binary files and store machine programs on tape.

Machine programs are stored in the user RWM, at the upper end of the RWM area. The structure of a machine program is the same as of an add-on ROM module. In fact machine programs are managed by the system exactly like ROM blocks. A machine program can contain one or more commands or functions with their own keywords. These keywords can be used in the same way in BASIC programs or command lines like with build-in functions.

The ROM block structure is described in the U.S. patent 4,012,725 (Programmable Calculator). Below follows an excerpt from this patent.

Communication with the routines in the various plug-in read-only memory modules is accomplished through a series of mnemonic tables and jump tables. The standard firmware, the cassette operating firmware, and each of the plug-in read-only memory modules all contain the following tables:

1. A statement mnemonic table
2. A statement syntax jump table
3. A statement execution jump table
4. A system command mnemonic table
5. A system command execution jump table
6. A function mnemonic table
7. A function execution jump table
8. A non-formula operator mnemonic table

All of these tables, with the exception of the statement execution jump table, may appear anywhere within a memory module. The last five words in each module are used by the table scan routines [...] to find the actual location of the tables. The last word of each module contains a unique operation code word for that particular module. The second from the last word contains a relative address of the statement mnemonic table. The third from the last word contains a relative address of the system command mnemonic table. The fourth from the last word contains a relative address to the function mnemonic table. The fifth from the last word contains a relative address to the non-formula operator table. The jump tables for statement syntax, system command execution, and function execution are located directly above their respective mnemonic tables. The statement execution jump table is located directly above the fifth from the last word of each module. The complete set of tables for the standard firmware is shown in the firmware listings [...].

Each of the mnemonic tables consists of a string of seven-bit ASCII character and six-bit operation code characters packed two characters per sixteen-bit word. The eighth bit of each character is used to indicate whether that character is ASCII or an operator code. A zero in the eighth bit indicates ASCII and a one indicates an operation code. The seventh bit of each operation code character is used to indicate whether that operation code is the last character in that table. The jump table address for each mnemonic is found by subtracting the operation code for that mnemonic from the starting address of the associated mnemonic table. The internal stored format for program statements consists of a series of operation codes, operand codes, and other special codes. The first word of each statement contains the line number of that statement in binary format. The second word contains both the operation code for that particular statement mnemonic and also the length of the statement. The length information is used by various firmware routines to scan from one statement to the next. The third word contains the operation code for the table or optional read-only memory module, and it also contains the first operand code. The remainder of the statement is stored with one operator code and one operand code in each word.

A.2.1 Coding the Program

The opcodes of the machine program have to be supplemented by a pointer structure and memory for the keyword(s) as explained above. This can be demonstrated best by means of a sample program.

The following program represents a PEEK function which returns the contents of a memory location whose address is given as function parameter. In this example the program is located at the end of first memory page 000000-001777. Generally the machine program should be written page relocatable, that means that only page relative addressing should be used. Later the program will be loaded at the end of the last RWM page.

```
001750      4           4           ; Execution Jump Table: +4 for Opcode=1
001751      50105       "PE"
001752      42513       "EK"
001753      140400      OPCODE=1    ; Opcode-Bit(0x80) + Last-Opcode(0x40) + 1
001754      060445      JSM XFAR2+1 ; Transfer Argument -> AR2
001755      160225      JSM FLTRA,I ; Convert Float AR2 -> Int B
001756      074742      SBR 16      ; If Overflow: B=0
001757      074537      LDB B,I     ; load contents of memory location in B
001760      164227      JMP FXFLA,I ; Convert Int to Float ->AR2 and return
001761      000000                  ; unused memory
001762      000000                  ;
001763      000000                  ;
001764      000000                  ;
```

```
001765      000000                ;
001766      000000                ;
001767      000000                ;
001770      000000                ;
001771      000000                ; unused memory
001772      177777      -1        ; Statement Jump Table
001773      177777      -1        ; Ptr. to Non-formula Operator Mnemonic
Table
001774      177755      -19       ; Ptr. to Function Mnemonic Table
001775      177775      -3        ; Ptr. to Command Mnemonic Table
001776      177774      -4        ; Ptr. to Statement Mnemonic Table
001777      060000                ; Module operation code 60000
```

A.2.2 Entering the Program

For preparation all opcodes of the machine program have to be represented as decimal integer values between -32767 and +32767. This can conveniently done using OpenOffice Calc. Next a BASIC program has to be written which creates a binary tape file from the assembler opcodes. The decimal opcodes have to be stored in DATA statements.

For creation of the binary file a special plug-in ROM is necessary, which is supplied with the GO9800 emulator. The Infotek Fast Basic ROM II contains the FDATA instruction which allows creation of arbitrary file types. For further informations refer to the original Infotek manual.

The appropriate BASIC program for the above PEEK example is presented here:

```
10 DIM HI[7],BI[24]
20 FOR I=1 TO 7
30 READ H[I]
40 NEXT I
50 FOR I=1 TO 24
60 READ B[I]
70 NEXT I
80 FDATA S,H[1],B
90 STOP
100 DATA 0,24,1,50,0,0,0
110 DATA 4,20549,17739,-16128,24869,-8043,31202,31071,-5993,0,0,0
120 DATA 0,0,0,0,0,0,-1,-1,-19,-3,-4,24576
130 END
```

The DATA statement in line 100 defines the header of the binary file. The first value is the file number (in this example 0), the second the aktual file size (24 words), the third value is the file type (HP9830 binary=1), and the fourth value is the absolute file size.

Before executing the program MARK an empty file 0 (or whatever was defined as file number) of the same absolute size as in the above DATA statement. Then position the tape to the beginning of the MARKed file using FIND 0 (or whatever). Then RUN the program.

A.2.3 Executing the Program

An assembler program can be loaded from a binary tape file into the calculator memory by the command

LOAD BIN <file number>

After that the assembler program can be executed by entering the corresponding keyword, which is defined in the program.

The example program above can be executed by entering PEEK <address>, where address is an integer value. PEEK is a function and returns an integer value which can be used in expressions like any other value.

A.3 HP9820A / HP9821A

At the moment there is no method known to directly create assembler programs on a HP9820A. Nevertheless it is possible to create binary files for this machine using a HP9830A and to load and execute machine programs on the HP9820/21A.

To execute assembler programs on a HP9820A the presence of the HP11223A Cassette Memory / Special Programs ROM is necessary. The HP9821A contains the required functions in system ROM.

Machine programs are stored in the user RWM, at the lower end of the RWM area, beginning with address 016426 (octal). The structure of a machine or so called 'special' program is much similar to that used for the HP9830A, which is described above. A machine program can contain one or more commands or functions identified by keywords. A program is found by its keyword using the instruction CSP "*keyword*" (Call Special Program). The memory used by the program has to be a integer multiple of 8 words plus 3 and the last word has to be 0177777 octal (-1 decimal).

To create HP9820/21A assembler programs on a HP9830A and store them permanently on tape the presence of a Infotek Fast Basic ROM II is necessary. The FDATA command is used to create binary files and store machine programs on tape.

A.3.1 Coding the Program

The opcodes of the machine program have to be supplemented by a pointer structure and memory for the keyword(s). This can be demonstrated best by means of a sample program.

The following program represents a PEEK function which returns the contents of a memory location whose address is given in register Z. The result is returned in register Z. The program code is nearly identical to that of the HP9810A example program above.

```
16426 016430           OCT D16430  ; Pointer to first name
16427 177777           -1          ; End of list
16430 050105    D16430 "PE"
16431 042513           "EK"
16432 177773           -4-1        ; -(len+1) = End of name
16433 016434           J16434      ; Pointer to program start
16434 020442    J16434 LDA D442    ; A = addr of A-reg.
16435 000451           ADA D451    ; A = A+024 = addr of Z-reg.
16436 024447           LDB D447    ; B = addr of AR1
16437 170004           XFR         ; transfer Z->AR1
16440 074742           SBR 16      ; B = 0
16441 035717           STB D1717   ; (BTMP) = B
16442 021744           LDA D1744   ; A = Exponent word of AR1
16443 070113           SAP *+2     ; A>=0?
16444 067460           JMP J16460  ; If not: address=0
```

```
16445 070342    J16445  SAR 8       ; A = Exponent byte
16446 024403            LDB D403    ; B = 1
16447 070037            ADB A       ; B = A+1
16450 000411            ADA D411    ; A = A-5
16451 070112            SAM *+2     ; Exponent >4 ?
16452 024377            LDB D377    ; B = 5           ; max. 5 digits
16453 035716    J16453  STB D1716   ; (addrC) = B     ; Address counter
16454 175400    J16454  DLS         ; get highest digit of AR1 and shift AR1
left
16455 062513            JSM J16513  ; find binary equivalent
16456 055716            DSZ D1716   ; all digits processed?
16457 066454            JMP J16454  ; no, repeat
;
16460 074537    J16460  LDB B,I     ; B = (B)   ; load contents of memory
location in B
16461 035717            STB D1717   ; (BTEMP) = B
16462 020450            LDA D450    ; A = addr of AR2
16463 170000            CLR         ; Clear AR2
16464 020364            LDA D364    ; A = 16    ; process 16 bits
16465 031716            STA D1716   ; initialize counter
16466 025717    J16466  LDB D1717   ; B = (BTEMP)
16467 074133            SBP *+2,C   ; Is bit15 of B zero?
16470 072075            SEC *+1,S   ; Set E to 1111 binary
16471 074744            SBL 1       ; B = B<<1
16472 035717            STB D1717   ; (BTEMP) = B
16473 020450            LDA D450    ; A = addr of AR2
16474 024447            LDB D447    ; B = addr of AR1
16475 170004            XFR         ; transfer AR2->AR1
16476 170560            FXA         ; AR2 = AR2+AR1+E = 2*AR2 + bit15(BTEMP)
16477 055716            DSZ D1716   ; all bits processed?
16500 066466            JMP J16466  ; no, continue
16501 171450            NRM         ; normalize AR2
16502 074056            CMB         ; B = !B = -B-1
16503 004370            ADB D370    ; B = 12-B-1 = 11-B
16504 074404            SBL 8       ; B = B<<8  ; exponent byte
16505 035754            STB D1754   ; Exponent word of AR2 = B
16506 020450            LDA D450    ; A=addr of AR2
16507 024442            LDB D442    ; B=addr of A-reg.
16510 004451            ADB D451    ; B = B+024 = addr of Z-reg.
16511 170004            XFR         ; transfer AR2->Z
16512 170402            RET
;
16513 025717    J16512  LDB D1717   ; B = (BTEMP)
16514 074704            SBL 2       ; B = 4*B
16515 005717            ADB D1717   ; B = B+BTEMP = 5*BTEMP
16516 074037            ADB B       ; B = 10*BTEMP
16517 070037            ADB A       ; B = B+A = 10*BTEMP+A
16520 035717            STB D1717   ; (BTEMP) = B
16521 170402            RET
16522 000000            0           ; Filler word to int. mult. of 8 words
16523 000000            0           ; Filler word to int. mult. of 8 words
16524 000000            0           ; Filler word to int. mult. of 8 words
16525 000000            0           ; Filler word to int. mult. of 8 words
16526 000000            0           ; Filler word
16527 000000            0           ; Filler word
16528 177777            -1          ; End of memory for binary program
```

A.3.2 Entering the Program

For preparation all opcodes of the machine program have to be represented as decimal integer values between -32767 and +32767. This can conveniently done using OpenOffice

Calc. Next a BASIC program has to be written which creates a binary tape file from the assembler opcodes. The decimal opcodes have to be stored in DATA statements.

For creation of the binary file a special plug-in ROM is necessary, which is supplied with the GO9800 emulator. The INFOTEK Fast Basic ROM II contains the FDATA instruction which allows creation of arbitrary file types. For further informations refer to the original Infotek manual.

The appropriate BASIC program for the above PEEK example is presented here:

```
10 DIM HI[7],BI[60]
20 FOR I=1 TO 7
30 READ H[I]
40 NEXT I
50 FOR I=1 TO 60
60 READ B[I]
70 NEXT I
80 FDATA S,H[1],B
90 STOP
100 DATA 0,60,28,100,0,0,0
110 DATA 7448,-1,20549,17739,-5,7452,8482,297
120 DATA 10535,-4092,31202,15311,9188,28747,28464,28898
130 DATA 10499,28703,265,28746,10495,15310,-1280,25931
140 DATA 23502,27948,31071,15311,8488,-4096,8436,13262
150 DATA 11215,30811,29757,31204,15311,8488,10535,-4092
160 DATA -3728,23502,27958,-3288,30766,2296,30980,15340
170 DATA 8488,10530,2345,-4092,-3838,11215,31172,3023
180 DATA 30751,28703,15311,-3838
190 DATA 0,0,0,0,0,0,-1
200 END
```

The DATA statement in line 100 defines the header of the binary file. The first value is the file number (in this example 0), the second the actual file size (60 words), the third value is the file type (HP9820 binary=28), and the fourth value is the absolute file size.

Before executing the program MARK an empty file 0 (or whatever was defined as file number) of the same absolute size as in the above DATA statement (100). Then position the tape to the beginning of the MARKed file using FIND 0 (or whatever). Then RUN the program.

A.3.3 Executing the Program

A HP9820A machine program can be loaded from a binary tape file into the calculator memory by the command

LDF <file number>

After that the program can be executed this the CSP instruction with the corresponding keyword parameter, which is defined in the program.

The example program above can be executed by entering

<address> → Z

CSP “PEEK”

The register Z now contains the memory value at the given address.

A.3.4 Saving and Deleting the Program

There are two more functions in the Cassette Memory / Special Programs ROM for managing special programs by means of the key ISP (Initialize Special Program).

A special program already loaded in the calculator memory may be saved to a new tape file be means of the command

ISP 1,<file number>

Later it can be loaded again from this file.

A HP9820A machine program can normally not deleted by pressing the ERASE key. Instead the command

ISP 2

has to be executed. This carries out a soft reset of the machine (like power-on).

WARNING: the complete RWM is erased by ISP 2, including user programs and data!

B Machine Instruction Set

The design of the series 9800 calculators, a firmware listing, and the description of the HP9800 CPU instruction set is contained in the following U.S. patents:

HP9810A: 3,859,635 Programmable Calculator

HP9820A: 3,839,630 Programmable Calculator Employing Algebraic Language

HP9830A: 4,012,725 Programmable Calculator

Below is an excerpt of the HP9830A patent which describes the CPU instructions and their corresponding opcodes.

BASIC INSTRUCTION SET

Every routine and subroutine of the calculator comprises a sequence of one or more of 71 basic sixteen-bit instructions listed below. These 71 instructions are all implemented serially by the micro-processor in a time period which varies according to the specific instruction, to whether or not it is indirect, and to whether or not the skip condition has been met.

Upon completion of the execution of each instruction, the program counter (P register) has been incremented by one except for instructions JMP, JSM, and the skip instructions in which the skip condition has been met. The M-register is left with contents identical to the P-register. The contents of the addressed memory location and the A and B registers are left unchanged unless specified otherwise.

Memory Reference Group

The 14 memory reference instructions refer to the specific address in memory determined by the address field <*m*>, by the ZERO/CURRENT page bit, and by the DIRECT/INDIRECT bit. Page addressing and indirect addressing are both described in detail in the reference manuals for the Hewlett-Packard Model 2116 computer (hereinafter referred to as the HP 2116).

The address field <*m*> is a 10 bit field consisting of bits 0 through 9. The ZERO/CURRENT page bit is bit 10 and the DIRECT/INDIRECT bit is bit 15, except for reference to the A or B register in which case bit 8 becomes the DIRECT/INDIRECT bit. An indirect reference is denoted by a <, I> following the address <*m*>.

REGISTER REFERENCE OF A OR B REGISTER: If the location <A> or <B> is used in place of <*m*> for any memory reference instruction, the instruction will treat the contents of A or B exactly as it would be contents of location <*m*>. See the note below on the special restriction for direct register reference of A or B.

ADA *m*,I Add to A. The contents of the addressed memory location m are added (binary add) to contents of the A register, and the sum remains in the A register. If carry occurs from bit 15, the E register is loaded with 0001, otherwise E is left unchanged.

ADB *m*,I Add to B. Otherwise identical to ADA.

CPA *m*,I Compare to A and skip if unequal. The contents of the addressed memory location are compared with the contents of the A register. If the two 16-bit words are different, the next instruction is skipped; that is, the P and M registers are advanced by two instead of one. Otherwise, the next instruction will be executed in normal sequence.

CPB *m*,I Compare to B and skip is unequal. Otherwise identical to CPA.

LDA *m*,I Load into A. The A register is loaded with the contents of the addressed memory location.

LDB *m*,I Load into B. The B register is loaded with the contents of the addressed memory location.

STA *m*,I Store A. The contents of the A register are stored into the addressed memory location. The previous contents of the addressed memory location are lost.

STB *m*,I Store B. Otherwise identical to STA.

IOR *m*,I Inclusive OR to A. The contents of the addressed location are combined with the contents of the A register as an INCLUSIVE OR logic operation.

ISZ *m*,I Increment and Skip if Zero. The ISZ instruction adds ONE to the contents of the addressed memory location. If the result of this operaion is ZERO, the next instruction is skipped, that is, the P and M registers are advanced by TWO instead of ONE. The incremental value is writted back into the addressed memory location. Use of ISZ with the A or B register is limited to indirect reference; see footnote on restrictions.

AND *m*,I Logical AND to A. The contents of the addressed location are combined with the contents of the A register as an AND logic operation.

DSZ *m*,I Decrement and Skip if Zero. The DSZ instruction subtracts ONE from the contents of the addressed memory location. If the result of this operation is zero, the next instruction is skipped. The decremented value is writted back into the addressed memory location. Use of DSZ with the A or B register is limited to indirect reference; see footnote on restrictions.

JSM *m*,I Jump to Subroutine. The JSM instruction permits jumping to a subroutine in either ROM or R/W memory. The contents of the P register is stored at the address contained in location 1777 (stack pointer). The contents of the stack pointer is incremented by one, and both M and P are loaded with the referenced memory location.

JMP *m*,I Jump. This instruction transfers control to the contents of the addressed location. That is, the referenced memory location is loaded into both M and P registers, effecting a jump to that location.

Shift-Rotate Group

The eight shift-rotate instructions all contain a 4 bit variable shift field <*n*> which permits a shift of one through 16 bits; that is, $1 \leq n \leq 16$. If <*n*> is omitted, the shift will be treated as a one bit shift. The shift code appearing in bits 8,7,6,5 is the binary code for *n*-1, except for SAL and SBL, in which cases the complementary code for *n*-1 is used.

AAR *n* Arithmethic right shift of A. The A register is shifted right *n* places with the sign bit (bit 15) filling all vacated bit positions. That is, the *n*+1 most significant bits become equal to the sign bit.

ABR *n* Arithmetic right shift of B. Otherwise identical to AAR.

SAR *n* Shift A right. The A register is shifted right *n* places with all vacated bit positions cleared. That is, the *n* most significant bits become equal to zero.

SBR *n* Shift B right. Otherwise identical to SAR.

SAL *n* Shift A left. The A register is shifted left *n* places with the *n* least significant bits equal to zero.

SBL *n* Shift B left. Otherwise identical to SAL.

RAR *n* Rotate A right. The A register is rotated right *n* places, with bit 0 rotated around to bit 15.

RBR *n* Rotate B right. Otherwise identical to RAR.

Alter-Skip Group

The sixteen alter-skip instructions all contain a 5-bit variable skip field <*n*> which, upon meeting the skip condition, permits a relative branch to any one of 32 locations. Bits 9,8,7,6,5 are coded for positive or negative relative branching in which the number <*n*> is the number to be added to the current address, (skip in forward direction), and the number <-*n*> is the number to be subtracted from the current address, (skip in negative direction). If <*n*> is omitted, it will be interpreted as a ONE.

<*n*>=0	CODE=00000 REPEAT SAME INSTRUCTION
<*n*>=1	CODE=00001 DO NEXT INSTRUCTION
<*n*>=2	CODE=00010 SKIP ONE INSTRUCTION
<*n*>=15	CODE=01111 ADD 15 TO ADDRESS
<*n*>=-1	CODE=11111 DO PREVIOUS INSTRUCTION
<*n*>=-16	CODE=10000 SUBTRACT 16 FROM ADDRESS
<*n*>=nothing	CODE=00001 DO NEXT INSTRUCTION

The alter bits consist of bits 10 and bits 4. The letter <S> following the instruction places a ONE in bit 10 which causes the tested bit to be set after the test. Similarly the letter <C> will place a ONE In bit 4 to clear the test bit. If both a set and clear bit are given, the set will take precedence. Alter bits do not apply to SZA, SZB, SIA, and SIB.

SZA *n* Skip if A zero. If all 16 bits of the A register are zero, skip to location defined by *n*.

SZB *n* Skip if B zero. Otherwise identical to SZA.

RZA *n* Skip if A not zero. This is a Reverse Sense skip of SZA.

RZB *n* Skip if B not zero. Otherwise identical to RZA.

SIA *n* *S*kip if A zero; then increment A. The A register is tested for zero, then incremented by one. If all 16 bits of A were zero before incrementing, skip to location defined by *n*.

SIB *n* Skip if B zero; then increment B. Otherwise identical to SIA.

RIA *n* Skip if A not zero; then increment A. This is a Reverse Sense skip of SIA.

RIB *n* Skip if B not zero; then increment B. Otherwise identical to RIA.

SLA *n*,S/C Skip if Least Significant bit of A is zero. If the least significant bit (bit 0) of the A register is zero, skip to location defined by *n*. If either S or C is present, the test bit is altered accordingly after test.

SLB *n*,S/C Skip if Least Significant bit of B is zero. Otherwise identical to SLA.

SAM *n,*S/C Skip if A is Minus. If the sign bit (bit 15) of the A register is a ONE, skip to location defined by *n*. If either S or C is present, bit 15 is altered after the test.

SBM *n*,S/C Skip if B is Minus. Otherwise identical to SAM.

SAP *n*,S/C Skip if A is Positive. If the sign bit (bit 15) of the A register is a ZERO, skip to location defined by *n*. If either S or C is present, bit 15 is altered after the test.

SBP *n*,S/C Skip if B is Positive. Otherwise identical to SAP.

SES *n*,S/C Skip if Least Significant bit of E is Set. If bit 0 of the E register is a ONE, skip to location defined by *n*. If either S or C is present, the entire E register is set or cleared respectively.

SEC *n*,S/C Skip if Least Significant bit of E is Clear. If bit 0 of the E register is a ZERO, skip

to location defined by *n*. If either S or C is present, the entire E register is set or cleared respectively.

Complement-Execute-DMA Group.

These seven instructions include complement operations and several special-purpose instructions chosen to speed up printing and extended memory operations.

CMA Complement A. The A register is replaced by its One's complement.

CMB Complement B. The B register is replaced by its One's complement.

TCA Two's Complement A. The A register is replaced by its One's Complement and incremented by one.

TBC Two's complement B. The B register is replaced by its One's Complement and incremented by one.

EXA Execute A. The contents of the A register are treated as the current instruction, and executed in the normal manner. The A register is left unchanged unless the instruction code causes A to be altered.

EXB Execute B. Otherwise identical to EXA.

DMA Direct Memory Access. The DMA control in Extended Memory is enabled by setting the indirect bit in M and giving a WTM instruction. The next ROM clock transfers A→M and the following two cycles transfer B→M. ROM clock then remains inhibited until released by DMA control.

Note: **Special Restriction for Direct Register Reference of A or B**

For the five register reference instructions which involve a write operation during execution, a register reference to A or B must be restricted to an INDIRECT reference. These instructions are STA, STB, ISZ, DSZ, and JSM. A DIRECT register reference to A or B with these instructions may result in program modification. (This is different from the hp 2116 in which a memory reference to the A or B register is treated as a reference to locations 0 or 1 respectively.) A reference to location 0 or 1 will actually refer to locations 0 or 1 in Read Only Memory.

Input/Output Group (IOG)

The eleven IOG instructions, when given with a select code, are used for the purpose of checking flags, setting or clearing flag and control flip-flops, and transferring data between the A/B registers and the I/O register.

STF <SC> Set the flag. Set the flag flip-flop of the channel indicated by select code <SC>.

CLF <SC> Clear the flag flip-flop of the channel indicated by select code <SC>.

SFC <SC> Skip if flag clear. If the flag flip-flop is clear in the channel indicated by <SC>, skip the next instruction.

SFS <SC> H/C Skip if flag set. If the flag flip-flop is set in the channel indicated by <SC>, skip the next instruction. H/C indicates if the flag flip-flop should be held or cleared after executing SFS.

CLC <SC> H/C Clear control. Clear the control flip-flop in the channel indicated by <SC>. H/C indicates if the flag flip-flop should be held or cleared after executing CLC.

STC <SC> H/C Set Control. Set the control flip-flop in the channel indicated by <SC>. H/C indicates if the flag flip-flop should be held or cleared after executing STC.

OT* <SC> H/C Output A or B. Sixteen bits from the A/B register are output to the I/O register.

H/C allows holding or clearing the flag flop after execution of OT*. The different select codes allow different functions to take place after loading the I/O register.

SC=00 Data from the A or B register is output eight bits at a time for each OT* instruction given. The A or B register is rotated right eight bits.

SC=01 The I/O register is loaded with 16 bits from the A/B registers.

SC=02 Data from the A/B register is output one bit at a time for each OT* instruction for the purpose of giving data to the Magnetic Card Reader. The I/O register is unchanged.

SC=04 The I/O register is loaded with 16 bits from the A/B register and the control flip flop for the printer is then set.

SC=08 The I/O register is loaded with 16 bits from the A/B register and the control flip flop for the display is then set.

SC=16 The I/O register is loaded with 16 bits from the A/B register and then data in the I/O register is transferred to the switch latches.

LI* <01> H/C Load into A or B. Load 16 bits of data into the A/B register from the I/O register. H/C allows holding or clearing the flag flop after LI* has been executed.

LI* <00> The least significant 8 bits of the I/O register are loaded into the most significant locations in the A or B register.

MI* <01> H/C Merge into A or B. Merge 16 bits of data into the A/B register from the I/O register by inclusive or. H/C allows holding or clearing the flag flop after MT* has been executed.

MI* <00> The least significant 8 bits of the I/O register are combined by inclusive OR with the least significant 8 bits of the A or B register, and rotated to the most significant bit locations of the A or B register.

Mac instruction Group

A total of 16 MAC instructions are available for operation
a. with the whole floating-point data (like transfer, shifts, etc), or
b. with two floating-point data words to speed up digit and word loops in arithmethic routines.

NOTE: $<A_{0-3}>$ means: contents of A-register bit 0 to 3

AR1 is a mnemonix for arithmetic pseudo-register located in R/W memory on addresses 1744 to 1747 (octal)

AR2 is a mnemonix for arithmethic pseudo-register located in R/W memory on addresses 1754 to 1757 (octal)

D_i means: mantissas i-th decimal digit;
most significant digit is D1
least significant digit is D12
decimal point is located between D1 and D2

Every operation with mantissa means BCD-coded decimal operation.

RET Return

16-bit-number stored at highest occupied address in stack is transferred to P- and M-registers. Stack pointer (=next free address in stack) is decremented by one. <A>, <B>, <E> unchanged.

MOV Move overflow

The contents of E-register is transferred to A_{0-3}. Rest of A-register and E-register are filled by

zeros. <B> unchanged.

CLR Clear a floating-point data register in R/W memory on location <A>
Zero→<A>, <A>+1, <A>+2, <A>+3
<A>, <B>, <E> unchanged

EXF Floating-point data transfer in R/W memory from location <A> to location <B>.
Routine starts with exponent word transfer.
Data on location <A> is unchanged.
<E> unchanged.

MRX AR1 Mantissa is shifted to right *n*-times. Exponent word remains unchanged.
$<B_{0-3}> = n$ (binary coded)
1st shift: $<A_{0-3}> \rightarrow D_1$; $D_i \rightarrow D_{i+1}$; D_{12} is lost
*j*th shift: $0 \rightarrow D_1$; $D_i \rightarrow D_{i+1}$; D_{12} is lost
*n*th shift: $0 \rightarrow D_1$; $D_i \rightarrow D_{i+1}$; $D_{12} \rightarrow A_{0-3}$
$0 \rightarrow E, A_{4-15}$
each shift: $<B_{0-3}> - 1 \rightarrow B_{0-3}$
$<B_{4-15}>$ unchanged

MRY AR2 Mantissa is shifted to right *n*-times. Otherwise identical to MRX

MLS AR2 Mantissa is shifted to left once. Exponent word remains unchanged.
$0 \rightarrow D_{12}$; $D_i \rightarrow D_{i-1}$; $D_1 \rightarrow A_{0-3}$
<B> unchanged

DRS AR1 Mantissa is shifted to right once Exponent word remains unchanged
$0 \rightarrow D_1$; $D_i \rightarrow D_{i+1}$; $D_{12} \rightarrow A_{0-3}$
$0 \rightarrow E$ and A_{4-15}
<B> unchanged

DLS AR1 Mantissa is shifted to left once. Exponent word remains unchanged.
$<A_{0-3}> \rightarrow D_{12}$; $D_i \rightarrow D_{i-1}$; $D_1 \rightarrow A_{0-3}$
$0 \rightarrow E, A_{4-15}$
<B> unchanged

FXA Fixed-point addition Mantissas in pseudo-registers AR2 and AR1 are added together and result in placed into AR2. Both exponent words remain unchanged. When overflow occurs 0001 is set into E-reg., in opposite case <E> will be zero.
<AR2> + <AR1> + DC → AR2
DC = 0 if <E> was 0000 before routine execution
DC = 1 if <E> was 1111 before routine execution
<B>, <AR1> unchanged

FMP Fast multiply
Mantissas in pseudo-registers AR2 and AR1 are added together $<B_{0-3}>$-times and result is placed into AR2. Total decimal overflow is placed to A_{0-3}. Both exponent words remain unchanged.
<AR2> + <AR1> * $<B_{0-3}>$+DC → AR2
DC = 0 if <E> was 0000 before routine execution
DC = 1 if <E> was 1111 before routine execution
$0 \rightarrow E, A_{4-15}$
<AR1> unchanged

FDV Fast divide
Mantissas in pseudo-registers AR2 and AR1 are added together so many times until first decimal overflow occurs. Result is placed into AR2. Both exponent words remain unchanged. Each

addition without overflow causes +1 increment of <B>.
1st addition: <AR2> + <AR1> + DC → AR2
DC = 0 if <E> was 0000 before routine execution
DC = 1 if <E> was 1111 before routine execution
next additions: <AR2> + <AR1> → AR2
0 → E
<AR1> unchanged

CMX 10's complement of AR1 mantissa is placed back to AR1, and ZERO is set into E-register. Exponent word remains unchanged
<B> unchanged

CMY 10's complement of AR2 mantissa.
Otherwise identical to CMY

MDI Mantissa decimal increment.
Mantissa on location <A> is incremented by decimal ONE on D_{12} level, result is placed back into the same location, and zero is set into E-reg.
Exponent word is unchanged.
When overflow occurs, result mantissa will be 1,000 0000 0000 (dec)
and 0001 (bin) will be set into E-reg.
<B> unchanged.

NRM Normalization
Mantissa in pseudo-register AR2 is rotated to the left to get $D_1 \neq 0$. Number of these 4-bit left shifts is stored in B_{0-3} in binary form ($<B_{4-15}>=0$)
when $< B_{0-3} > = 0,1,2,. . . . , 11$ (dec) → <E> = 0000
When $< B_{0-3} > = 12$ (dec) → mantissa is zero, and <E>= 0001
Exponent word remains unchanged
<A> unchanged.

The binary codes of all of the above instructions are listed in the following coding table, where * implies the A or B register, D/I means direct/indirect, A/B means A register/B register, Z/C means zero page (base page) / current page, H/S means hold test bit/set test bit, and H/C means hold test bit/clear test bit. D/I, A/B, Z/C, H/S and H/C are all coded as 0/1.

CODING TABLE

GROUP	OCTAL	INSTR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEMORY	-0----	AD*	D/I	0	0	0	A/B	Z/C	←			MEMORY ADDRESS						→
REFERENCE	-1----	CP*	D/I	0	0	1	A/B	Z/C										
GROUP	-2----	LO*	D/I	0	1	0	A/B	Z/C										
	-3----	ST*	D/I	0	1	1	A/B	Z/C										
	-4----	IOR	D/I	1	0	0	0	Z/C										
	-4----	ISZ	D/I	1	0	0	1	Z/C										
	-5----	AND	D/I	1	0	1	0	Z/C										
	-5----	DSZ	D/I	1	0	1	1	Z/C										
	-6----	JSM	D/I	1	1	0	0	Z/C										
	-6----	JMP	D/I	1	1	0	1	Z/C										
SHIFT-	07---0	A*R	0	1	1	1	A/B	-		-	←SHIFT CODE→				0	0	0	0
ROTATE	07---2	S*R	0	1	1	1	A/B	-		-					0	0	1	0
GROUP	07---4	S*L	0	1	1	1	A/B	-		-					0	1	0	0
	07---6	R*R	0	1	1	1	A/B	-		-					0	1	1	0
ALTER-	07---0	SZ*	0	1	1	1	A/B	0	← SKIP CODE →					0	1	0	0	0
SKIP	07---0	RZ*	0	1	1	1	A/B	1						0	1	0	0	0
GROUP	07---0	SI*	0	1	1	1	A/B	0						1	1	0	0	0
	07---0	RI*	0	1	1	1	A/B	1						1	1	0	0	0
	07---1	SL*	0	1	1	1	A/B	H/S						H/C	1	0	0	1
	07---2	S*M	0	1	1	1	A/B	H/S						H/C	1	0	1	0
	07---3	S*P	0	1	1	1	A/B	H/S						H/C	1	0	1	1
	07---4	SES	0	1	1	1	A/B	H/S						H/C	1	1	0	0
	07---5	SEC	0	1	1	1	A/B	H/S						H/C	1	1	0	1
D/I	07--17	ADA	0	1	1	1	A/B	-	-	D/I	0	0	0	0	1	1	1	1
REFERENCE	07--37	ADB	0	1	1	1	A/B	-	-	D/I	0	0	0	1	1	1	1	1
GROUP	07--57	CPA	0	1	1	1	A/B	-	-	D/I	0	0	1	0	1	1	1	1
	07--77	CPB	0	1	1	1	A/B	-	-	D/I	0	0	1	1	1	1	1	1
	07--17	LDA	0	1	1	1	A/B	-	-	D/I	0	1	0	0	1	1	1	1
	07--37	LDB	0	1	1	1	A/B	-	-	D/I	0	1	0	1	1	1	1	1
	07-557	STA	0	1	1	1	A/B	-	-	1	0	1	1	0	1	1	1	1
	07-577	STB	0	1	1	1	A/B	-	-	1	0	1	1	1	1	1	1	1
	07--17	IOR	0	1	1	1	A/B	-	-	D/I	1	0	0	0	1	1	1	1
	07-637	ISZ	0	1	1	1	A/B	-	-	1	1	0	0	1	1	1	1	1
	07--57	AND	0	1	1	1	A/B	-	-	D/I	1	0	1	0	1	1	1	1
	07-677	DSZ	0	1	1	1	A/B	-	-	1	1	0	1	1	1	1	1	1
	07-717	JSM	0	1	1	1	A/B	-	-	1	1	1	0	0	1	1	1	1
	07--37	JMP	0	1	1	1	A/B	-	-	D/I	1	1	0	1	1	1	1	1
COMP	07-016	EX*	0	1	1	1	A/B	-	-	-	-	-	0	0	1	1	1	0
EXECUTE	070036	DMA	0	1	1	1	0	-	-	-	-	-	0	1	1	1	1	0
DMA	07-056	CM*	0	1	1	1	A/B	-	-	-	-	-	1	0	1	1	1	0
	07-076	TC*	0	1	1	1	A/B	-	-	-	-	-	1	1	1	1	1	0
INPUT	1727--	STF	1	1	1	1	-	1	0	1	1	1	1	← SELECT CODE →				
OUTPUT	1737--	CLF	1	1	1	1	-	1	1	1	1	1	1					
GROUP	17-7--	SFC	1	1	1	1	-	1	H/C	1	1	1	0					
	17-5--	SFS	1	1	1	1	-	1	H/C	1	0	1	0					
	17-5--	CLC	1	1	1	1	-	1	H/C	1	0	1	1					
	17-6--	STC	1	1	1	1	-	1	H/C	1	1	0	0					
	17-1--	OT*	1	1	1	1	A/B	1	H/C	0	0	1	1					
	17-2--	LI*	1	1	1	1	A/B	1	H/C	0	1	0	1					
	17-0--	MI*	1	1	1	1	A/B	1	H/C	0	0	0	1					

GROUP	OCTAL	INSTR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAC	170402	RET	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0
GROUP	170002	MOV	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0
	170000	CLR	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	170004	XFR	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0
	174430	MRX	1	1	1	1	1	0	0	1	0	0	0	1	1	0	0	0
	174470	MRY	1	1	1	1	1	0	0	1	0	0	1	1	1	0	0	0
	171400	MLS	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0
	170410	DRS	1	1	1	1	0	0	0	1	0	0	0	0	1	0	0	0
	175400	DLS	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0
	170560	FXA	1	1	1	1	0	0	0	1	0	1	1	1	0	0	0	0
	171460	FMP	1	1	1	1	0	0	1	1	0	0	1	1	0	0	0	0
	170420	FDV	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	0
	174400	CMX	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0
	170400	CMY	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0
	170540	MDI	1	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0
	171450	NRM	1	1	1	1	0	0	1	1	0	0	1	0	1	0	0	0

The following describes the function of each I/O instruction with the 5 allowable select codes.

```
STF <SC> Set the flag. STF is a 240 nanosecond
         positive true pulse which accomplishes the
         following with the various select codes.
STF 00   Not used by the calculator.
STF 01   a. Sets the "Service Inhibit" flip-flop
            to the true state (SIH = Low; interrupt
           not allowed).
         b. Causes parallel input data and status
            to be loaded into the I/O register.
STF 02   Generates a 240 nanosecond positive true MCR
         pulse.
STF 04,08,16  Not used by the calculator.
CLF <SC> Clear the flag. CLF is a 240 ns positive
         true pulse which accomplishes the follow-
         ing with the various select codes.
CLF 00   Not used by the calculator
CLF 01   a. Clears the "Service Inhibit" flip-flop
            to the false state. (SIH = High;
            interrupt allowed.)
         b. Loads address locations in I/O register
            gister with 0's. (0 = Low)
         c. Clears "Device Ready" flip-flop (CEO =
            High).
CLF 02   Clears MCR flag flip-flop.
CLF 04   Clears PEN flip-flop (PEN = Low).
CLF 08   Clears DEN flip-flop (DEN = High).
CLF 16   Generates a 240 nanosecond positive true
         KLS pulse
SFC <SC> H/C  Skip if flag clear. SFC is a 240 ns
         positive true pulse which accomplishes
         the following with the various select
         codes. If C is given a 240 nanosecond CLF
         pulse is given after SFC.
SFC 00   Causes the next instruction to be
         skipped if the STOP key has not been
         depressed.
SFC 01   Causes the next instruction to be skipped
```

if Device Ready is true ($\overline{CEO}$ = Low).

SFC 02 Causes the next instruction to be skipped if the MCR flag flip-flop is clear.

SFC 04 Causes the next instruction to be skipped if the PEN flip-flop is clear. ($\overline{PEN}$ = Low).

SFS <SC> H/C Skip is flag set. SFS is a 240 nanosecond positive true pulse which accomplishes the following with the various codes. If C is given then a 240 nanosecond CLF Pulse is issued after SFS.

SFS 00 Causes the next instruction to be skipped if the STOP key is depressed.

SFS 01 Causes the next instruction to be skipped if "Device Ready" is false ($\overline{CEO}$ = High).

SFS 02 Causes the next instruction to be skipped if the MCR flag flip-flop is set.

SFS 04 Causes the next instruction to be skipped if the PEN flip-flop is set ($\overline{PEN}$ = High).

CLC <SC> H/C Clear Control. $\overline{CLC}$ is a 240 nanosecond negative true pulse and is not used by the calculator. If C is given then a 240 nanosecond positive true CLF pulse is given after CLC.

STC <SC> H/C Set the Control. STC is a 240 nanosecond positive true pulse which accomplishes the following with the various select codes. If C is given a 240 nanosecond CLF pulse is issued after STC.

STC 00 Not used by the calculator.

STC 01 Sets the "Device Ready" flip-flop ($\overline{CEO}$ = Low).

STC 02 Generates a 240 nanosecond positive true MLS pulse for the magnetic card reader.

STC 04,08,16 Not used by the calculator.

OTX <SC> H/C Output A or B causes data bits from A or B to be shifted to the I/O register and accomplishes the following with the various select codes. If C is given, a 240 nanosecond CLF pulse is given after OTX is executed.

OTX 00 The 8 least significant bits in the A or B register are shifted non-inverted to the 8 least significant locations in the I/O register, and 120 nanosecond after the 8th shift the "Device Ready" flip-flop is set ($\overline{CEO}$ = Low). The 8 most significant bits are shifted right 8 places and the least 8 significant bits are recirculated to the 8 most significant locations in the A or B registers. The 8 most significant bits in the I/O register are untouched.

OTX 01 Sixteen bits from the A or B register are shifted non-inverted to the I/O register. The data in A or B recirculates.

OTX 02 Not used by the calculator

OTX 04 Same as OTX 01 and in addition, 120 ns after the 16th bit has been shifted nanoseconds printer enable flip-flop is set

OTX 03 Same as OTX 01 and in addition, 120 nanoseconds after the 16th bit has been shifted the display enable flip-flop is set.

```
OTX 16   Same as OTX 01 and in addition, 120 nanoseconds
         after the 16th bit has been shifted the
         240 nanosecond KLS signal is generated
LIX <SC> H/C  Load into A or B. Loads data bits from
         the I/O register into the A or B register
         and accomplishes the following with the
         various select codes. If C is given, a
         240 nanosecond CLF pulse is given after LIX is
         executed.
LIX 00   The eight least significant bits in the
         I/O register are shifted inverted to the
         eight most significant locations of A or
         B, and 120 nanoseconds after the 8th shift the
         "Device Ready" flip-flop is set (CEO = Low).
         A or B is shifted right eight places as
         the I/O register data comes in. The 8
         most significant bits in the I/O register
         are untouched.
LIX 01   The 16 bits of the I/O register are trans-
         ferred inverted to the A or B register.
         Data in the I/O register is lost.
LIX 02,04,08,16  Not used by the calculator.
MIX <SC> H/C  Merge into A or B. Merges data from the
         I/O register into A or B registers and
         accomplishes the following with various
         select codes. If C is given, a 240 nanosecond
         CLF pulse is given after MIX is executed.
MIX 00   The eight least significant bits in the
         I/O register are merged with the eight
         least significant bits of the A or B
         register and shifted to the 8 most signi-
         ficant locations of A or B; 120 nanosecond
         after the merge takes place the Device Ready
         flip-flop is set (CEO = Low). A or B
         shifts right 8 places as the data is
         merged and shifted to the most significant
         locations. The 8 most significant bits
         of the I/O register are untouched.
MIX 01   The 16 bits of the I/O register are
         merged with the 16 bits of the A or B
         register and contained in the A or B
         register.
MIX 02,04,08,16  Not used by the calculator.
```

B.1 Some Remarks on the Instruction Set

Some of the above I/O machine instructions are marked as 'not used by the calculator'. Careful study of the CPU and I/O-register schematics shows that for the affected instructions resp. select codes there is no decoding or logic implementation. So in fact the unused instructions are not functional. This is also true for the emulator.

The instruction DMA (op-code 070036) is decoded by the micro-program but the actual DMA-logic mentioned in the instruction description is not implemented in the calculators hardware. So in effect the instruction DMA does nothing, unless the necessary logic is implemented by an external device, connected by the I/O-bus. For this several signals are put on the I/O-connector backplane, such as M15 (bit 15 of M-register), MTS (M-register to S-bus), SCK (shift-clock), RDM (read memory), and WTM (write memory).

C Using the GO9800 Console

The emulator includes a console which allows inspection of the CPU registers and analysis of machine programs (either in ROM or RWM) at run-time. Additionally there is a special key-log function which allows to conveniently create key assignment configuration files.

C.1 Disassembler Functions

The disassembler may be controlled in a separate console dialog window. The console can be opened from any of the emulated calculator models by pressing the keys Ctrl+D. The dialog is also automatically opened, when a breakpoint or watchpoint condition is met (chapter 1.1.8). It can be closed by the window close icon or by again pressing the keys Ctrl+D when the calculator has input focus.

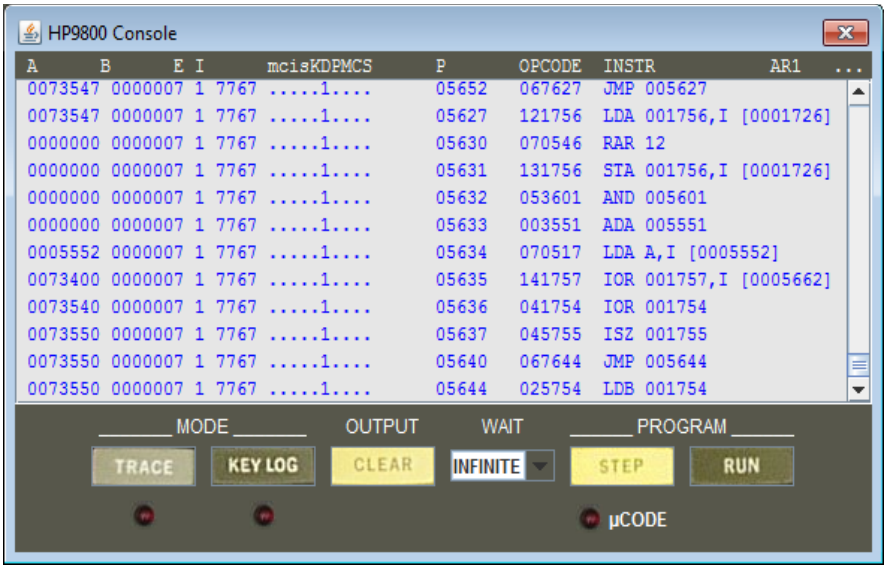


Figure C-1: The console dialog.

In the dialog window each diassembled CPU instruction is output *before* its execution together with the current contents of the following registers and flags in one text line:

- CPU registers A and B (16 bit octal values)
- Extend register E (4 bit hexadecimal value)
- I/O register (16 bit hexadecimal value)
- I/O flags (1 bit value, either 1 or 0, symbolized by a dot):
  - m (MLS): magnetic card reader control
  - c (MCR): magnetic card reader output
  - i (SIH): service inhibit
  - s (SSF): single service flip-flop
  - K (KLS): keyboard LEDs enable

- D (DEN): display enable
- P (PEN): printer enable
- M (MFL): magnetic card input flag
- C (CEO): device control enable output
- S (STP): stop flag

- Program counter PC (16 bit octal value)
- Instruction OPCODE (16 bit octal value)
- Disassembled instruction

See appendix B for a detailed description of the CPU instructions. All memory addresses are given as octal values, relative branch values are given as offset to the program counter (*+n or *-n). For Instructions using indirect addressing (e.g. LDA B,I) the computed effective address of the indirection is given as octal value in square brackets at right of the instruction.

Additionally for BCD arithmetic instructions the contents of the two pseudo registers AR1 (memory address 01744-01747) and AR2 (address 01754-01757) are shown at right of the instruction. They can be made visible by enlarging the disassembler window. Values of AR1 and AR2 are given as four hexadecimal 16-bit words and are shown before and after execution of any BCD instruction, so one can see the computation result.

Figure C-2: Disassembler output showing the arithmetic pseudo registers.

C.2 Usage of the Disassembler

C.2.1 Online Disassembler Mode

The disassembler may be started by clicking on the TRACE button on the left side. Since the output of each executed instruction takes a certain amount of time, the speed of the calculator is slowed down significantly. Clicking again on on the TRACE stops the online disassembler mode.

The disassembled instructions can be inspected using the scroll bar on the right side. There are a maximum of 4096 lines kept in the window output buffer. Output of one additional line at the end results in deletion of the first line. The content of the buffer may be marked by mouse-dragging and copied and pasted in a text editor for later analysis.

The output buffer may be completely emptied using the CLEAR button.

C.2.2 Single Step Mode

It is also possible to step through a machine program, either manually or automatically at a selectable speed. Clicking on the STEP button halts the normal execution and switches the emulator to the single step mode. The next machine instruction is disassembled but the execution is delayed until either STEP is pressed again or the number of milliseconds selected in the drop-down list has elapsed. The default behaviour is infinite waiting for pressing of the STEP key. This may changed by selecting one of alternative values in the drop-down list or by keying in an arbitrary integer value.

The single step mode may be cancelled and normal execution resumed by clicking the RUN button.

The single step mode is automatically activated when a breakpoint or watchpoint condition is met.

C.2.3 Breakpoints

Breakpoints are locations in a machine program at which the normal execution is automatically halted and the disassembler single-step mode is entered. They can be defined in the calculator configuration file (see chapter 1.1.8).

Breakpoints can be set for any valid memory address. They become effective only when the program counter reaches the breakpoint address, not when the corresponding memory location is read or written by a CPU instruction.

C.2.4 Watchpoints

Watchpoints are locations in the calculator memory which are continuously monitored for any read or write access by a CPU instruction. There are two types of breakpoints: conditional and unconditional ones. If a memory location is accessed and there is a unconditional watchpoint defined, the disassembler single-step mode is activated. The watchpoint address and the current content of the memory location are output to the disassembler window.

If a memory location is accessed and there is a conditional watchpoint defined, the current content of that memory location is compared with a test value by a comparison operator (=, <, or >). If the result is logically true, the single-step mode is activated.

Note: A watchpoint is also effective if the CPU program counter reaches the watchpoint address.

C.3 Key-Log Mode

To assist in creation of keyboard configuration files the console implements a key-log mode with which every calculator and PC keystroke is recorded in the console log. To open the Console window press Ctrl+D.

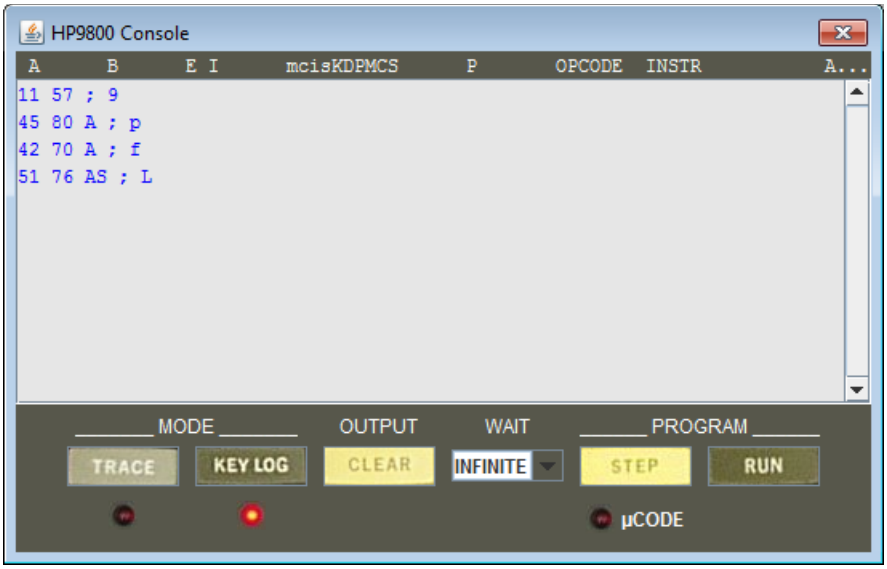


Figure C-3: Console with key-log mode activated.

To activate this mode press the KEY LOG button. The LED below this button is lit when key-log is active. In key-log mode all keys pressed on the PC keyboard and all mouse clicks on calculator keys are not sent to the calculator but their corresponding key codes are output in the console window.

C.3.1 Creating Keyboard Configuration Files

Before creating a new keyboard configuration the console window should be cleared using the CLEAR button. Click on the calculator window to give it the input focus.

To record one complete calculator / PC key assignment proceed as follows.

1. Decide which calculator key has to be assigned.
2. Mouse-click on the the calculator key. The corresponding octal key code is displayed in a new line.
3. Press the desired PC key together with the necessary modifier key(s) (Shift, Alt, Ctrl). The corresponding decimal key code and modifier shortcuts (S, A, C) are displayed on the same line. The equivalent ASCII character is added as a comment.
4. Mistakes can be corrected directly in the console output by mouse-clicking in the line to be corrected and using the common editor and text keys. Detailed comments maybe typed behind the semicolon.
5. Repeat from step 1 until all desired calculator keys are assigned.

Finally open an existing or new text file (e.g. the existing keyboard configuration of that calculator) in a text editor of your choice. Then, in the console output, mark all key assignment lines using the mouse or cursor keys, copy them to the clipboard (Ctrl+C), and paste them to the open text file. The text file should now be saved with the name <Model>-keyb.cfg in the emulator installation folder.

C.4 Micro-Code Output

Since release 2.0 the console comprises a mode switch for displaying decoded micro-

instructions. The micro-code mode is enabled by clicking on the µCODE text label or the LED-symbol left of it. This mode works only in combination with the trace or single-step disassembler mode.

When active, the head line of the console window changes to show more columns. During execution of any CPU instruction the associated micro-instructions are decoded and the following values are output to the console window:

- CPU arithmetic registers A and B (16 bit octal values)
- Extend register E (4 bit hexadecimal value)
- Memory address register M (16 bit octal value)
- Memory data register T (16 bit octal value)
- Qualifier register Q (16 bit octal value)
- Program counter register P (16 bit octal value)
- I/O register (16 bit hexadecimal value)
- I/O flags (1 bit value, either 1 or 0, symbolized by a dot):
  - m (MLS): magnetic card reader control
  - c (MCR): magnetic card reader output
  - i (SIH): service inhibit
  - s (SSF): single service flip-flop
  - q (SRQ): service request
  - K (KLS): keyboard LEDs enable
  - D (DEN): display enable
  - P (PEN): printer enable
  - M (MFL): magnetic card input flag
  - C (CEO): device control enable output
  - S (STP): stop flag
- AR1: BCD arithmetic register 1 (four 16 bit hexadecimal values)
- AR2: BCD arithmetic register 2 (four 16 bit hexadecimal values)
- Instruction OPCODE (16 bit octal value)
- CPU flip-flops (1 bit value, either 1 or 0, symbolized by a dot):
  - b (BC): binary carry flip-flop
  - d (DC): decimal (BCD) carry flip-flop
- A or B register select flip-flop x, used in TTX and XTR micro-operations
- Primary and secondary micro-program address (upper and lower 4 bits) PASA (2 x 2 digit octal value)
- Next micro-program address (upper and lower 4 bits) NA (2 x 2 digit octal value)
- Branch condition BRC (designator of Q-register bit or other qualifiers)
- Shift-clock inhibit condition IQN (designator of Q-register bit or other qualifiers)

- Shift-clock count minus one C (1 digit hexadecimal value)
  - an asterisk * right of the clock count denotes an optimized shift cycle (see D.6)
- 'A/B-register to R-bus' micro-instruction XTR
- R-code micro-instruction RC (R-bus instruction mnemonic)
- S-code micro-instruction SC (S-bus instruction mnemonic)
- X-code micro-instruction XC (miscellanous instruction mnemonic)
- 'T-bus to M-register' micro-instruction TTM
- 'T-bus to T-register' micro-instruction TTT
- ALU-code micro-instruction AC (ALU or BCD instruction mnemonic)

Before execution, the disassembled CPU instruction together with a leading '>' and the program-counter in column 'A' and op-code in column 'B' is output to the console window.

Figure C-4: Console with micro-code and single-step mode activated.

C.5 CPU Initialization and Diagnostics

After startup of the emulator the CPU is initialized by starting the micro-program at address 1717 and executing the 'diagnostic' routine (see also D.2.2). This process is automatically logged to the console. When opening the console window for the first time, one can see these decoded micro-instructions. What the diagnostic routine does can be seen in the console output and is the following:

- The registers A, M, T, B, P, Q and E in this sequence are filled with 0 (zero).
- This is done by using the ALU operations AND, IOR, XOR, ADD, and CBC. By doing this the ALU ROM and BC flip-flop are checked.
- The register and shift logic is checked.
- If any of these registers fails (by having one or more bits constantly 1), the binary carry BC is set and execution branches to the DMA routine.

- Since the DMA logic is not implemented in the basic machine (see remarks in B.1) the execution is then continued normally.
- In case of a real failure in one of the registers the machine will propably hang or loop somewhere in the micro-program.

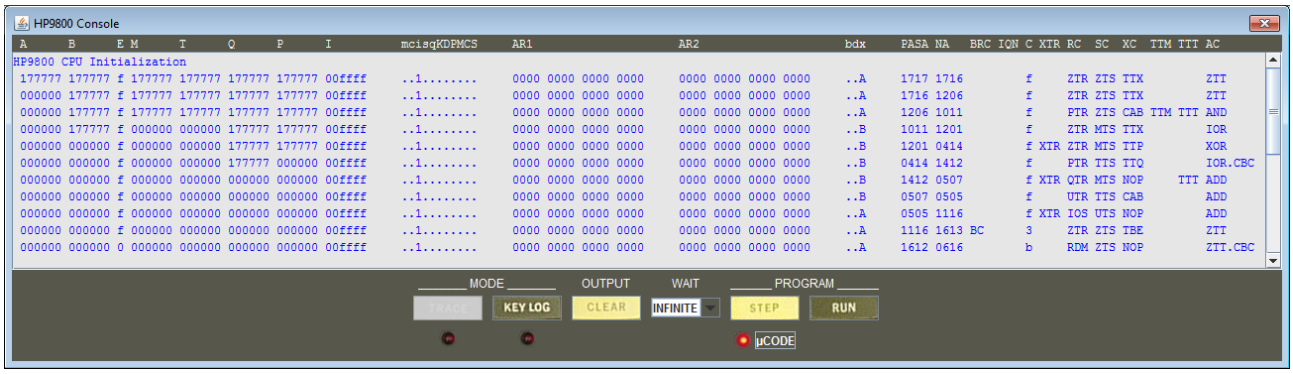


Figure C-5: Console output of CPU initialization and diagnostic routine.

This behaviour was evaluated by programming 'bit-failures' in one of the emulated CPU registers. Unfortunately only failures with constant 1 can be detected by the diagnostic. Constant 0 failures in one of the registers don't lead to a binary carry, but nevertheless the machine would hang up. It is interesting to find, that the complete CPU is roughly checked in only ten instructions.

More about the CPU and micro-code can be found in the following appendix D.

D The HP9800 CPU and Micro-Code

The instructions described in appendix B are executed by a CPU, which is unique to the HP9800 series and made of four printed circuit boards containing nearly 80 integrated TTL circuits. The CPU comprises two accumulator registers A and B, the extend register E, the qualifier register Q, and the program counter register P. The core computing logic is implemented in two 256x4-bit bipolar ROMs making up the arithmetic-logic-unit (ALU) for binary logic (AND, OR, XOR, ADD) and binary-coded-decimal (BCD) operations (ADD, SUB). Also part of the CPU is the I/O-unit containing an IO-register and associated logic for communication with I/O-devices.

The CPU is supported by another 60 TTL ICs on three boards making up the M-register, T-register and memory controller for interaction with the internal memory (RWM and ROM).

Decoding and execution of the CPU instructions is controlled by a micro-program, which is part of the CPU and originally stored in seven 256x4-bit bipolar ROMs. In order to execute the micro-program in GO9800, it is neccessary to emulate the CPU hardware down to single shift-registers, flip-flops and logic gates.

D.1 Architecture of the HP9800 CPU

The archicture of the CPU is described in several US-patents (see 6.1). Brent Hilpert has created a complete new set of schematics for the HP9830A which also contains the block diagram shown below.

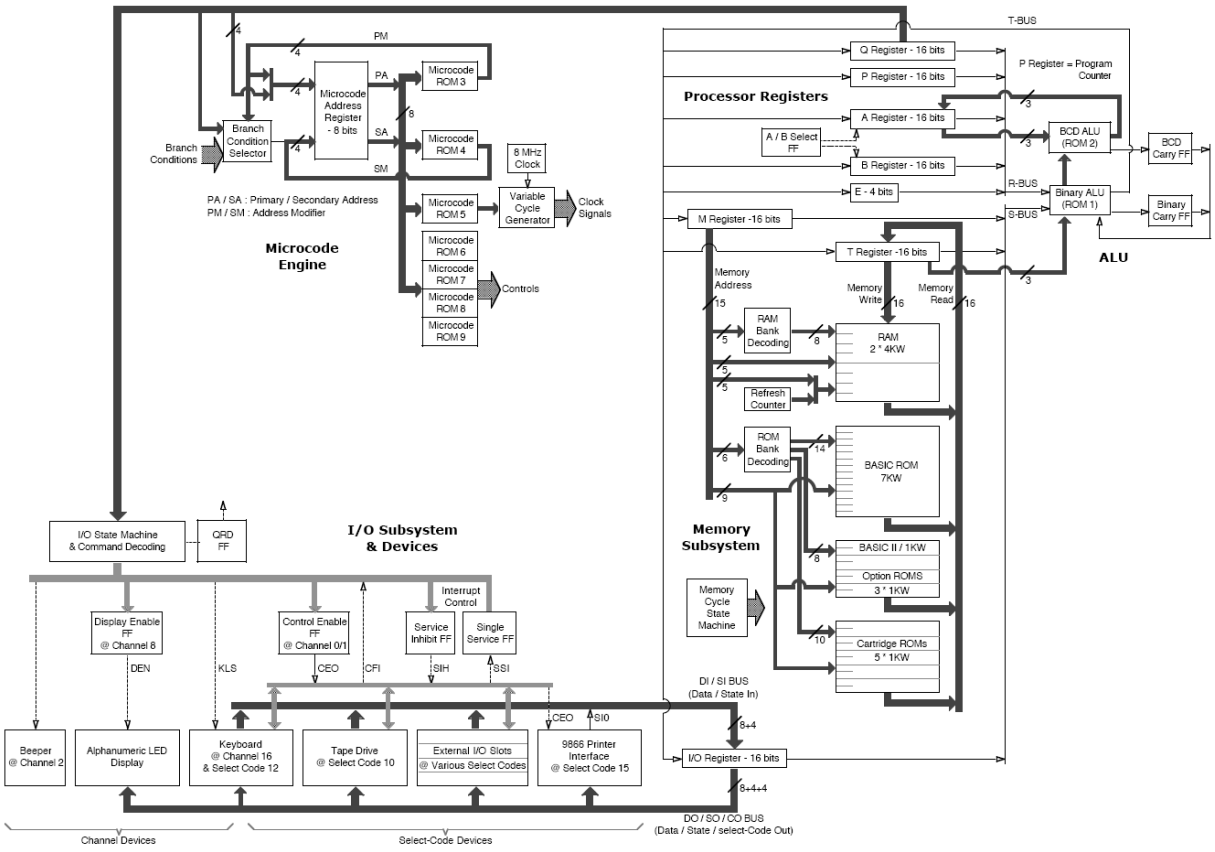


Figure D-1: Block diagram of the HP9830A with CPU, I/O-unit, and main memory (from //www.cs.ubc.ca/~hilpert/e/HP9830)

The architecture was inspired by the series HP21xx minicomputers of the late 1960s, but to reduce complexity and costs some hardware restrictions had to be implemented. To build a calculator with as little hardware as possible and to reduce the complexity of the data bus structure, the engineers decided to use bit-serial buses inside the CPU. Thus the CPU is peculiar in so far, as it has 16 bit wide operand registers but only 1 bit wide data buses and ALU functions. This of course makes it intrinsically slow, as even the simplest 16-bit operation takes several ten clock cycles for execution. This is also due to the very reduced and condensed micro-program, which is optimized to reduce hardware and not to speed.

D.1.1 CPU Registers and Data Buses

The CPU comprises four 16-bit shift-registers A, B, Q, and P and one 4-bit shift-register E which are all connected to the ALU via the R-bus and T-bus. The R-bus transports the register contents to the ALU and the T-bus back into the registers. A third data bus, the S-bus, connects the memory adress and data registers M and T as well as the IO-register to the ALU. All transfers are bit-serial with least-significant bit first.

The M-register holds the address for memory read or write operations and the T-register is used to transfer values to and from memory and as operand register for ALU operations. P is the program counter for assembler level CPU instructions.

The registers A and B are universal accumulator registers. There is a register select flip-flop which determines which of the registers A and B is used in certain micro-instructions.

The state of the flip-flop may be set ot altered by micro-operations.

The registers and buses may have several sources, from which they receive data and destinations where they send data to, shown in fig. D-1. Which register is connected to which bus is controlled by several micro-operations, as explained in D.2. In addition to the registers, the R- and S-bus may have constant 0 (Zero) and 1 (One) as input. It is important for understanding of the micro-instructions, that the R-bus and S-bus may have more than one source at a time (e.g. the A-register and Zero). In this case the sources are logically or-ed. In the hardware this mixing is done by complex AND/OR-gates.

D.1.2 Q-register and conditional execution

The 'qualifier' register Q is used by the CPU mainly for conditional instructions, branches, and as temporary storage. It is not accessible in assembler level instructions. The bits Q0 – Q6, Q8, and Q10 can be tested in micro-instructions. If the bit value is 0 (zero) branching in the micro-program may occur, when the micro-operation BRC is used. Second purpose of the qualifier bits is conditional execution of a micro-instruction, using the micro-operation IQN. If again the tested bit value is 0, the shifting of all registers involved in the instruction will be inhibited. So the instruction is effectively not executed (with some exceptions explained below). Besides the Q-register there are more qualifiers, which may be interrogated for branching and conditional shifting. These are described in D.2.1.

Branching and conditional shifting may be used in any micro-instruction. One difference between branching and conditional shifting is, that the former is executed after the current micro-instruction while the latter inhibits execution of the current instruction. Both methods may also be combined in one instruction.

The bits Q11 – Q14 are used by the micro-operation TQR for jumping inside the micro-programm (a kind of 'computed Goto'). Another pecularity of the Q-register is, that its bit 6 may be filled directly from the T-bus, without shifting of Q. This is accomplished by the micro-operation TQ6. This is used in conjuntion with conditional branching and shifting with qualifier Q6.

D.1.3 The Arithmetic Logic Unit (ALU)

The ALU executes 1-bit binary operations with values on the R- and S-bus and puts the result on the T-bus. The ADD micro-operation also uses the binary carry (BC) flip-flop as input and sets it according to the result of the addition. The ALU functions are not implemented with logic gates but as preprogrammed results in one of the ALU ROMs. This made the hardware design rather simple.

Decimal operations are executed on the lowest four bits of the registers A and T. Bits A0 and T0 are processed by the first ALU ROM and bits A1 – A3 and T1 – T3 are processed by a second BCD ROM, which also uses some outputs of ROM 1 as inputs. The combined result bits are stored back to the A-register bits A0 – A3. Instead of the binary carry a second decimal carry flip-flop is used in BCD operations.

D.1.4 The CPU Shift Cycle

Every operation of the CPU involves the R-bus, S-bus, T-bus, and the ALU. Due to the bit-serial execution of binary operations, during one clock cycle, the least significant bits (LSB) of the source registers are shifted serially to the R- and S-bus respectively, processed by the ALU, the result bit placed on the T-bus and shifted into the most significant bit (MSB) of

the destination register, of which the LSB is lost. During this, the LSB of each source register is recycled into the MSB. Thus after 16 clock cycles all bits are processed and the source registers are restored to their original value, unless a source register is also destination of the T-bus. Nevertheless there are micro-instructions which use less than 16 shift cycles and leave the source and destination registers in a somewhat 'unfinished' state (which of course is intended at this moment). Partial shifting may be required to access a certain bit for testing (see TQ6 above). As described in D.1.2, shifting and therefore instruction execution may be inhibited by means of the micro-operation IQN together with any of the 'qualifier' flags. The inhibit always affects the complete micro-instruction and doesn't occur in the middle of a shift cycle.

D.1.5 The I/O-Unit State Machine

I/O instructions are not executed directly be the micro-program but by the I/O subsystem consisting of the 'Control and System-Clock' board and the 'I/O-register and Gate Interface'. For this the instruction opcode is transfered to the Q-register and during the micro-instruction IOS the value in bits Q5-Q8 is decoded by a 1-of-16 decoder. Depending on the opcode several status may occur, which initiate further actions. Setting and testing of signal lines and flip-flop states, like STF, CLF, or SFC are executed in one state (clock cycle), while input and output operations, like OTA and LIB require 8 or 16 states (clock cycles) to shift the contents in and out of the I/O register and the source or destination register. Finally a flag may be given to the device, signalling the availability of output data or the input completion. Bits Q0-Q4 define the device select code. During an I/O instruction the micro-program loops, testing the QRD signal, which is cleared after completion of the I/O instruction (look at the right side of flow-chart 1 in D.2.3, instruction 0512 and 1207).

D.2 Micro-Operations, Micro-Instructions, and Micro-Program

The micro-program uses a set of 28 bit wide micro-instructions, each of which comprises four to six micro-operations plus address and clock values. The operations and values are coded in several, one to eight bit wide code groups in the instruction word.

Each instruction has an eight bit ROM address (0 – 255), which is devided in the four upper bits of the primary address (PA) and the lower four bits of the secondary address (SA). By convention the PA and SA are denoted by seperate two digit octal values. Thus the decimal address 255 (hexadecimal FF) is denoted as PASA = 1717.

There are three sets of 3-bit micro-operations (code groups) which control the data flow between the CPU registers, the data buses and the ALU:

- R-code (RC) operations define the source for the R-bus,
- S-code (SC) operations define the source for the S-bus,
- X-code (XC) operations define the destination register for the T-bus and some special operations, like setting of the A/B-register selector.

This is completed by three 1-bit operations which control transfer of the registers A or B to the R-bus (XTR) and of the T-bus to the registers M (TTM) and T (TTT).

A fourth 3-bit group (AC) defines the type of ALU operation.

Some combinations of XTR and R-codes, which have no practical sense, define derived operations like IOS (I/O operation). Combinations of the X-code BCD and some R- and A-codes define the operations of decimal ADD, SUB, and carry handling.

In eight more bits the address of the next micro-instruction is coded. This is used for unconditional jumping and conditional branching in the micro-program. The next adress is not coded literally but as a modifier to the actual address of the current micro-instruction. This modifier is XOR-ed with the current address to give the next address. The modifier is also devided in a 4-bit primary (PM) and a 4-bit secondary (SM) modifier. This is important to understand, since both modifiers may be independently altered by some micro-operations. In the micro-program listing below, the next address is already calculated by PASA XOR PMSM.

Program branching is indicated in one bit, where the branch condition (which of 16 qualifier bits to test) is coded in the four bits of the PM. This dual usage of the address modifier code leads to quite confusing jumps in the micro-program. If the tested qualifier is 0 (zero) branching occurs by altering the lowest bit of the secondary modifier SM to 0. After the XOR, this gives a next address with the lowest bit inverted.

Four bits of the micro-code define the number of shift-cycles (1 to 16) to be executed. As explained in D.1.4, shifting may be conditionally inhibited by the micro-operation IQN, which takes another 1-bit field in the micro-code. Again, the shift condition (qualifier bit to be tested) is defined by the primary modifier PM. If the tested qualifier is 0 (zero) shifting is inhibited and the instruction is not executed with the following exceptions:

- ALU operations with the lowest bits of S- and R-bus source registers are executed and result is put on T-bus and in the binary carry (this is relevant for I/O operatons),
- BCD operations with the lowest four bits of the A- und T-register are executed and the result is put in the lowest four bits of the A-register,
- the micro-operation TQ6 is executed (lowest bit T0 of the T-register is transfered to bit Q6 of the Q-register).

In addition, shifting in the CPU is inhibited by the micro-operation IOS, which gives control to the I/O subsystem, as explained in D.1.5. Instead, the I/O subsystem executes its own shifting of the source register via R-bus, ALU and T-bus into the I/O register. Or from the I/O register via S-bus, ALU and T-bus to the destination register. It is important to emphasize, that every input or output operation also includes an ALU operation. This feature is used by the CPU instruction MIX, which 'mixes' the contents of register A or B with the contents of the I/O register by or-ing them in the ALU. The instruction MIX A does the following:

A-register → R-bus
IO-register → S-bus
R-bus OR S-bus → T-bus
T-bus → A-register

Other I/O operations take place by or-ing the S-bus with 0, so effectively shifting the value unaltered through the ALU. The OR-operation is caused by the two micro-instructions 0512 and 1207 of the I/O wait loop (see flow-chart 1 in appendix D.2.3, on the right side) in confunction with the shift logic in the I/O unit.

By the way, it is a frequently used technique in micro-instructions to transfer values from one register to another by or-ing the source with 0 or and-ing it with 1 or adding 0 to it. This is necessary, as the only data source for a register is the T-bus and every transport over it has to pass the ALU (see fig. D-1).

D.2.1 Micro-Operation Set

The micro-operation set is divided into several groups, each of which is coded in one to three bits of the micro-code. The complete micro-instruction is merged of the bit fields of six micro-operation groups, the branch and shift-inhibit fields, and codes for primary and secondary adresss modifier and clock cycles:

Instruction = PM + SM + CC + IQN + AC + XTR + BRC + RC + XC + TC + SC

but not every possible combination is used in the actual micro-program.

There are combinations of operations, some which have no other practical sense (e.g. XTR and UTR), which define derived operations:

TQR = XTR • UTR

IOS = XTR • PTR

ETR = TBE • TRE

The operation ETR is not mentioned in the original patent, although obviously build in the CPU logic and used in a micro-instruction. The mnemonic was given by the author.

Furthermore, the micro-operation BCD, which has itself no function, works as a mode switch in ALU operations. Together with the operation UTR, which is here used only as another mode signal, the ALU has the following BCD operations:

BCD • UTR : Set decimal carry DC

BCD • $\overline{\text{UTR}}$ • ZTT : decimal add (A + T)

BCD • $\overline{\text{UTR}}$ • ADD : decimal subtract (A – T)

These operations are implemented in the ALU and BCD ROMs.

The following table lists all implemented micro-operations with their mnemonics and bit-values in the 28-bit micro-code. In this table 'x' means an arbitrary bit value.

Mnemonic	#	Micro Code	Operation
RC Group, source of R-Bus			
UTR	0	xxxx xxxx xxxx xxx1 x000 xxxx xxxx	1 → R-bus
PTR	1	xxxx xxxx xxxx xxx1 x100 xxxx xxxx	P-reg → R-bus
TRE	2	xxxx xxxx xxxx xxxx x010 xxxx xxxx	T-reg → E-reg → R-bus
WTM	3	xxxx xxxx xxxx xxxx x110 xxxx xxxx	T-reg → Memory[M-reg], 0 → R-bus
TQ6	4	xxxx xxxx xxxx xxxx x001 xxxx xxxx	T-bus → Q-reg bit 6, 0 → R-bus
QTR	5	xxxx xxxx xxxx xxxx x101 xxxx xxxx	Q-reg → R-bus
RDM	6	xxxx xxxx xxxx xxxx x011 xxxx xxxx	Memory[M-reg] → T-reg, 0 → R-bus
ZTR	7	xxxx xxxx xxxx xxxx x111 xxxx xxxx	0 → R-bus
SC Group, source of S-bus			
ZTS	0	xxxx xxxx xxxx xxxx xxxx xxxx xx00	0 → S-bus
MTS	1	xxxx xxxx xxxx xxxx xxxx xxxx xx10	M-reg → S-bus
TTS	2	xxxx xxxx xxxx xxxx xxxx xxxx xx01	T-reg → S-bus
UTS	3	xxxx xxxx xxxx xxxx xxxx xxxx xx11	1 → S-bus
XC Group, destination of T-bus and miscellanous other operations			
TTQ	0	xxxx xxxx xxxx xxxx xxxx 000x xxxx	T-bus → Q-reg
QAB	1	xxxx xxxx xxxx xxxx xxxx 100x xxxx	Q-reg bit 10 → A/B-register selector, 0 → DC (decimal carry)
BCD	2	xxxx xxxx xxxx xxxx xxxx 001x xxxx	BCD operation mode
TBE	3	xxxx xxxx xxxx xxxx xxxx 101x xxxx	T-bus → E-reg → R-bus
CAB	4	xxxx xxxx xxxx xxxx xxxx 010x xxxx	Complement A/B-register selector
TTP	5	xxxx xxxx xxxx xxxx xxxx 110x xxxx	T-bus → P-reg
TTX	6	xxxx xxxx xxxx xxxx xxxx 011x xxxx	T-bus → A- or B-reg
NOP	7	xxxx xxxx xxxx xxxx xxxx 111x xxxx	No operation
AC Group, binary ALU operations			
XOR	0	xxxx xxxx xxxx x00x xxxx xxx0 xxxx	R-bus XOR S-bus → T-bus
AND	1	xxxx xxxx xxxx x10x xxxx xxx0 xxxx	R-bus AND S-bus → T-bus
IOR	2	xxxx xxxx xxxx x01x xxxx xxx0 xxxx	R-bus OR S-bus → T-bus
ZTT	3	xxxx xxxx xxxx x11x xxxx xxx0 xxxx	0 → T-bus
ZTT.CBC	4	xxxx xxxx xxxx x00x xxxx xxx1 xxxx	0 → T-bus, 0 → BC (binary carry)
IOR.CBC	5	xxxx xxxx xxxx x01x xxxx xxx1 xxxx	R-bus OR S-bus → T-bus, 0 → BC
IOR.SBC	6	xxxx xxxx xxxx x10x xxxx xxx1 xxxx	R-bus OR S-bus → T-bus, 1 → BC
ADD	7	xxxx xxxx xxxx x11x xxxx xxx1 xxxx	R-bus + S-bus + BC → T-bus, carry → BC
TC Group, destination of T-bus			
		xxxx xxxx xxxx xxxx xxxx xxxx 11xx	No operation
TTM		xxxx xxxx xxxx xxxx xxxx xxxx 01xx	T-bus → M-reg (may be combined with TTT)
TTT		xxxx xxxx xxxx xxxx xxxx xxxx 10xx	T-bus → T-reg (may be comined with TTM)
Register source of R-bus			
		xxxx xxxx xxxx xxx1 xxxx xxxx xxxx	No operation
XTR		xxxx xxxx xxxx xxx0 xxxx xxxx xxxx	A- or B-reg → R-bus

Mnemonic	#	Micro Code	Operation
BCD decimal ALU operations (derived from combinations of other operations)			
BCD•UTR		xxxx xxxx xxxx xxxx x000 001x xxxx	1 → DC
BCD•$\overline{\text{UTR}}$•ZTT		xxxx xxxx xxxx x11x x111 0010 xxxx	A_{3-0} + T_{3-0} + DC → A_{3-0} , carry → DC
BCD•$\overline{\text{UTR}}$•ADD		xxxx xxxx xxxx x11x x111 0011 xxxx	A_{3-0} + 9-T_{3-0} + DC → A_{3-0} , carry → DC
Miscellanous operations (derived from combinations of other operations)			
TQR (XTR•UTR)		xxxx xxxx xxxx xxx0 x000 xxxx xxxx	Q_{14} Q_{13} Q_{12} Q_{14}•Q_{11} → PM (set primary modifier from bits of Q-reg)
IOS (XTR•PTR)		xxxx xxxx xxxx xxx0 x100 xxxx xxxx	I/O operation if Q-reg bit Q_{10}=1, SRA if Q_{10}=0 (service request acknowledge)
ETR (TBE•TRE)		xxxx xxxx xxxx xxxx x010 101x xxxx	0 → E-reg → R-bus, T-bus → A- or B-reg
Conditional branch and shift			
BRC		qqqq xxxx xxxx xxxx 1xxx xxxx xxxx	Branch if qualifier coded in qqqq is false (0)
IQN		qqqq xxxx xxxx 1xxx xxxx xxxx xxxx	Inhibit shift if qualifier coded in qqqq is false
Next address modifiers and shift clock			
PM		pppp xxxx xxxx xxxx xxxx xxxx xxxx	Primary address modifier coded in pppp
SM		xxxx ssss xxxx xxxx xxxx xxxx xxxx	Secondary address modifier coded in ssss
CC		xxxx xxxx cccc xxxx xxxx xxxx xxxx	Shift clock cycles - 1 coded in cccc

The primary adress modifier also defines the qualifier code qqqq, when BRC or IQN operations are used.

The qualifier codes have the following meaning:

Qualifier code	Mnemonic	Value	Function
0000	Q0	Q-register bit 0	Shift/skip one bit
0001	Q1	Q-register bit 1	Shift/skip two bits
0010	Q2	Q-register bit 2	Shift/skip four bits
0011	Q3	Q-register bit 3	Shift/skip eight bits
0100	Q4	Q-register bit 4	Used in fast quare root
0101	Q5	Q-register bit 5	Used in FDIV instruction
0110	Q6	Q-register bit 6	Value of T-bus by TQ6 operation
0111	QBC	BC	Binary carry
1000	QP0	P-Register bit 0	Used in BCD counting
1001	Q15	Q-register bit 15	Indirect memory addressing
1010	QMR	Memory reference	Used in CPU instructions with memory access
1011	Q10	Q-register bit 10	Zero / current page and used in FXA instruction
1100	QNR	$\overline{\text{SRQ}}$	No service request
1101	Q8	Q-register bit 8	Used in FMP instruction
1110	QDC	DC	Decimal carry
1111	QRD	ROM disable	Set during I/O operation

D.2.2 Micro-Program Listing

As mentioned, the micro-program consists of 256 instructions, each 28 bits wide, and is stored in seven 256x4-bit ROMs. The contents of these ROMs are listed in various patents of the HP9800 series calculators. Brent Hilpert has typed the ROM dumps into a file and published it on http://www.cs.ubc.ca/~hilpert/e/HP9830/. Those values where used as input for this project. During testing it showed, that there was one typing error in Brents listing, which is corrected below and marked in yellow at address 1704. For the emulator the contents of all seven ROMs has been concatenated to 28 bit words and stored as binary values in a text file Microcode_ROM.dmp stored in the installation folder ./media/HP9800/.

The following table lists the complete micro-program with micro-operations, address and clock values, and the bit values of the seven ROMs. The column PASA gives the primary+secondary address of the instruction as two 2-digit octal values, the column NEXT is the next address to be executed, if no branching occurs. In case of a branch, the lowest bit of NEXT is inverted. The other columns denote the micro-operation groups and binary micro-codes.

In case of the derived operations TQR, IOS, and ETR these mnemonics replace the original RCodes UTR, PTR, and TRE. In case of ETR also the XCode TBE is replaced by TTX.

PASA	NEXT	BRC	IQN	C	XTR	RC	SC	XC	TTM	TTT	AC	ROM3	ROM4	ROM5	ROM6	ROM7	ROM8	ROM9
0000	0005			f	XTR	ZTR	TTS	TTX			ADD	0000	1010	1111	0110	0111	0111	1101
0001	0202	Q2		0		ZTR	ZTS	NOP			ZTT	0010	1001	0000	0111	1111	1110	1100
0002	1700		QRD	0		TQR	ZTS	NOP			ZTT	1111	0001	0000	1110	0000	1110	1100
0003	0602			f		TQ6	TTS	NOP	TTM		IOR	0110	1000	1111	0011	0001	1110	0101
0004	1710		QRD	0		TQR	ZTS	NOP			ZTT	1111	0110	0000	1110	0000	1110	1100
0005	0704	BC	BC	0		ZTR	UTS	TBE			IOR	0111	1000	0000	1011	1111	1010	1111
0006	1501		Q8	1	XTR	ZTR	TTS	BCD			ZTT	1101	1011	0001	1110	0111	0010	1101
0007	1501		Q8	1	XTR	ZTR	TTS	BCD			ADD	1101	0011	0001	1110	0111	0011	1101
0010	0406			f	XTR	TQ6	ZTS	NOP			IOR	0100	0111	1111	0010	0001	1110	1100
0011	0402	Q4		0		TQ6	UTS	NOP			IOR	0100	1101	0000	0011	1001	1110	1111
0012	1313	Q10		9		ZTR	TTS	NOP	TTM		IOR	1011	1000	1001	0011	1111	1110	0101
0013	1112	Q15		0		ZTR	ZTS	NOP			IOR.CBC	1001	1000	0000	0101	1111	1111	1100
0014	1217			f		TRE	UTS	NOP		TTT	AND	1010	1001	1111	0101	0010	1110	1011
0015	0501			f	XTR	ZTR	ZTS	TTX			AND	0101	0110	1111	0100	0111	0110	1100
0016	0017			0		ZTR	ZTS	TTX			ZTT	0000	1000	0000	0111	0111	0110	1100
0017	0001			3		QTR	ZTS	NOP	TTM		IOR.CBC	0000	0111	0011	0101	0101	1111	0100
0100	0601			0		PTR	ZTS	NOP			ZTT	0111	1000	0000	0111	0100	1110	1100
0101	0607			0		PTR	ZTS	NOP			ZTT	0111	0011	0000	0111	0100	1110	1100
0102	0106			0		ZTR	ZTS	NOP			ZTT	0000	0010	0000	0111	0111	1110	1100
0103	0603			3	XTR	ZTR	UTS	TTX			ADD	0111	0000	0011	0110	0111	0111	1111
0104	0103			0		ZTR	ZTS	NOP			ZTT.CBC	0000	1011	0000	0001	0111	1111	1100
0105	0106			f	XTR	ZTR	ZTS	TTX			ADD	0000	1001	1111	0110	0111	0111	1100
0106	0506			b		RDM	ZTS	CAB			ZTT	0100	0000	1011	0111	0011	0100	1100
0107	1505			0		ZTR	ZTS	CAB			ZTT	1100	0001	0000	0111	0111	0100	1100
0110	0310		Q2	3		ZTR	ZTS	TTX			ZTT	0010	0000	0011	1111	0111	0110	1100
0111	0311		Q2	3		UTR	UTS	TTX			IOR	0010	0000	0011	1011	0000	0110	1111
0112	0417			0		ZTR	MTS	NOP			IOR.SBC	0101	1010	0000	0011	0111	1111	1110
0113	1317			5		ZTR	ZTS	NOP	TTM		ZTT.CBC	1010	0010	0101	0001	0111	1111	0100
0114	0514			3		TRE	UTS	NOP		TTT	AND	0100	0000	0011	0101	0010	1110	1011
0115	1317			5		ZTR	ZTS	NOP	TTM		ZTT.CBC	1010	0001	0101	0001	0111	1111	0100
0116	0106			b		ZTR	MTS	NOP			ZTT	0000	0100	1011	0111	0111	1110	1110
0117	0101			b		RDM	ZTS	NOP			ZTT	0000	0111	1011	0111	0011	1110	1100
0200	0612			f	XTR	ZTR	TTS	NOP		TTT	XOR	0100	0101	1111	0000	0111	1110	1001
0201	0202			0		ZTR	ZTS	TTX			ZTT	0000	1001	0000	0111	0111	0110	1100
0202	1603			0		PTR	UTS	TTP	TTM		ADD	1100	1000	0000	0111	0100	1101	0111
0203	0204	Q0		0		ZTR	ZTS	NOP			ZTT	0000	1011	0000	0111	1111	1110	1100
0204	1216			b		UTR	UTS	NOP	TTM		IOR.CBC	1000	0101	1011	0101	0000	1111	0111
0205	1216			b		ZTR	ZTS	NOP	TTM		ZTT	1000	1101	1011	0111	0111	1110	0100
0206	0017			e	XTR	ZTR	ZTS	NOP			ZTT	0010	1100	1110	0110	0111	1110	1100
0207	1306	Q15	Q15	0		ZTR	ZTS	TTX			ZTT	1001	1000	0000	1111	1111	0110	1100
0210	0110		Q3	7		ZTR	ZTS	TTX			ZTT	0011	0000	0111	1111	0111	0110	1100
0211	0111		Q3	7		UTR	UTS	TTX			IOR	0011	0000	0111	1011	0000	0110	1111
0212	0201		Q0	0		ZTR	ZTS	TTX			ZTT	0000	1101	0000	1111	0111	0110	1100
0213	0202		Q0	0		UTR	UTS	TTX			IOR	0000	1100	0000	1011	0000	0110	1111
0214	0707	Q5	Q5	3		ZTR	UTS	TBE			IOR	0101	1101	0011	1011	1111	1010	1111
0215	1204	P0		b		WTM	ZTS	NOP			IOR.CBC	1000	1100	1011	0101	1110	1111	1100
0216	1715	Q8		0		QTR	ZTS	NOP			ZTT	1101	1001	0000	0111	1101	1110	1100
0217	0512		BC	1	XTR	IOS	ZTS	NOP			ZTT	0111	1010	0001	1110	0100	1110	1100
0300	1317			5		ZTR	ZTS	NOP	TTM		ZTT.CBC	1000	1111	0101	0001	0111	1111	0100
0301	1707			b		WTM	ZTS	NOP			IOR.CBC	1100	0011	1011	0101	0110	1111	1100
0302	0202			0		ZTR	ZTS	NOP			ZTT	0001	0000	0000	0111	0111	1110	1100
0303	0202			f	XTR	ZTR	TTS	TTX			AND	0001	1000	1111	0100	0111	0110	1101
0304	0504			0		QTR	ZTS	NOP			ZTT	0110	0000	0000	0111	0101	1110	1100
0305	0202			0		ZTR	ZTS	NOP			ZTT.CBC	0001	1011	0000	0001	0111	1111	1100
0306	1017			0		ZTR	ZTS	NOP	TTM		IOR.SBC	1011	1100	0000	0011	0111	1111	0100
0307	1017			0		UTR	UTS	NOP	TTM		IOR.SBC	1011	0100	0000	0011	0000	1111	0111
0310	0212		Q1	1		ZTR	ZTS	TTX			ZTT	0001	0001	0001	1111	0111	0110	1100
0311	0213		Q1	1		UTR	UTS	TTX			IOR	0001	0001	0001	1011	0000	0110	1111
0312	1707			3		ZTR	ZTS	TBE			ADD	1100	1110	0011	0111	0111	1011	1100
0313	0606			3		UTR	TTS	NOP			ADD	0101	1110	0011	0111	0000	1111	1101
0314	0214			0		ZTR	ZTS	NOP			ZTT	0001	0000	0000	0111	0111	1110	1100
0315	1614	Q8		b		WTM	ZTS	CAB			IOR.SBC	1101	1000	1011	0011	1110	0101	1100
0316	1013	Q10		f		ETR	UTS	TTX			AND	1011	1010	1111	0101	1010	1010	1111
0317	0717			f		UTR	TTS	NOP		TTT	XOR	0100	0000	1111	0001	0000	1110	1001
0400	0202			f		ZTR	TTS	TTX			IOR	0110	0001	1111	0011	0111	0110	1101
0401	0312	BC		b	XTR	ZTR	TTS	NOP			ZTT	0111	1101	1011	0110	1111	1110	1101
0402	1711			3		ZTR	ZTS	TBE			ADD	1011	1101	0011	0111	0111	1011	1100
0403	1202	DC		0		ZTR	ZTS	NOP			ZTT	1110	1000	0000	0111	1111	1110	1100
0404	0401			1	XTR	ZTR	ZTS	NOP			ADD	0000	1010	0001	0110	0111	1111	1100
0405	1405			0		PTR	ZTS	NOP	TTM		AND	1000	0000	0000	0101	0100	1110	0100
0406	0211	Q6	Q6	0		UTR	UTS	TTX			IOR	0110	1111	0000	1011	1000	0110	1111
0407	1700	Q10		5		ZTR	ZTS	NOP	TTM		ZTT.CBC	1011	1011	0101	0001	1111	1111	0100
0410	0510			f		TQ6	ZTS	NOP		TTT	ZTT	0001	0000	1111	0111	0001	1110	1000
0411	0511		Q1	1	XTR	ZTR	UTS	NOP		TTT	IOR	0001	0000	0001	1010	0111	1110	1011
0412	0017			f	XTR	ZTR	ZTS	TTX			ADD	0100	1010	1111	0110	0111	0111	1100

PASA	NEXT	BRC	IQN	C	XTR	RC	SC	XC	TTM	TTT	AC	ROM3	ROM4	ROM5	ROM6	ROM7	ROM8	ROM9
0413	0017			0		ZTR	ZTS	NOP			ZTT	0100	0010	0000	0111	0111	1110	1100
0414	1412			f		PTR	TTS	TTQ			IOR.CBC	1000	0011	1111	0101	0100	0001	1101
0415	0614			f	XTR	ZTR	ZTS	TTX	TTM		ADD	0010	1000	1111	0110	0111	0111	0100
0416	0404			1	XTR	ZTR	ZTS	NOP			IOR.SBC	0000	0101	0001	0010	0111	1111	1100
0417	1511			e		ZTR	MTS	NOP	TTM		ADD	1001	0011	1110	0111	0111	1111	0110
0500	0215			0		PTR	ZTS	NOP			ZTT	0111	1110	0000	0111	0100	1110	1100
0501	0006	Q5		0		PTR	ZTS	NOP			ZTT	0101	1011	0000	0111	1100	1110	1100
0502	1507	P0		b		WTM	ZTS	NOP			IOR.SBC	1000	1010	1011	0011	1110	1111	1100
0503	0302	Q6	Q6	0	XTR	ZTR	ZTS	NOP			ZTT	0110	1000	0000	1110	1111	1110	1100
0504	1604			f		ZTR	MTS	NOP	TTM		ADD	1011	0000	1111	0111	0111	1111	0110
0505	1116			f	XTR	IOS	UTS	NOP			ADD	1100	1101	1111	0110	0100	1111	1111
0506	0516			f		UTR	TTS	TTX			AND	0000	0100	1111	0101	0000	0110	1101
0507	0505			f		UTR	TTS	CAB			ADD	0000	0001	1111	0111	0000	0101	1101
0510	0611		Q3	7	XTR	ZTR	UTS	NOP		TTT	IOR	0011	1000	0111	1010	0111	1110	1011
0511	0503		Q0	0	XTR	ZTR	UTS	NOP		TTT	IOR	0000	0101	0000	1010	0111	1110	1011
0512	1207			1		ZTR	ZTS	NOP			IOR	1111	1110	0001	0011	0111	1110	1100
0513	0202			0		ZTR	ZTS	NOP			ZTT	0111	1100	0000	0111	0111	1110	1100
0514	0502			4		PTR	ZTS	NOP			ZTT	0000	0111	0100	0111	0100	1110	1100
0515	0605			3	XTR	ZTR	UTS	TTX			ADD	0011	0100	0011	0110	0111	0111	1111
0516	0517			2		ZTR	MTS	NOP			IOR.SBC	0000	1000	0010	0011	0111	1111	1110
0517	0117			c		ZTR	MTS	NOP	TTM		ADD	0100	0000	1100	0111	0111	1111	0110
0600	1015			f	XTR	ZTR	ZTS	NOP		TTT	IOR	1110	1110	1111	0010	0111	1110	1000
0601	1606	P0		b		WTM	ZTS	CAB			ZTT	1000	1011	1011	0111	1110	0100	1100
0602	0003	Q6		b		RDM	ZTS	NOP			ZTT	0110	1000	1011	0111	1011	1110	1100
0603	0106	BC		b	XTR	ZTR	ZTS	NOP			ZTT	0111	1010	1011	0110	1111	1110	1100
0604	0113	BC		3		ZTR	ZTS	TTX			AND	0111	1111	0011	0101	1111	0110	1100
0605	0114	BC		b	XTR	ZTR	ZTS	NOP			ZTT	0111	1100	1011	0110	1111	1110	1100
0606	0113	BC		0		PTR	ZTS	NOP			ZTT	0111	1110	0000	0111	1100	1110	1100
0607	1103		QRD	1	XTR	ZTR	TTS	BCD			ZTT	1111	0010	0001	1110	0111	0010	1101
0610	0510			0		TQ6	UTS	NOP			IOR	0011	0000	0000	0011	0001	1110	1111
0611	0411		Q2	3	XTR	ZTR	UTS	NOP		TTT	IOR	0010	0000	0011	1010	0111	1110	1011
0612	0202			f		UTR	TTS	NOP			ADD	0100	0100	1111	0111	0000	1111	1101
0613	0617			f	XTR	ZTR	ZTS	NOP	TTM		IOR	0000	0010	1111	0010	0111	1110	0100
0614	0615			b		RDM	ZTS	CAB			IOR.SBC	0000	1000	1011	0011	0011	0101	1100
0615	0315			f	XTR	ZTR	ZTS	TTX	TTM		ADD	0101	0000	1111	0110	0111	0111	0100
0616	1213	QNR		f		UTR	TTS	TTQ			AND	1100	1010	1111	0101	1000	0000	1101
0617	1211			0		ZTR	ZTS	CAB			ZTT	1100	0011	0000	0111	0111	0100	1100
0700	0616			f	XTR	ZTR	ZTS	NOP		TTT	IOR.CBC	0001	0111	1111	0100	0111	1111	1000
0701	0613			b		WTM	ZTS	NOP			ZTT	0001	0101	1011	0111	0110	1110	1100
0702	0703			e	XTR	ZTR	ZTS	TTX			ADD	0000	1000	1110	0110	0111	0111	1100
0703	0202			0		ZTR	ZTS	NOP			IOR.CBC	0101	1000	0000	0101	0111	1111	1100
0704	0202			2		ZTR	ZTS	TBE			ZTT.CBC	0101	0011	0010	0001	0111	1011	1100
0705	0202			0		ZTR	ZTS	NOP			ZTT	0101	1011	0000	0111	0111	1110	1100
0706	1611		Q15	3		ZTR	ZTS	TBE			AND	1001	1111	0011	1101	0111	1010	1100
0707	0017			0		ZTR	ZTS	NOP			ZTT	0111	0100	0000	0111	0111	1110	1100
0710	0714			b		ZTR	TTS	TTX			AND	0000	0010	1011	0101	0111	0110	1101
0711	0206	Q5	Q5	0		UTR	UTS	TTX			IOR	0101	1111	0000	1011	1000	0110	1111
0712	1013			3	XTR	ZTR	UTS	TBE			AND	1111	1000	0011	0100	0111	1010	1111
0713	0114			b		RDM	ZTS	CAB			ZTT	0110	1011	1011	0111	0011	0100	1100
0714	0604			3		UTR	TTS	NOP			ADD	0001	0100	0011	0111	0000	1111	1101
0715	0316			d		ZTR	MTS	NOP			ZTT	0100	1001	1101	0111	0111	1110	1110
0716	0715			0		PTR	ZTS	NOP	TTM		AND	0000	1001	0000	0101	0100	1110	0100
0717	0202			f		ZTR	TTS	NOP		TTT	ADD	0101	1110	1111	0111	0111	1111	1001
1000	0202			f	XTR	ZTR	TTS	TTX			IOR	1010	0001	1111	0010	0111	0110	1101
1001	1406	Q4		5		UTR	UTS	NOP	TTM		ADD	0100	1011	0101	0111	1000	1111	0111
1002	0416			f	XTR	ZTR	ZTS	TTX			ADD	1100	0110	1111	0110	0111	0111	1100
1003	1504	Q5		0		ZTR	ZTS	NOP			ZTT	0101	1011	0000	0111	1111	1110	1100
1004	1703	BC		1		ZTR	ZTS	NOP	TTM		AND	0111	1011	0001	0101	1111	1110	0100
1005	0306	Q10		3		ZTR	ZTS	TBE			ZTT	1011	1001	0011	0111	1111	1010	1100
1006	1001			0		UTR	UTS	NOP	TTM		IOR.CBC	0000	1011	0000	0101	0000	1111	0111
1007	1001			0		ZTR	ZTS	NOP	TTM		IOR.CBC	0000	0011	0000	0101	0111	1111	0100
1010	1513	Q5		f	XTR	ZTR	UTS	NOP			ADD	0101	1001	1111	0110	1111	1111	1111
1011	1201			f		ZTR	MTS	TTX			IOR	0010	0100	1111	0011	0111	0110	1110
1012	1317			5		ZTR	ZTS	NOP	TTM		ZTT.CBC	0011	1010	0101	0001	0111	1111	0100
1013	0515			b		RDM	ZTS	CAB			IOR.CBC	1101	0011	1011	0101	0011	0101	1100
1014	1214			b		WTM	ZTS	NOP			IOR.CBC	0010	0000	1011	0101	0110	1111	1100
1015	0202			b		WTM	ZTS	NOP			ZTT	1010	1111	1011	0111	0110	1110	1100
1016	0116			0		ZTR	MTS	NOP	TTM		ADD	1001	0000	0000	0111	0111	1111	0110
1017	1414	Q4		1		UTR	ZTS	SDC			XOR	0100	1001	0001	0001	1000	0010	1100
1100	1105			0		UTR	TTS	NOP		TTT	ADD	0000	1010	0000	0111	0000	1111	1001
1101	1304	Q2		3		ZTR	UTS	TBE			ADD	0010	1010	0011	0111	1111	1011	1111
1102	0003	Q15		b		RDM	ZTS	NOP			ZTT.CBC	1001	1000	1011	0001	1011	1111	1100
1103	0100	P0		3	XTR	ZTR	UTS	NOP		TTT	AND	1000	1001	0011	0100	1111	1110	1011
1104	0003	Q15		0		ZTR	ZTS	QAB			ZTT	1001	1011	0000	0111	1111	1000	1100
1105	1015			e		ZTR	TTS	NOP		TTT	ADD	0001	0100	1110	0111	0111	1111	1001

PASA	NEXT	BRC	IQN	C	XTR	RC	SC	XC	TTM	TTT	AC	ROM3	ROM4	ROM5	ROM6	ROM7	ROM8	ROM9
1106	0017			0		UTR	UTS	TTX			IOR	1001	1100	0000	0011	0000	0110	1111
1107	0016	Q15	Q15	e	XTR	ZTR	ZTS	NOP			ZTT	1001	1100	1110	1110	1111	1110	1100
1110	0711		DC	0	XTR	TQ6	ZTS	NOP			IOR	1110	1000	0000	1010	0001	1110	1100
1111	1612		BC	0	XTR	IOS	ZTS	NOP			ZTT	0111	1001	0000	1110	0100	1110	1100
1112	0217	Q10		0		ZTR	ZTS	NOP			ZTT	1011	1010	0000	0111	1111	1110	1100
1113	0004			4		QTR	ZTS	NOP			ZTT	1001	1111	0100	0111	0101	1110	1100
1114	1014			f		ZTR	TTS	NOP		TTT	ADD	0001	0000	1111	0111	0111	1111	1001
1115	1114			b		RDM	ZTS	NOP			IOR.SBC	0000	1000	1011	0011	0011	1111	1100
1116	1613	BC		3		ZTR	ZTS	TBE			ZTT	0111	1010	0011	0111	1111	1010	1100
1117	0407	Q8		4		UTR	UTS	NOP	TTM		IOR	1101	0100	0100	0011	1000	1110	0111
1200	0202			f	XTR	ZTR	TTS	TTX			AND	1000	0001	1111	0100	0111	0110	1101
1201	0414			f	XTR	ZTR	MTS	TTP			XOR	1110	1110	1111	0000	0111	1100	1110
1202	1317			5		ZTR	ZTS	NOP	TTM		ZTT.CBC	0001	1110	0101	0001	0111	1111	0100
1203	0405			2		UTR	UTS	NOP	TTM		IOR.SBC	1110	0011	0010	0011	0000	1111	0111
1204	1511			f		UTR	MTS	NOP	TTM		ADD	0111	1110	1111	0111	0000	1111	0110
1205	1002	Q2		0		TQ6	UTS	NOP			IOR.SBC	0010	1011	0000	0011	1001	1111	1111
1206	1011			f		PTR	ZTS	CAB	TTM	TTT	AND	0010	1111	1111	0101	0100	0100	0000
1207	0512	QRD		1		ZTR	ZTS	NOP			IOR	1111	1110	0001	0011	1111	1110	1100
1210	1411			f	XTR	TQ6	UTS	NOP			XOR	0110	1000	1111	0000	0001	1110	1111
1211	0202			f	XTR	ZTR	ZTS	NOP	TTM		IOR	1000	1101	1111	0010	0111	1110	0100
1212	1602			f		PTR	UTS	TTP			ADD	0100	0100	1111	0111	0100	1101	1111
1213	0012	QMR		0		ZTR	ZTS	QAB			ZTT	1010	1000	0000	0111	1111	1000	1100
1214	1215			f		UTR	TTS	NOP	TTM		ADD	0000	1000	1111	0111	0000	1111	0101
1215	1315			f		PTR	ZTS	NOP		TTT	IOR.CBC	0001	0000	1111	0101	0100	1111	1000
1216	1612			f		PTR	MTS	TTP	TTM		ADD	0100	0010	1111	0111	0100	1101	0110
1217	0215			4		PTR	ZTS	NOP			ZTT	1000	0001	0100	0111	0100	1110	1100
1300	1311			f		UTR	TTS	NOP		TTT	ADD	0000	1100	1111	0111	0000	1111	1001
1301	1502	Q6		5		ZTR	ZTS	CAB	TTM		ZTT	0110	1001	0101	0111	1111	0100	0100
1302	1101			f		UTR	TTS	TTP			AND	0010	1001	1111	0101	0000	1100	1101
1303	0202			f		UTR	TTS	TTP			AND	1001	1000	1111	0101	0000	1100	1101
1304	1413			0		TQ6	UTS	NOP	TTM		IOR.CBC	0111	1111	0000	0101	0001	1111	0111
1305	1004	Q3		2		TQ6	UTS	NOP	TTM		IOR	0011	1000	0010	0011	1001	1110	0111
1306	0017			e	XTR	ZTR	ZTS	NOP			ZTT	1011	1100	1110	0110	0111	1110	1100
1307	0017			0		ZTR	ZTS	NOP			ZTT	1011	0100	0000	0111	0111	1110	1100
1310	1411			f	XTR	TQ6	ZTS	NOP			IOR	0111	1000	1111	0010	0001	1110	1100
1311	0317			b		WTM	ZTS	NOP			IOR.SBC	1000	0011	1011	0011	0110	1111	1100
1312	1102			5		ZTR	ZTS	NOP	TTM		ZTT	0010	0100	0101	0111	0111	1110	0100
1313	1102			5		ZTR	MTS	NOP			ZTT	0010	1100	0101	0111	0111	1110	1110
1314	1612			f		QTR	UTS	TTP	TTM		AND	0101	0011	1111	0101	0101	1100	0111
1315	1314			b		WTM	ZTS	NOP			ZTT	0000	1000	1011	0111	0110	1110	1100
1316	1104			3		QTR	ZTS	NOP			ZTT	0010	0101	0011	0111	0101	1110	1100
1317	1301			3		UTR	UTS	QAB	TTM		IOR	0000	0111	0011	0011	0000	1000	0111
1400	1514			f		UTR	MTS	TTQ			AND	0001	0110	1111	0101	0000	0000	1110
1401	1402	Q0		b		WTM	ZTS	NOP			IOR.SBC	0000	1001	1011	0011	1110	1111	1100
1402	1502			f		UTR	TTS	NOP	TTM		AND	0001	0000	1111	0101	0000	1110	0101
1403	1502			f		ZTR	ZTS	NOP	TTM		ADD	0001	1000	1111	0111	0111	1111	0100
1404	0103	Q8		5		ZTR	ZTS	TBE	TTM		AND	1101	1011	0101	0101	1111	1010	0100
1405	0104	Q8		b		ZTR	MTS	QAB			ZTT	1101	1000	1011	0111	1111	1000	1110
1406	1713	Q3		5		ZTR	ZTS	TBE	TTM		ZTT.CBC	0011	1110	0101	0001	1111	1011	0100
1407	0712	Q10		5		ZTR	ZTS	TBE	TTM		ZTT.CBC	1011	1110	0101	0001	1111	1011	0100
1410	0314		QRD	0		TQ6	UTS	TBE			XOR	1111	0010	0000	1001	0001	1010	1111
1411	1106	Q5	Q5	e	XTR	ZTR	ZTS	NOP			ZTT	0101	1111	1110	1110	1111	1110	1100
1412	0507			f	XTR	QTR	MTS	NOP		TTT	ADD	1001	1110	1111	0110	0101	1111	1010
1413	1006	Q4		1		UTR	UTS	NOP	TTM		ADD	0100	1110	0001	0111	1000	1111	0111
1414	0112			f	XTR	ZTR	ZTS	NOP	TTM		ADD	1101	0011	1111	0110	0111	1111	0100
1415	1117			0		ZTR	ZTS	NOP	TTM		ZTT	0101	0001	0000	0111	0111	1110	0100
1416	1417			b		RDM	ZTS	NOP			IOR.CBC	0000	1000	1011	0101	0011	1111	1100
1417	1401			f		UTR	TTS	NOP		TTT	ADD	0000	0111	1111	0111	0000	1111	1001
1500	0616			f		ZTR	MTS	TTP			IOR.CBC	1011	0111	1111	0101	0111	1101	1110
1501	0500	P0		3	XTR	ZTR	UTS	NOP		TTT	AND	1000	1000	0011	0100	1111	1110	1011
1502	1303	Q6		b		RDM	ZTS	NOP			IOR.CBC	0110	1000	1011	0101	1011	1111	1100
1503	1401			f		PTR	UTS	NOP		TTT	AND	0001	0001	1111	0101	0100	1110	1011
1504	0301	DC	DC	3		ZTR	ZTS	TBE		TTT	ADD	1110	1010	0011	1111	1111	1011	1000
1505	1707			f		ETR	UTS	TTX			AND	0010	0001	1111	0101	0010	1010	1111
1506	0716			0		ZTR	MTS	CAB			ZTT	1010	0100	0000	0111	0111	0100	1110
1507	0713			f		ZTR	MTS	CAB	TTM		ADD	1010	0110	1111	0111	0111	0101	0110
1510	0214		QRD	0		TQ6	ZTS	TBE			IOR	1111	0010	0000	1011	0001	1010	1100
1511	0014	Q8		b		RDM	ZTS	NOP			ZTT	1101	1010	1011	0111	1011	1110	1100
1512	1517			0		TQ6	ZTS	NOP			ADD	0000	1010	0000	0111	0001	1111	1100
1513	1517			0		TQ6	UTS	NOP			ADD	0000	0010	0000	0111	0001	1111	1111
1514	1515			9		UTR	UTS	NOP	TTM		IOR	0000	1000	1001	0011	0000	1110	0111
1515	1115			5		ZTR	ZTS	NOP	TTM		ZTT	0100	0000	0101	0111	0111	1110	0100
1516	1416			5		ZTR	ZTS	NOP	TTM		AND	0001	0000	0101	0101	0111	1110	0100
1517	0412	Q15		0		ZTR	ZTS	NOP			IOR.SBC	1001	1010	0000	0011	1111	1111	1100
1600	0701	Q15	Q15	0		UTR	UTS	NOP	TTM		IOR	1001	1000	0000	1011	1000	1110	0111

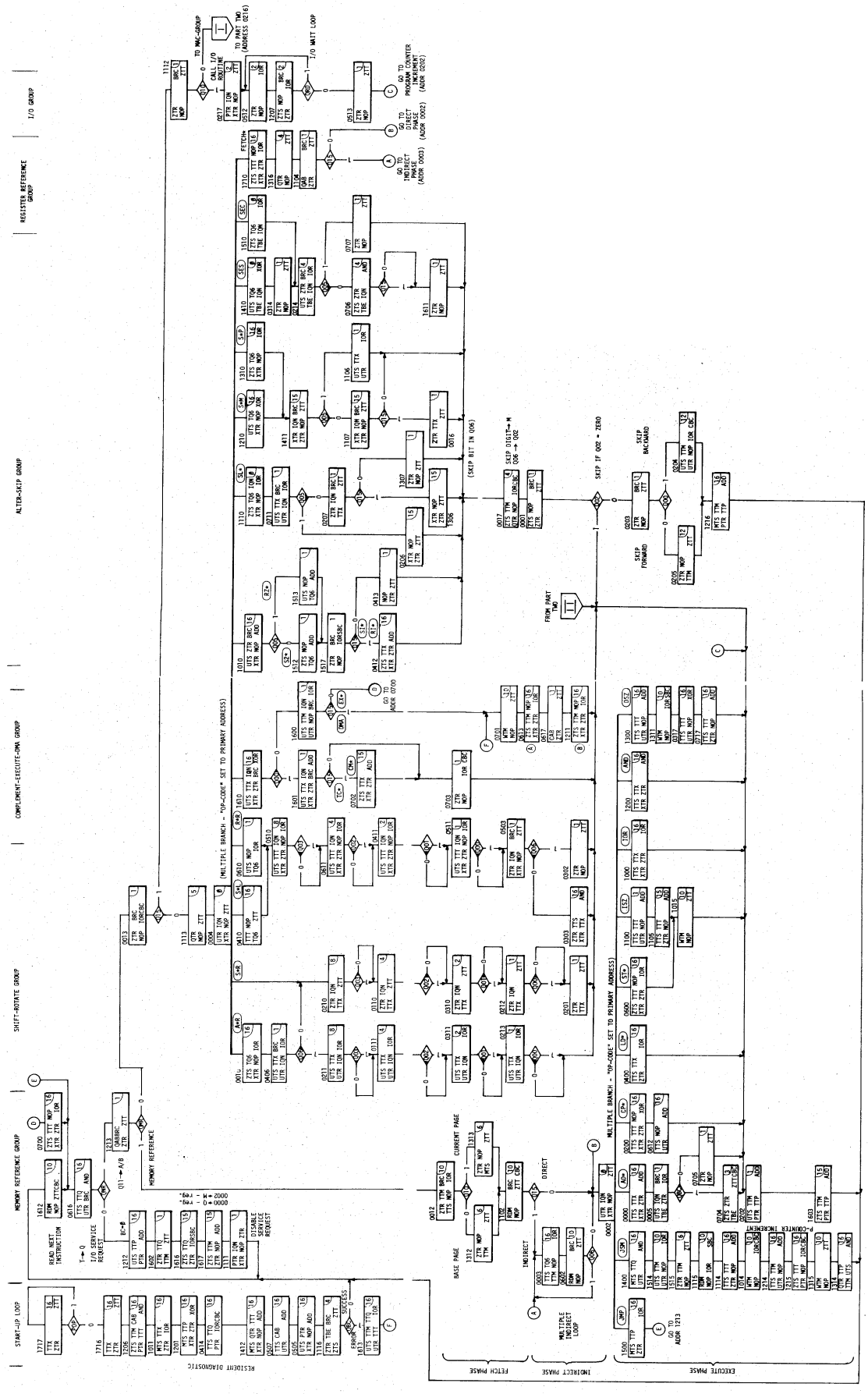
PASA	NEXT	BRC	IQN	C	XTR	RC	SC	XC	TTM	TTT	AC	ROM3	ROM4	ROM5	ROM6	ROM7	ROM8	ROM9
1601	0702	Q15	Q15	0	XTR	ZTR	UTS	TTX			ADD	1001	1001	0000	1110	1111	0111	1111
1602	1616			0		ZTR	ZTS	TTQ	TTM		ZTT	0000	0110	0000	0111	0111	0000	0100
1603	1612			e		PTR	ZTS	TTP	TTM		ADD	0000	1100	1110	0111	0100	1101	0100
1604	0305	Q8		b		WTM	ZTS	NOP			IOR.SBC	1101	1000	1011	0011	1110	1111	1100
1605	0615			b		RDM	ZTS	CAB			IOR.CBC	1000	0100	1011	0101	0011	0101	1100
1606	1016			2		UTR	MTS	NOP	TTM		ADD	0110	0100	0010	0111	0000	1111	0110
1607	0011		DC	0		ZTR	ZTS	NOP			IOR.SBC	1110	0111	0000	1011	0111	1111	1100
1610	1601	Q0	Q0	f	XTR	ZTR	UTS	TTX			XOR	0000	1100	1111	1000	1111	0110	1111
1611	0017			0		ZTR	ZTS	NOP			ZTT	1110	0011	0000	0111	0111	1110	1100
1612	0616			b		RDM	ZTS	NOP			ZTT.CBC	1000	0010	1011	0001	0011	1111	1100
1613	0701			f		UTR	UTS	TTQ	TTM	TTT	IOR	1001	0101	1111	0011	0000	0000	0011
1614	0202			0		ZTR	ZTS	NOP			ZTT.CBC	1100	0111	0000	0001	0111	1111	1100
1615	0415			0		QTR	ZTS	NOP			ZTT	1010	0000	0000	0111	0101	1110	1100
1616	1617			e		ZTR	ZTS	TTQ			IOR.SBC	0000	1000	1110	0011	0111	0001	1100
1617	1111			e		ZTR	ZTS	NOP	TTM		ADD	0111	0011	1110	0111	0111	1111	0100
1700	1511			3	XTR	ZTR	UTS	TBE			AND	0010	1100	0011	0100	0111	1010	1111
1701	1511			0		ZTR	ZTS	NOP			ZTT	0010	0100	0000	0111	0111	1110	1100
1702	1404			4		UTR	UTS	CAB	TTM		IOR.CBC	0011	0011	0100	0101	0000	0101	0111
1703	1702			1		UTR	ZTS	SDC			XOR	0000	1000	0001	0001	0000	0010	1100
1704	1605	Q1		f	XTR	ZTR	UTS	NOP	TTM		AND	0001	1000	1111	0100	1111	1110	0111
1705	0202			f		ETR	UTS	TTX			AND	1101	1011	1111	0101	0010	1010	1111
1706	1516			9		TQ6	UTS	NOP	TTM		IOR	0010	0100	1001	0011	0001	1110	0111
1707	1317			5		ZTR	ZTS	NOP	TTM		ZTT.CBC	0100	0100	0101	0001	0111	1111	0100
1710	1316			f	XTR	ZTR	ZTS	NOP		TTT	IOR	0100	0011	1111	0010	0111	1110	1000
1711	1202	Q5		0		ZTR	ZTS	NOP			ZTT	0101	1101	0000	0111	1111	1110	1100
1712	0710			b		RDM	ZTS	CAB			ZTT	1000	0001	1011	0111	0011	0100	1100
1713	1013			3	XTR	ZTR	UTS	TBE			AND	0111	0000	0011	0100	0111	1010	1111
1714	1705	Q0		f		ZTR	ZTS	NOP		TTT	AND	0000	1100	1111	0101	1111	1110	1000
1715	1706	Q0		0		TQ6	ZTS	NOP			AND	0000	1101	0000	0101	1001	1110	1100
1716	1206			f		ZTR	ZTS	TTX			ZTT	0101	0100	1111	0111	0111	0110	1100
1717	1716			f		ZTR	ZTS	TTX			ZTT	0000	1000	1111	0111	0111	0110	1100

D.2.3 Micro-Program Flow-Chart

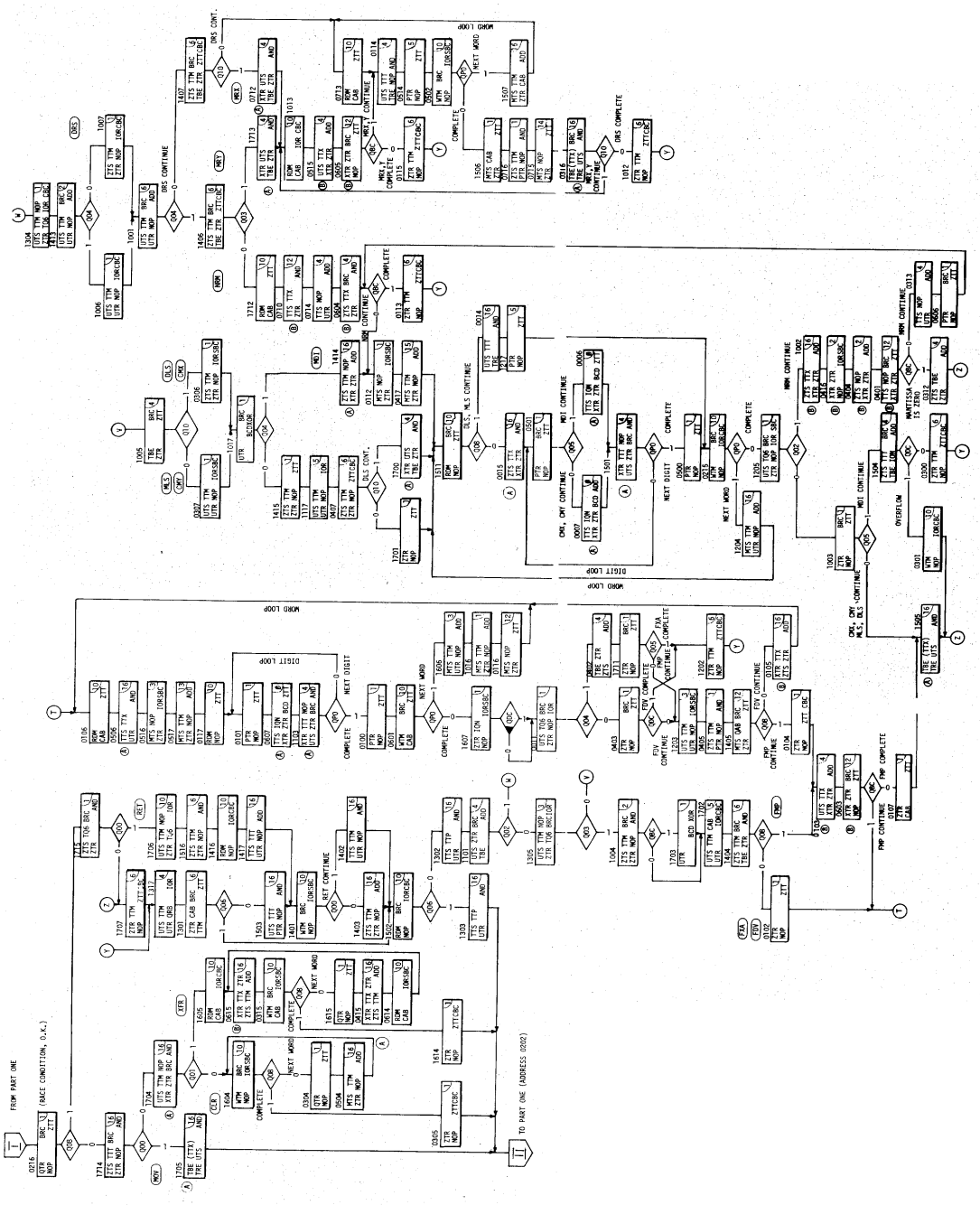
In the HP9800 series patents the micro-program is depicted as flow-charts, which make it easier to understand the program logic. These flow-charts are reproduced on the following pages.

Each micro-instruction is drawn as a rectangular box with the program address above the upper left corner. Inside the box the micro-operations are denoted and clock cycles are in the upper right corner. Branch and clock inhibit conditions are shown as diamond below the instruction box, with the qualifier to be tested inside. For IQN both outgoing branches meet at the next instruction.

It is remarkable, that certain instructions are denoted with 0 clock cycles, although from the decoded listing one would expect a clock count >0 (remember that in the listing column C contains clock count -1). For example this is the case for instructions 1410 and 1510, which are on the first flow-chart in the long horizontal multiple branch. The reason for this is, that the qualifier tested, QRD, is in these cases always false, because we are not in an I/O operation. Another example is instruction 1110, where the qualifier is the decimal carry DC, which is always 0 at this point. So shifting will always be inhibited and the effective clock count is 0. This technique is used, because the involved micro-op TQ6, which transfers the T-bus to Q-register bit 6, doesn't need any shifting.



Micro program flow chart 1 (extracted from HP9810A patent 3,859,635)



Micro program flow chart 2 (extracted from HP9810A patent 3,859,635)

D.2.4 Listing of the ALU ROM

As described above, the ALU functions XOR, AND, ADD, etc. are implemented as pre-programmed results in a 256x4-bit ROM. The inputs for this ROM are mainly the R- and S-bus, the binary carry, and the micro-code AC, which controls the kind of function to be executed. The output Y goes to the T-bus and carry flag BC. This ROM is also used in BCD operations. For this, additional inputs BCD and T-register bits 1, 2, and 3 are used. The latter two also replace the AC bits 0 and 1.

Decimal ALU functions are implemented in a second 256x4-bit ROM. It operates on the A- and T-register bits 1, 2, and 3. The inputs for this ROM are the said A-register bits and three additional outputs from the first ALU ROM as partial result. The outputs are the result bits Z 1, 2, and 3, which go back to the A-register and the decimal carry DC in Z0. The result bit 0 is the same as in binary operations, Y3 and comes from the first ALU ROM. In decimal operations only the DC carry is used whereas binary operations only use BC as carry input and output.

The following table lists the output values of the ALU ROM in column Y. The input values are in columns BCD, AC, T1, BC, R (-bus), and S (-bus). The ADDR column gives the combined ROM adress from the input values.

The result values where typed from the patent prints. During the tests several errors in the printed values (typing errors in the orginal patents) where detected and corrected in the listings of the ALU and BCD ROM. Some of these errors occur in all patents, some only in the HP9830 patent. The corrected values are marked in yellow. The errors where found due to incorrect calculation results and affected only the BCD functions. See the last paragraph of ch. 6 more informations about an interesting error in the BCD ROM.

For the emulator the contents of the two ROMs are stored as binary values in text files ALUcode_ROM.dmp and BCDcode_ROM.dmp in the installation folder ./media/HP9800/.

BCD	AC	T1	BC	R	S	Y	ADDR
0	000	0	0	0	0	0111	7
0	000	0	0	0	1	1111	6
0	000	0	0	1	0	1111	5
0	000	0	0	1	1	0111	4
0	000	0	1	0	0	0111	3
0	000	0	1	0	1	1111	2
0	000	0	1	1	0	1111	1
0	000	0	1	1	1	0111	0
0	000	1	0	0	0	0111	39
0	000	1	0	0	1	1111	38
0	000	1	0	1	0	1111	37
0	000	1	0	1	1	0111	36
0	000	1	1	0	0	0111	35
0	000	1	1	0	1	1111	34
0	000	1	1	1	0	1111	33
0	000	1	1	1	1	0111	32
0	001	0	0	0	0	0111	23
0	001	0	0	0	1	0111	22
0	001	0	0	1	0	0111	21
0	001	0	0	1	1	1111	20
0	001	0	1	0	0	0111	19
0	001	0	1	0	1	0111	18
0	001	0	1	1	0	0111	17
0	001	0	1	1	1	1111	16
0	001	1	0	0	0	0111	55
0	001	1	0	0	1	0111	54
0	001	1	0	1	0	0111	53
0	001	1	0	1	1	1111	52
0	001	1	1	0	0	0111	51
0	001	1	1	0	1	0111	50
0	001	1	1	1	0	0111	49
0	001	1	1	1	1	1111	48
0	010	0	0	0	0	0111	15
0	010	0	0	0	1	1111	14
0	010	0	0	1	0	1111	13
0	010	0	0	1	1	1111	12
0	010	0	1	0	0	0111	11
0	010	0	1	0	1	1111	10
0	010	0	1	1	0	1111	9
0	010	0	1	1	1	1111	8
0	010	1	0	0	0	0111	47
0	010	1	0	0	1	1111	46
0	010	1	0	1	0	1111	45
0	010	1	0	1	1	1111	44
0	010	1	1	0	0	0111	43
0	010	1	1	0	1	1111	42
0	010	1	1	1	0	1111	41
0	010	1	1	1	1	1111	40
0	011	0	0	0	0	0111	31
0	011	0	0	0	1	0111	30
0	011	0	0	1	0	0111	29
0	011	0	0	1	1	0111	28
0	011	0	1	0	0	0111	27
0	011	0	1	0	1	0111	26
0	011	0	1	1	0	0111	25
0	011	0	1	1	1	0111	24
0	011	1	0	0	0	0111	63
0	011	1	0	0	1	0111	62
0	011	1	0	1	0	0111	61
0	011	1	0	1	1	0111	60
0	011	1	1	0	0	0111	59
0	011	1	1	0	1	0111	58
0	011	1	1	1	0	0111	57
0	011	1	1	1	1	0111	56
0	100	0	0	0	0	0011	71
0	100	0	0	0	1	0011	70
0	100	0	0	1	0	0011	69
0	100	0	0	1	1	0011	68
0	100	0	1	0	0	0011	67
0	100	0	1	0	1	0011	66
0	100	0	1	1	0	0011	65
0	100	0	1	1	1	0011	64
0	100	1	0	0	0	0011	103

BCD	AC	T1	BC	R	S	Y	ADDR
1	000	0	0	0	0	0000	159
1	000	0	0	0	1	1000	158
1	000	0	0	1	0	1000	157
1	000	0	0	1	1	0100	156
1	000	0	1	0	0	1000	155
1	000	0	1	0	1	0100	154
1	000	0	1	1	0	0100	153
1	000	0	1	1	1	1100	152
1	000	1	0	0	0	0000	191
1	000	1	0	0	1	1000	190
1	000	1	0	1	0	1000	189
1	000	1	0	1	1	0100	188
1	000	1	1	0	0	1000	187
1	000	1	1	0	1	0100	186
1	000	1	1	1	0	0100	185
1	000	1	1	1	1	1100	184
1	001	0	0	0	0	0010	151
1	001	0	0	0	1	1010	150
1	001	0	0	1	0	1010	149
1	001	0	0	1	1	0110	148
1	001	0	1	0	0	1010	147
1	001	0	1	0	1	0110	146
1	001	0	1	1	0	0110	145
1	001	0	1	1	1	1110	144
1	001	1	0	0	0	1111	183
1	001	1	0	0	1	1111	182
1	001	1	0	1	0	1111	181
1	001	1	0	1	1	1111	180
1	001	1	1	0	0	1111	179
1	001	1	1	0	1	1111	178
1	001	1	1	1	0	1111	177
1	001	1	1	1	1	1111	176
1	010	0	0	0	0	0001	143
1	010	0	0	0	1	1001	142
1	010	0	0	1	0	1001	141
1	010	0	0	1	1	0101	140
1	010	0	1	0	0	1001	139
1	010	0	1	0	1	0101	138
1	010	0	1	1	0	0101	137
1	010	0	1	1	1	1101	136
1	010	1	0	0	0	0001	175
1	010	1	0	0	1	1001	174
1	010	1	0	1	0	1001	173
1	010	1	0	1	1	0101	172
1	010	1	1	0	0	1001	171
1	010	1	1	0	1	0101	170
1	010	1	1	1	0	0101	169
1	010	1	1	1	1	1101	168
1	011	0	0	0	0	1111	135
1	011	0	0	0	1	1111	134
1	011	0	0	1	0	1111	133
1	011	0	0	1	1	1111	132
1	011	0	1	0	0	1111	131
1	011	0	1	0	1	1111	130
1	011	0	1	1	0	1111	129
1	011	0	1	1	1	1111	128
1	011	1	0	0	0	1111	167
1	011	1	0	0	1	1111	166
1	011	1	0	1	0	1111	165
1	011	1	0	1	1	1111	164
1	011	1	1	0	0	1111	163
1	011	1	1	0	1	1111	162
1	011	1	1	1	0	1111	161
1	011	1	1	1	1	1111	160
1	100	0	0	0	0	1010	223
1	100	0	0	0	1	0010	222
1	100	0	0	1	0	0110	221
1	100	0	0	1	1	1010	220
1	100	0	1	0	0	0110	219
1	100	0	1	0	1	1010	218
1	100	0	1	1	0	1110	217
1	100	0	1	1	1	0110	216
1	100	1	0	0	0	1001	255

BCD	AC	T1	BC	R	S	Y	ADDR
0	100	1	0	0	1	0011	102
0	100	1	0	1	0	0011	101
0	100	1	0	1	1	0011	100
0	100	1	1	0	0	0011	99
0	100	1	1	0	1	0011	98
0	100	1	1	1	0	0011	97
0	100	1	1	1	1	0011	96
0	101	0	0	0	0	0011	87
0	101	0	0	0	1	1011	86
0	101	0	0	1	0	1011	85
0	101	0	0	1	1	1011	84
0	101	0	1	0	0	0011	83
0	101	0	1	0	1	1011	82
0	101	0	1	1	0	1011	81
0	101	0	1	1	1	1011	80
0	101	1	0	0	0	0011	119
0	101	1	0	0	1	1011	118
0	101	1	0	1	0	1011	117
0	101	1	0	1	1	1011	116
0	101	1	1	0	0	0011	115
0	101	1	1	0	1	1011	114
0	101	1	1	1	0	1011	113
0	101	1	1	1	1	1011	112
0	110	0	0	0	0	0111	79
0	110	0	0	0	1	1111	78
0	110	0	0	1	0	1111	77
0	110	0	0	1	1	1111	76
0	110	0	1	0	0	0111	75
0	110	0	1	0	1	1111	74
0	110	0	1	1	0	1111	73
0	110	0	1	1	1	1111	72
0	110	1	0	0	0	0111	111
0	110	1	0	0	1	1111	110
0	110	1	0	1	0	1111	109
0	110	1	0	1	1	1111	108
0	110	1	1	0	0	0111	107
0	110	1	1	0	1	1111	106
0	110	1	1	1	0	1111	105
0	110	1	1	1	1	1111	104
0	111	0	0	0	0	0011	95
0	111	0	0	0	1	1011	94
0	111	0	0	1	0	1011	93
0	111	0	0	1	1	0111	92
0	111	0	1	0	0	1011	91
0	111	0	1	0	1	0111	90
0	111	0	1	1	0	0111	89
0	111	0	1	1	1	1111	88
0	111	1	0	0	0	0011	127
0	111	1	0	0	1	1011	126
0	111	1	0	1	0	1011	125
0	111	1	0	1	1	0111	124
0	111	1	1	0	0	1011	123
0	111	1	1	0	1	0111	122
0	111	1	1	1	0	0111	121
0	111	1	1	1	1	1111	120

BCD	AC	T1	BC	R	S	Y	ADDR
1	100	1	0	0	1	0001	254
1	100	1	0	1	0	0101	253
1	100	1	0	1	1	1001	252
1	100	1	1	0	0	0101	251
1	100	1	1	0	1	1001	250
1	100	1	1	1	0	1101	249
1	100	1	1	1	1	0101	248
1	101	0	0	0	0	1000	215
1	101	0	0	0	1	0000	214
1	101	0	0	1	0	0100	213
1	101	0	0	1	1	1000	212
1	101	0	1	0	0	0100	211
1	101	0	1	0	1	1000	210
1	101	0	1	1	0	1100	209
1	101	0	1	1	1	0100	208
1	101	1	0	0	0	1111	247
1	101	1	0	0	1	1111	246
1	101	1	0	1	0	1111	245
1	101	1	0	1	1	1111	244
1	101	1	1	0	0	1111	243
1	101	1	1	0	1	1111	242
1	101	1	1	1	0	1111	241
1	101	1	1	1	1	1111	240
1	110	0	0	0	0	1001	207
1	110	0	0	0	1	0001	206
1	110	0	0	1	0	0101	205
1	110	0	0	1	1	1001	204
1	110	0	1	0	0	0101	203
1	110	0	1	0	1	1001	202
1	110	0	1	1	0	1101	201
1	110	0	1	1	1	0101	200
1	110	1	0	0	0	1000	239
1	110	1	0	0	1	0000	238
1	110	1	0	1	0	0100	237
1	110	1	0	1	1	1000	236
1	110	1	1	0	0	0100	235
1	110	1	1	0	1	1000	234
1	110	1	1	1	0	1100	233
1	110	1	1	1	1	0100	232
1	111	0	0	0	0	1111	199
1	111	0	0	0	1	1111	198
1	111	0	0	1	0	1111	197
1	111	0	0	1	1	1111	196
1	111	0	1	0	0	1111	195
1	111	0	1	0	1	1111	194
1	111	0	1	1	0	1111	193
1	111	0	1	1	1	1111	192
1	111	1	0	0	0	1111	231
1	111	1	0	0	1	1111	230
1	111	1	0	1	0	1111	229
1	111	1	0	1	1	1111	228
1	111	1	1	0	0	1111	227
1	111	1	1	0	1	1111	226
1	111	1	1	1	0	1111	225
1	111	1	1	1	1	1111	224

D.2.5 Listing of the BCD ROM

Decimal ALU functions are implemented in a second 256x4-bit ROM. It operates on the A- and T-register bits 1, 2, and 3. The inputs for this ROM are the said A-register bits and three additional outputs from the first ALU ROM as partial result from T-register bits. The outputs are the result bits Z 1, 2, and 3, which go back to the A-register and the decimal carry DC. The result bit 0 is the same as in binary operations Y0 and comes from the first ALU ROM. In decimal operations only the DC carry is used whereas binary operations only use BC.

The following table lists the output values of the BCD ROM in column Z. The input values are in columns UTR, AC (which means T2, T3, and AC2), Y, and T1. The ADDR column gives the combined ROM adress from the input values.

UTR	AC	Y	T1	Z	ADDR
0	000	000	0	0000	128
0	000	000	1	0010	129
0	000	001	0	0100	130
0	000	001	1	0110	131
0	000	010	0	1000	132
0	000	010	1	1111	133
0	000	011	0	1111	134
0	000	011	1	1111	135
0	000	100	0	0010	192
0	000	100	1	0100	193
0	000	101	0	0110	194
0	000	101	1	1000	195
0	000	110	0	0001	196
0	000	110	1	1111	197
0	000	111	0	1111	198
0	000	111	1	1111	199
0	001	000	0	0010	136
0	001	000	1	0100	137
0	001	001	0	0110	138
0	001	001	1	1000	139
0	001	010	0	0001	140
0	001	010	1	1111	141
0	001	011	0	1111	142
0	001	011	1	1111	143
0	001	100	0	0100	200
0	001	100	1	0110	201
0	001	101	0	1000	202
0	001	101	1	0001	203
0	001	110	0	0011	204
0	001	110	1	1111	205
0	001	111	0	1111	206
0	001	111	1	1111	207
0	010	000	0	0100	144
0	010	000	1	0110	145
0	010	001	0	1000	146
0	010	001	1	0001	147
0	010	010	0	0011	148
0	010	010	1	1111	149
0	010	011	0	1111	150
0	010	011	1	1111	151
0	010	100	0	0110	208
0	010	100	1	1000	209
0	010	101	0	0001	210
0	010	101	1	0011	211
0	010	110	0	0101	212
0	010	110	1	1111	213
0	010	111	0	1111	214
0	010	111	1	1111	215
0	011	000	0	0110	152
0	011	000	1	1000	153
0	011	001	0	0001	154
0	011	001	1	0011	155
0	011	010	0	0101	156

UTR	AC	Y	T1	Z	ADDR
1	000	000	0	1111	0
1	000	000	1	1111	1
1	000	001	0	1111	2
1	000	001	1	1111	3
1	000	010	0	1111	4
1	000	010	1	1111	5
1	000	011	0	1111	6
1	000	011	1	1111	7
1	000	100	0	1111	64
1	000	100	1	1111	65
1	000	101	0	1111	66
1	000	101	1	1111	67
1	000	110	0	1111	68
1	000	110	1	1111	69
1	000	111	0	1111	70
1	000	111	1	1111	71
1	001	000	0	1111	8
1	001	000	1	1111	9
1	001	001	0	1111	10
1	001	001	1	1111	11
1	001	010	0	1111	12
1	001	010	1	1111	13
1	001	011	0	1111	14
1	001	011	1	1111	15
1	001	100	0	1111	72
1	001	100	1	1111	73
1	001	101	0	1111	74
1	001	101	1	1111	75
1	001	110	0	1111	76
1	001	110	1	1111	77
1	001	111	0	1111	78
1	001	111	1	1111	79
1	010	000	0	1111	16
1	010	000	1	1111	17
1	010	001	0	1111	18
1	010	001	1	1111	19
1	010	010	0	1111	20
1	010	010	1	1111	21
1	010	011	0	1111	22
1	010	011	1	1111	23
1	010	100	0	1111	80
1	010	100	1	1111	81
1	010	101	0	1111	82
1	010	101	1	1111	83
1	010	110	0	1111	84
1	010	110	1	1111	85
1	010	111	0	1111	86
1	010	111	1	1111	87
1	011	000	0	1111	24
1	011	000	1	1111	25
1	011	001	0	1111	26
1	011	001	1	1111	27
1	011	010	0	1111	28

UTR	AC	Y	T1	Z	ADDR
0	011	010	1	1111	157
0	011	011	0	1111	158
0	011	011	1	1111	159
0	011	100	0	1000	216
0	011	100	1	0001	217
0	011	101	0	0011	218
0	011	101	1	0101	219
0	011	110	0	0111	220
0	011	110	1	1111	221
0	011	111	0	1111	222
0	011	111	1	1111	223
0	100	000	0	1000	160
0	100	000	1	0001	161
0	100	001	0	0011	162
0	100	001	1	0101	163
0	100	010	0	0111	164
0	100	010	1	1111	165
0	100	011	0	1111	166
0	100	011	1	1111	167
0	100	100	0	0001	224
0	100	100	1	0011	225
0	100	101	0	0101	226
0	100	101	1	0111	227
0	100	110	0	1001	228
0	100	110	1	1111	229
0	100	111	0	1111	230
0	100	111	1	1111	231
0	101	000	0	1111	168
0	101	000	1	1111	169
0	101	001	0	1111	170
0	101	001	1	1111	171
0	101	010	0	1111	172
0	101	010	1	1111	173
0	101	011	0	1111	174
0	101	011	1	1111	175
0	101	100	0	1111	232
0	101	100	1	1111	233
0	101	101	0	1111	234
0	101	101	1	1111	235
0	101	110	0	1111	236
0	101	110	1	1111	237
0	101	111	0	1111	238
0	101	111	1	1111	239
0	110	000	0	1111	176
0	110	000	1	1111	177
0	110	001	0	1111	178
0	110	001	1	1111	179
0	110	010	0	1111	180
0	110	010	1	1111	181
0	110	011	0	1111	182
0	110	011	1	1111	183
0	110	100	0	1111	240
0	110	100	1	1111	241
0	110	101	0	1111	242
0	110	101	1	1111	243
0	110	110	0	1111	244
0	110	110	1	1111	245
0	110	111	0	1111	246
0	110	111	1	1111	247
0	111	000	0	1111	184
0	111	000	1	1111	185
0	111	001	0	1111	186
0	111	001	1	1111	187
0	111	010	0	1111	188
0	111	010	1	1111	189
0	111	011	0	1111	190
0	111	011	1	1111	191
0	111	100	0	1111	248
0	111	100	1	1111	249
0	111	101	0	1111	250
0	111	101	1	1111	251
0	111	110	0	1111	252
0	111	110	1	1111	253
0	111	111	0	1111	254
0	111	111	1	1111	255

UTR	AC	Y	T1	Z	ADDR
1	011	010	1	1111	29
1	011	011	0	1111	30
1	011	011	1	1111	31
1	011	100	0	1111	88
1	011	100	1	1111	89
1	011	101	0	1111	90
1	011	101	1	1111	91
1	011	110	0	1111	92
1	011	110	1	1111	93
1	011	111	0	1111	94
1	011	111	1	1111	95
1	100	000	0	1111	32
1	100	000	1	1111	33
1	100	001	0	1111	34
1	100	001	1	1111	35
1	100	010	0	1111	36
1	100	010	1	1111	37
1	100	011	0	1111	38
1	100	011	1	1111	39
1	100	100	0	1111	96
1	100	100	1	1111	97
1	100	101	0	1111	98
1	100	101	1	1111	99
1	100	110	0	1111	100
1	100	110	1	1111	101
1	100	111	0	1111	102
1	100	111	1	1111	103
1	101	000	0	1111	40
1	101	000	1	1111	41
1	101	001	0	1111	42
1	101	001	1	1111	43
1	101	010	0	1111	44
1	101	010	1	1111	45
1	101	011	0	1111	46
1	101	011	1	1111	47
1	101	100	0	1111	104
1	101	100	1	1111	105
1	101	101	0	1111	106
1	101	101	1	1111	107
1	101	110	0	1111	108
1	101	110	1	1111	109
1	101	111	0	1111	110
1	101	111	1	1111	111
1	110	000	0	1111	48
1	110	000	1	1111	49
1	110	001	0	1111	50
1	110	001	1	1111	51
1	110	010	0	1111	52
1	110	010	1	1111	53
1	110	011	0	1111	54
1	110	011	1	1111	55
1	110	100	0	1111	112
1	110	100	1	1111	113
1	110	101	0	1111	114
1	110	101	1	1111	115
1	110	110	0	1111	116
1	110	110	1	1111	117
1	110	111	0	1111	118
1	110	111	1	1111	119
1	111	000	0	1111	56
1	111	000	1	1111	57
1	111	001	0	1111	58
1	111	001	1	1111	59
1	111	010	0	1111	60
1	111	010	1	1111	61
1	111	011	0	1111	62
1	111	011	1	1111	63
1	111	100	0	1111	120
1	111	100	1	1111	121
1	111	101	0	1111	122
1	111	101	1	1111	123
1	111	110	0	1111	124
1	111	110	1	1111	125
1	111	111	0	1111	126
1	111	111	1	1111	127

D.3 Implemention of the CPU Emulation

As already mentioned, for execution of the micro-program it was necessary to program much of the CPU hardware functionality in Java code. Especially the shift registers, connected with serial buses, several flip-flops, logic gates, decoders, and status engines had to be simulated. The general problem in simulating digital electronic components is, that in real hardware many things happen at the same time and the state of the whole system depends on various interdependent signals connected by logic gates. In the real hardware the complete system state changes in discrete steps, triggered by clock pulses. During one clock cycle the state is usually stable and changes rapidly (in this case in some ten nanosecods) between consecuting clock pulses. But in a 'classical', non-parallel programming language actions may only be executed serially and dependencies between different components and signals have to be carefully scrutinized and resolved.

The main center of action is the CPU with its seven registers which are shifted at the same time and feed data through three buses and the ALU into each other. The data path is dynamically and controlled by the micro-operations. To resolve the interdependencies between these components, a state engine was implemented in the emulator, which is described in the following.

Each CPU register is programmed as Java class 'Register' with a 16 bit wide value (except E-register, which is 4 bits wide). The class has the following methods:

- setSource() – sets the source bus or register for shifting into the register as object variable
- loadInput() – stores the lowest bit of the source in a variable 'inputBit'
- getOutput() – returns the lowest bit of this register value
- shiftEnable() – enables shifting of this register value when used in a micro-op
- shift() – shifts the complete register value right by one bit and loads the inputBit in the highest bit position. The lowest bit of the register is lost.

Here is the Java code snipped from class Register:

```
// source Register or Bus for shift operation
void setSource(Register src)
{
     this.src = src;
     src.shiftEnabled = false; // shift must be enabled separately
}

// load input bit (LSB) from source
void loadInput()
{
     // load only if shift enabled
     if(shiftEnabled)
           inputBit = src.getOutput();
}

// return output bit (LSB) of register
int getOutput()
{
     return(value & 1);
}

// enable and disable shifting
```

```
  void shiftEnable(boolean enable) {
       shiftEnabled = enable;
  }

  void shift()
  {
       // shift only if enabled
       if(shiftEnabled) {
             // shift register 1 bit right and set input bit (MSB)
             value = (inputBit << width | value) >> 1;
       }
  }
}
```

If the source of a register is set to its own object, shifting recircles the register value to itself (bit rotation). This is the default behavior for all registers, if no other source is assigned.

The R- and S-buses are implemented as Register variables and the T-bus is a 1 bit wide Register object. Additionally, for consistency of the program logic, there are two 1 bit wide pseudo-register objects defined, 'zero' and 'one', with constant values 0 and 1 respectively. These are assigned as bus sources in the micro-operations ZTS, UTS, ZTR, and UTR.

The execution of each micro-instruction is devided into several clock cycles, the number of these is defined by the micro-code CC.

- As preparation for a shift cycle the sources of the R- and S-bus are assigned to the register object variables defined by the micro-codes RC and SC.
- Next the destination register of the T-bus, defined by the micro-codes XC, TTM or TTT, is assigned by setting its source to the T-bus object variable.
- Then the shift counter is initialized by the micro-code CC.
- If micro-op is IOS, shifting is inhibited by setting the counter to -1.
- If micro-op is IQN, the counter is set to -1, if the indicated qualifier test is not met.
- After this prepation for each shift count the following loop is executed:
  - The ALU operation is executed, using R- and S-bus getOutput() values as input. The result is stored in the T-bus value.
  - If applicable, BCD operation is executed using A- and T-register values as input.
  - If micro-op is BRC, the indicated qualifier will be tested and the result memorized.
  - If the loop is finished or shifting is inhibited (count = -1), the loop will be left now.
  - Otherwise for all involved registers the loadInput() method is excuted. This also loads the T-bus value in its destination register.
  - Then all involved registers are shifted by calling their shift() method.
  - At last the shift counter is decremented and the loop continued.
- If a branch has been detected above, the lowest bit of the secondary adress modifier SM will be set to 0, which effectively changes the next address.
- The next micro-program address will be computed by XOR-ing the current address with the PM and SM modifiers.

- At the end of the execution the sources of all registers are reset to themselves (recycle) for the next instruction.

The description above is somewhat simplified but shows how the parallel actions during a shift cycle maybe serialized, conserving logical dependencies. The execution order of several operations, like TQ6, and the carry handling is crucial to give correct results. For example, the clock signal ROMCLK changes in the real hardware at the end of the shift cycle from 1 to 0, which triggers loading of the BCD result into the A register and the DC carry flip-flop. So this functions may only be executed after finishing the shift loop.

Moreover, several micro-instructions define more than one source for a certain bus. For example XTR (A or B register to R-bus) and ZTR (zero to R-bus) are often combined in one instruction. What seems to be not logical causes no problem in the real hardware, because all possible sources of R- or S-buses are logically or-ed. And any value or-ed with 0 gives the unaltered value. In the CPU emulation there is only one possible source for each bus, so care must be taken, that due to the serial execution of micro-operations, ZTR doesn't overwrite the source assignment by XTR. To accomplish this, ZTR.exec() checks first, if there is already another source defined for the R-bus. If yes (the source object variable is not empty), it exits without further action. If not, the pseudo-register zero is assigned as source. The otherwise rather senseless combination XTR and UTR (one to R-bus) defines a new, derived operation TQR, and neither XTR nor UTR are executed.

There is also one case, instruction 1216, where two destinations of the T-bus are defined by TTM and TTT. This is also covered by the implementation, because the source (T-bus pseudo register) of M- and T-register may be set independently and will be properly evaluated for both.

Further informations maybe drawn from the commented source code of the Java class CPU.

D.4 Pre-decoding of Micro-Instructions

As mentioned the emulation performance is rather low due to the serial buses and execution of micro-instructions. To get things not even worse by doing a complex decoding of the binary micro-code with lots of case distictions for every instruction, this decoding is done only once during initialization of the CPU class. Every executable micro-operation is implemented as a Java class with only one method: exec(). Micro-instructions, which are made up of micro-ops, are also implemented as Java class, with object variables for each micro-op group. The instruction decoder parses the binary micro-code of an instruction and for each micro-op assigns the proper objects to the variables of the instruction. As a byproduct the decoded micro-op mnemonics and values are stored as in a string variable for use in trace and single step execution (see appendix C.4).

When it comes to execution of a micro-instruction, now only the exec() method of each micro-op object variable has to be called. This is much faster than decoding and simplifies the source code. For example the class for the micro-operation XTR looks like this:

```
class XTR extends MicroOperation
{
  /*
   *  A/B-register is shifted into R-bus
   */
  public XTR()
  {
    name = "XTR";
```

```
    }

    public void exec()
    {
      Rbus = ABselector ? Bregister : Aregister;
      Rbus.shiftEnable(true);
    }
  }
```

D.5 Pre-decoding of ALU and BCD functions

To save execution time of the binary and decimal ALU functions, the contents of ALU and BCD ROM are also predecoded during CPU initialization, but in a different way as for micro-instructions. In the real hardware, the ROM addressing is somewhat complex, as some adress bits have to be inverted and exchanged in order. So calculation of the ROM adress from the input values (ALU-code, T-register, carry bit etc.) requires complex shift and or operations to bring the adress bits into the required order. To minimize these operations during execution, the ROM contents are stored reordered in an array so that the adress computation is merely done by concatenating the bits of

- ALU code (3 bits),
- R-bus (1 bit),
- S-bus (1 bit),
- BCD flag (1 bit),
- T-register bit 1,
- binary carry (1 bit)

to a 8 bit adress value.

The array for the 4-bit BCD ROM is also reordered and the address is concatenated from

- A-register bits 3,2,1,
- ALU output bits 2,1,0,
- T-register bit 1,
- UTR (1 bit).

Again as a byproduct the decoded mnemonics are generated as text string, which may be output to the console in the emulators debug mode (see appendix F).

D.6 Further optimizations

A close look to the micro-program listing in D.2.2 shows, that several instructions don't involve any of the CPU registers, but require several shift clocks. For example this is the case for instructions 0001, 0106, and 0215. Especially all memory read and write operations require 12 shift clocks just as a delay for the memory hardware, which is not necessary for the emulation.To speed up the emulator, for all micro-instructions which don't require any registers to be shifted, the shifting is skipped. This increases the mean execution speed by about 25%. In the micro-code mode of the disassembler (see appendix C.4) as well as in the micro-program dump, which is generated in debug mode of the emulator (see appendix F), these instructions are marked with an asterisk (*) right of the clock value in column C.

E The HP2116 Control Panel

The HP9800 family of calculators uses a CPU architecture and instruction set derived from and similar to the HP21xx minicomputer series. These early machines had a fancy control panel with blinking lights displaying the contents of the CPU registers A, B, M, T, and P, which also exist in the HP9800 CPU. So just for fun and without any practical use the control panel of the first HP2116 minicomputer was implemented in the GO9800 emulator.

The control panel may be opened by pressing the Ctrl+C keys.

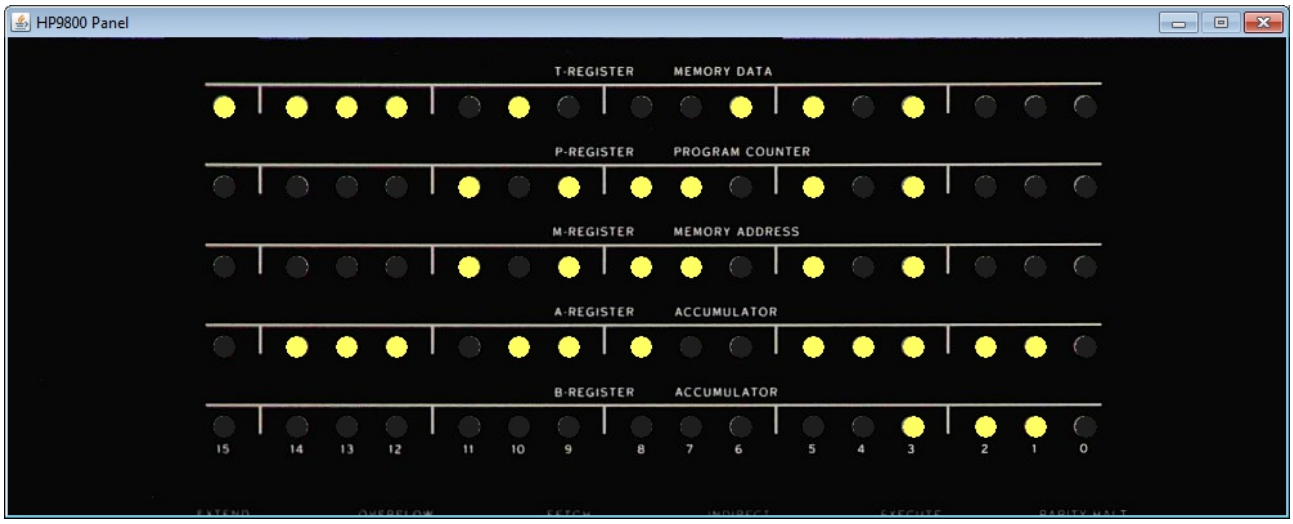


Figure E-1: HP9800 control panel in HP2116 style.

It displays the current contents of the five said registers in binary form. The display is updated only every 10 ms to keep the CPU load low and to get nice blinking lamps, like in those old-fashioned mainframe panels.

Together with the disassembler console in trace or single step mode, the HP2116 panel displays the same values as shown in the console window.

F Debug-Mode of the Emulator

The emulator may be started in debug mode by giving the command-line option -d, e.g.

```
java -jar GO9800.jar -d HP9810A
```

With this option given, several operations of the basic machine and some internal and peripheral devices print debug informations about their activity to the console window. This was used during development to identify problems in the communication between CPU and devices. Otherwise it may be used just for curiosity or to understand the functionality or data formats of peripheral devices.

Additionally, in debug mode, the complete decoded micro-program ROM is dumped to the console after startup of the emulator, followed by the decoded ALU and BCD ROMs.

Figure F-1: Console in debug mode with part of the decoded micro-program.

The micro-code, ALU, and BCD ROMs are described in detail in appendix D.

Informations about debug output of other devices, such as the internal magnetic card reader, tape drive, and more are available on request from the author.