USERS' GUIDE FOR ee9: AN ENGLISH ELECTRIC KDF9 EMULATOR

by **BILL FINDLAY**
(kdf9@findlayw.plus.com)

0: INTRODUCTION

This note is a guide for users of **ee9**, a program emulating the EE KDF9 computer. Readers not yet familiar with the KDF9 should consult the companion document, *The English Electric KDF9*.

ee9 is intended to be portable to any system that offers a basic POSIX API. It is written in Ada 2005 using the GNU Ada compiler, GNAT GPL. To date, **ee9** has been implemented on the Intel x86_64 and PowerPC G5 architectures under OS X; on the Intel x86_32 and x86_64 architectures under Linux/FreeBSD; on the ARM11 architecture for the Raspberry Pi under Raspbian (Debian Linux for ARM); and on the Intel x86_32 architecture under Microsoft Windows (XP/SP3 or newer). For the particular characteristics of this version of **ee9**, see §6. Note that the command line syntax for this version of **ee9** differs from that of all previous versions.

1: ee9 COMMAND SUMMARY

The emulator is invoked from the command line, thus:

./ee9 { **-s***s* | **-d***d* | **-m**[*m*] } < *program_file_name* >TP0

where the **–** flag parameters are optional and can be given in any number or order; *m* is a short string that specifies a **m**iscellany of options; *d* specifies a **d**iagnostic execution mode; and *s* is the initial CPU **s**tate for the KDF9 run.

The allowable state flag characters *s* are:

- **b**: for **b**ooting into Director state, which is how operating systems are loaded and run
- **p**: for **p**roblem **p**rogram state, the default, allowing user programs to be run without a Director (see §2.2)
- **t**: for **t**est program state, allowing programs to be run with OUTs serviced as in problem program state, but with the CPU actually in Director state; though inauthentic, this is useful for running 'hardware' test programs.

The allowable diagnostic flag characters *d* are:

- **f**: for **f**ast mode, the default
- **p**: for **p**ause mode
- **t**: for **t**race mode
- **x** for e**x**ternal trace mode (see §2.1)

The allowable characters in the string *m* are described in §5.2.

Commands are available to simplify calls on **ee9**, in systems that support a bash-compatible shell. See Appendix 1.

EXAMPLES

```
./ee9 –dt –mn KMW0201–UPU    # KMW0201–UPU is the Walgol compiler
./ee9 –sb  KKT40E007UPU      # KKT40E007UPU is the Timesharing Director
```

2: EMULATION MODES

2.1: DIAGNOSTIC MODES

A KDF9 program is run, at option, in one of four **diagnostic modes**. These are:

- **fast** mode; in which **ee9** runs the program at maximum speed, with no execution tracing or interactive diagnostic facilities available
- **pause** mode, in which **ee9** single-shots the program, pausing to interact with the user after each instruction
- **trace** mode, in which **ee9** runs the program at speed with extensive retrospective tracing enabled
- **external** trace mode, in which **ee9** writes a summary of every traced instruction to an external file

More precisely, things work as follows.

In fast mode **ee9** interacts with the user only by providing informative messages, either because the KDF9 program has terminated, or to log significant events during the run (such as the allocation of an I/O device). All tracing overhead is avoided in fast mode.

In pause mode **ee9** uses console-window text I/O to interact with the user. After each instruction is executed a short summary of the machine state is displayed and a prompt asks the user how to continue. The user replies with an optional single letter (which may be given in upper case or lower case) followed by RETURN, selecting one of the following:

- **f**: execution proceeds in **f**ast mode
- **p**: execution proceeds in **p**ause mode
- **t**: execution proceeds in **t**race mode
- (nothing): execution proceeds in the current mode.

All retrospective tracing types described in §4 are available in pause mode, trace mode, and external trace mode; but the manner of execution depends on whether the current instruction execution lies within a set range of addresses, and within set instruction-count bounds. If so, instructions are added to their appropriate traces; and breakpoints and watchpoints are monitored. If not, execution proceeds as in fast mode (but at about a third of the speed).

2.2: Run states

The run state specifies how the emulated KDF9 is to run the program:

- In **boot** mode the KDF9 reads a 9-word bootstrap routine from `TR0`, then jumps to word 0, in **Director** state.
- In **problem program** mode **ee9** reads into core, from `TR0`, a binary program prepared by a compiler (such as David Holdsworth's new Usercode cross-compiler). Its execution starts at word 0, in **program** state. **ee9** itself implements any `OUT`s requested by the program, so that it is not necessary to have a Director running.
- In **test program** mode **ee9** reads a binary program into core from `TR0`, just as in problem program mode. Its execution starts at word 0, but in **Director** state. The emulator implements any `OUT`s executed by the program.

2.3: Breakpoints, Fetchpoints , and Storepoints

Certain addresses in core can be marked as breakpoints or as watchpoints, to force diagnostic interaction with the user. A **breakpoint** is set on an instruction word, and causes interaction after an instruction beginning in that word has been executed. A **fetchpoint** is set on a data word, and causes interaction after data has been fetched from that word. A **storepoint** is set on a data word, and causes interaction after data has been stored into that word. A **watchpoint** combines a fetchpoint and a storepoint on the same word.

2.4: Authentic Timing Mode

At option, **ee9** can be made to insert timed pauses into its execution so that the elapsed time of a program run by **ee9** approximates the elapsed time of a run on the KDF9 hardware. This may be instructive for younger users, who have never seen characters being output by a computer, one at a time, and with noticeable delays! This mode can be set using the authenticity option setting or by means of the command-line miscellany parameter; see under '**A**' in §5.

3: Inputs and Outputs

3.1: Emulated KDF9 I/O Devices

At the start of a run **ee9** casts around for files to represent the virtual KDF9 peripherals. If no file can be found for a peripheral, it may be reported to be 'offline'. There are fixed assignments for the console Flexowriter, which is associated with the user's interactive terminal window; for paper tape reader 0, which is associated with the standard input; and for paper tape punch 0, which is associated with the standard output.

Other devices are associated with files having names derived from the device type. Magnetic tape deck *d*, for example, is always associated with the file named '`MT`*d*'. It will often be convenient to have file system *links* of these names, which may be redirected for each run of the emulator to the actual data files to be processed on that occasion. The full list of these associations is as follows:

- **c**ard **p**unches are '`CP`*d*'
- **c**ard **r**eaders are '`CR`*d*'
- **dr**um stores are '`DR`*d*'
- **f**ixed **d**isc stores are '`FD`*d*'
- **g**raph **p**lotters are '`GP`*d*'
- **l**ine **p**rinters are '`LP`*d*'
- KDF9 **m**agnetic **t**ape decks are '`MT`*d*'
- IBM **s**even-**t**rack tape decks are '`ST`*d*'
- paper **t**ape **p**unches are '`TP`*d*'
- paper **t**ape **r**eaders are '`TR`*d*'

3.2: The Flexowriter Console Typewriter

The terminal window is the means by which users, in their rôle as KDF9 operators, can mimic Flexowriter I/O. The Flexowriter is used to type-in responses to prompts output by problem programs or by Director. Repeatedly typing these responses quickly becomes tedious. If a file named `FW0` exists, it is used as a source of "canned" responses. They are defined, with their identifying prompts, in `FW0`; and are picked up automatically by **ee9**. If a prompt spreads over more than one line, a KDF9 Line Shift can be represented in `FW0` by a '®', and a KDF9 Page Change by a '©'.

When a prompt is issued, **ee9** scans `FW0`, down from the last match found. If it finds a new match, it injects the given response into the Flexowriter input stream; but if it reaches the end of the file without finding a match, it returns control of the Flexowriter to the user's terminal window, so that a manual response can be given. If a prompt matches a line in `FW0` that specifies a null response string (c.f. the second '`OUT;`' in the following example) then **ee9** terminates the run.

For example, the Whetstone Algol compiler prompts '`OUT;`' to which a typical reply is '`N.|`'. If the Algol program compiles, it runs and prompts '`STREAM;`' to which a typical reply is '`30.|`'; but if the compilation fails the compiler loops back to its '`OUT;`' prompt, where the user will normally want to terminate the run so that the Algol source code can be amended. The following data in `FW0` will achieve this without user intervention:

```
OUT;N.|
STREAM;30.|
OUT;
```

For a second example, as the Time Sharing Director bootstraps into action it issues a series of requests for basic configuration parameters. The following data in `FW0` supplies suitable responses without user intervention:

```
CORE MODULES;8.|
OUT 8 REEL NO;9.|
LEVELS;N.|
DATE  D/M/Y;4/5/67.|
TIME ON 24-HOUR CLOCK®HOURS/MINS;1/23.|
```

This facility had a real equivalent: the Flexowriter incorporated an 'edge-punched card' reader. It read data (in paper tape code) from the edge of a non-standard punched card. Cards prepared with replies to prompts could be inserted into the reader and read at the maximum rate, thus speeding input and avoiding any delay due to typing errors by the operator.

Note that ee9 requires every Flexowriter input string to be terminated by a RETURN, even when a read-to-End Message instruction is being obeyed. In reality, KDF9 would end the transfer immediately at the End Message, or when the required number of characters had been read; but data is not transferred to **ee9**'s input buffer until a RETURN is typed. A purely terminating RETURN is discarded from the input buffer by **ee9**, and is not passed to the KDF9 program.

In response to CTRL-C, **ee9** outputs a prompt of its own that lets the diagnostic mode be changed. Replying with a RETURN (only) causes a FLEX interrupt; when running Director in boot mode, this evokes a '`TINT;`' prompt.

Output to the KDF9 Flexowriter was typed in red; input from the computer operators was typed in black. This is simulated in **ee9** by using ANSI-terminal escape sequences to vary the displayed font colour. The Windows **cmd** command-line utility does not implement ANSI terminal escape sequences, so Flexowriter I/O under Windows is monochrome.

3.3: Reading More Than One Roll of Paper Tape or Deck of Cards

A means is provided to simulate the way in which KDF9's computer operators could satisfy a program's demand for data with several physically-separate rolls of paper tape, loaded into a tape reader in succession. If a program attempts to read from a tape reader, and the end of the associated file has been reached, **ee9** allows the user to specify a successor file to which the paper tape reader is re-attached. These files are named '`TR`*dr*' where *d* is the device number (0 or 1) and *r* is a letter identifying the "roll of tape". On reaching the end of the current file, **ee9** asks for the next letter *r*; if none is given the reader is left in the 'abnormal' condition and any further attempt to read from it provokes a parity error. Again, it may be convenient for the files '`TR`*dr*' to be realized as links to actual data files with more mnemonic names. See the '**N**' option in §5 about the disabling of this feature in non-interactive mode.

The above also applies, *mutatis mutandis*, to the punched card reader. Lines of less than 80 characters are padded with blanks to fill all 80 columns of the card; any line longer than a card is truncated. In 'direct' mode, lines may have up to 160 characters, notionally two per column. Any attempt to read a character not in the card set causes a parity error. (The card punch always generates files suitable as input to the card reader.)

3.4: Representing the KDF9 Character Sets

External data is read and written in the ISO Latin-1 character set, with automatic conversion between Latin-1 and KDF9's internal character codes (which are somewhat device-dependent). Several graphic characters in the KDF9 paper tape set are absent from Latin-1, so a simple transliteration is used to represent them externally. See Appendix 2. The break character is used for the non-escaping KDF9 underline, so that an Algol 60 reserved word such as '**real**', seen on KDF9 as '<u>`real`</u>', appears as '`_r_e_a_l`', and an underlined End Message, '<u>→</u>', appears as '`_|`'.

In the case of the Flexowriter, tape punches, and tape readers, Case Normal and Case Shift characters are generated on input, and interpreted on output. This means that when you are typing an input text, it is not necessary to type Case Normal and Case Shift characters, although it does no harm to do so. When such a text is being read as the input stream for a two-shift device, an appropriate case-character is generated automatically by the emulator, if the Latin-1 character being read is not available in the input device's current shift state. Two-shift devices always start out in the Case Normal condition. For example, the external Latin-1 string '`Bill Findlay`' is read into the KDF9 core store as the characters '`BßILL ñFßINDLAY`', with ß denoting the Case Shift character and ñ denoting the Case Normal character. A KDF9 program that writes the characters '`BßILL ñFßINDLAY`' to a two-shift device will generate the Latin-1 string '`Bill Findlay`' as its external representation.

Text-file input to **ee9** may use any of `CR`, `LF`, or a `CRLF` pair as the line terminator: **ee9** treats all three the same. Text-file output from **ee9** writes the line terminator most appropriate for the host OS.

Non-graphic KDF9 characters also have Latin-1 external representations, to enable faithful 1-to-1 conversion between the internal and external data formats. Apart from the format effectors (Horizontal Tab, New Line, Form Feed), users should never need to type these characters, as they could not be typed on a Flexowriter.

Characters are displayed **in tracing output and core dumps** using the line printer code, except as follows:

- the KDF9 Tab character is represented by ¬
- the KDF9 Carriage Return character is represented by ®
- the KDF9 Page Change character is represented by ©
- the KDF9 Filler, and other non-legible characters, are represented by Ø

Bootable Directors and compiled problem programs are not encoded in Latin-1, but natively, in the KDF9 paper tape code. They use an 8-bit byte to encode 6 data bits; 8 of these bytes are packed into a 48-bit KDF9 word.

3.5: Graph Plotting

ee9 includes an emulation of the model 564 Calcomp graph plotter, as described in Appendix 6, §5, p.302 of the Manual. There was provision on the KDF9 to switch a buffer manually between a tape punch and a graph plotter; in **ee9** this is done with a settings file option, **G**, or by including **g** in the *miscellany* parameter. When either of these is given, GP0 replaces TP1 on the shared buffer.

The KDF9 graph plotter takes commands that move the plotting position in steps of 0.005 inches; see Appendix 3 for the equivalent character codes. These are accumulated into vectors by **ee9**, and PostScript vector drawing commands are output to the GP0 file. It is possible to 'fit' the plotter with pens having a variety of ink colour and ball-point tip size. See under '**G**' in §5.

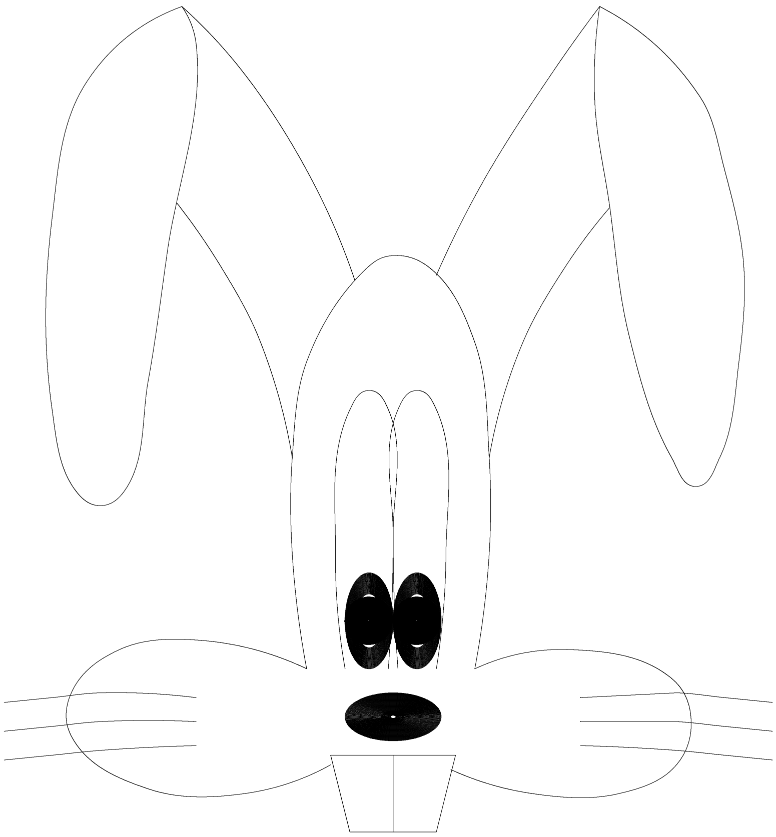
This program:

```
V9; W0;
RESTART; J999; J999;
PROGRAM;
   V1 = B20;         (plotter pen-down command);
   V2 = Q0/AV1/AV1;  (to plot data);
   V3 = Q0/AV1/AV1;  (to read data);
   V4 = B02;         (TR device-type code);
   V5 = B20;         (GP device-type code);

   V4; SET 5; OUT;   (claim TR);
   V3; =Q3; =C3;     (set up Q3 for TR input);
   V5; SET 5; OUT;   (claim GP);
   V2; =Q2; =C2;     (set up Q2 for GP output);
   POCQ2;            (pen down);
*1;
   PICQ3;            (read one plotting command from TR);
   PARQ3; J999TR;    (exit loop to 999 at EOF);
   POCQ2;            (write one plotting command to GP);
   J1;
999;
   ZERO; OUT;        (end run);
FINISH;|
```

can be run at the command line as follows (edited for convenience):

```
/Users/wf/KDF9/emulation/Testing: nine TR2GP wabbit_kdf9 - g
Welcome to ee9 V2.0q, the GNU Ada KDF9 emulator.
The shared buffer has been switched from TP1 to GP0.
...
ee9: OUT 5: requests a device of type #02; gets TR1.
ee9: OUT 5: requests a device of type #20; gets GP0.
...
ee9: OUT 0: end of run.
________________________________________________________________________
Final State:

At #00032/1 (26/1); ICR = 1688326; the instruction was #200:220:000, i.e. OUT
CIA:          #00032/1 (26/1)
NIA:          #00032/4 (26/4)
ORDERS:        1688326 executed (ICR)
CPU TIME:     25465440 KDF9 us. (RAN)
CLOCK TIME: 1732422678 KDF9 us. (EL)

The SJNS is empty.

Q store:
 Q2: Q #000003/ #000011/ #000011  = Q      3/      9/      9
 Q3: Q #000002/ #000011/ #000011  = Q      2/      9/      9

The NEST is empty.
________________________________________________________________________
End of Run.
TR1 on buffer #02 read 281384 character(s).
GP0 on buffer #03 plotted 281385 character(s).
________________________________________________________________________
```

It copies the file of plotter commands named `wabbit_kdf9.txt` from a tape reader to the graph plotter, producing the following charming portrait (after conversion to PDF from the output Encapsulated PostScript—EPS— format):



4: TRACING AND LOGGING

Messages that record the progress of the emulation, and details of any errors that were detected, are written to the interactive console window, along with interactive diagnostics and output intended for the KDF9 Flexowriter. A selection of these messages is also written to the file `KDF9_log.txt`. On completion of a run, the final machine state, any requested core store areas, and any retrospective traces may be written to the log file and to the console window.

It is possible to request the output of certain areas of the KDF9's core store, in a variety of suitable formats. These printouts can be taken either before the start of execution; or on termination; or at both times, to allow comparisons.

The tracing of instructions is subject to instruction-count and address-range bounds. Instruction executions within those bounds are traced; those that fall outside the bounds are not.

In the **interrupt** trace, which is produced only in boot mode, interrupt requests are listed with the privilege state and priority of the interrupting device; the elapsed time of occurrence (in μs); and the value of `ICR`, the Instruction Count Register, which is a count of the number of instructions executed so far. See, e.g.:

```
Retrospective trace of interrupt requests.
                                                   CPL     EL. TIME    ICR
Ended #03455/2: EDT                              D 0     @ 69589893    3376330
After #03555/3: EDT                              D 0     @ 69589259    3376231
After #03555/3: EDT                              D 0     @ 69588608    3376134
After #02534/3: FLEX                             D 0     @ 69578533    3374471
...
After earlier interrupts, whose tracing is now lost.
```

In the **peripheral I/O trace**, the events shown are transfer initiations and terminations, busy-buffer and store-access lockouts, and I/O status test operations. Each is listed with the device name, Q-store parameter, privilege state (`P` for problem program state and `D` for Director state) and priority of the transfer; the elapsed time of occurrence of the event; and the value of `ICR`. The C part of the parameter used in a `FD` seek operation is logged in the format `D`*d*`P`*pp*`S`*ss*, where *d*, *pp* and *ss* are, respectively, the drive number, platter number and seek area number being addressed. In a `FD` data transfer operation *d* and *pp* are irrelevant and *ss* is the starting sector number for the transfer.

Transfer operations appear twice, once for the initiation (`S`) and once for the termination (`E`).

Lockouts appear once, when they happen.

A test operation gives the result of the test as a Boolean. See, e.g.:

```
Retrospective trace of peripheral I/O events.
                                                   CPL     EL. TIME    ICR
Ended #00021/5: POEQ13       TP0 Q#4/#0/#454       P 0     S 1654305064  306451950
After #00132/5: POBQ14       TP0 Q#4/#72235/#72432  P 0     E 1654305003  306451936
After #00133/1: E#72235M7Q   TP0 Store Lockout at #72235    @ 1654295945  306451936
After #00132/5: POBQ14       TP0 Q#4/#72235/#72432  P 0     S 1654295913  306451936
After #00132/3: PIBQ15       TR1 Q#2/#72235/#72432  P 0     E 1654295913  306451934
After #00132/5: POBQ14       TR1 Store Lockout at #72235    @ 1654294945  306451934
After #00132/3: PIBQ15       TR1 Q#2/#72235/#72432  P 0     S 1654294913  306451934
After #00157/5: POAQ14       TP0 Q#4/#72235/#72235  P 0     E 1654294913  306451912
After #00132/3: PIBQ15       TP0 Store Lockout at #72235    @ 1654222370  306451932
After #00157/5: POAQ14       TP0 Q#4/#72235/#72235  P 0     S 1654222193  306451912
After #00132/5: POBQ14       TP0 Q#4/#72235/#72432  P 0     E 3907366     1493
After #00133/1: E#72235M7Q   TP0 Store Lockout at #72235    @ 2989908     1493
After #00132/5: POBQ14       TP0 Q#4/#72235/#72432  P 0     S 2989276     1493
After #00132/3: PIBQ15       TR1 Q#2/#72235/#72432  P 0     E 2989276     1491
After #00132/5: POBQ14       TR1 Store Lockout at #72235    @ 2888908     1491
After #00132/3: PIBQ15       TR1 Q#2/#72235/#72432  P 0     S 2888276     1491
After #00132/5: POBQ14       TP0 Q#4/#72235/#72432  P 0     E 2881121     145
After #00133/1: E#72235M7Q   TP0 Store Lockout at #72235    @ 2808475     145
After #00132/5: POBQ14       TP0 Q#4/#72235/#72432  P 0     S 2808401     145
After #00132/3: PIBQ15       TR1 Q#2/#72235/#72432  P 0     E 2808401     143
After #00132/5: POBQ14       TR1 Store Lockout at #72235    @ 2800475     143
After #00132/3: PIBQ15       TR1 Q#2/#72235/#72432  P 0     S 2800401     143
After #00132/5: POBQ14       TP0 Q#4/#72235/#72432  P 0     E 2799816     28
After #00133/1: E#72235M7Q   TP0 Store Lockout at #72235    @ 2727170     28
After #00132/5: POBQ14       TP0 Q#4/#72235/#72432  P 0     S 2727096     28
After #00020/0: POEQ13       TP0 Q#4/#0/#454       P 0     E 2727096     16
After #00132/3: PIBQ15       TR1 Q#2/#72235/#72432  P 0     E 8162        27
After #00132/5: POBQ14       TP0 Buffer Lockout             @ 236         27
After #00132/3: PIBQ15       TR1 Q#2/#72235/#72432  P 0     S 162         27
After #00020/0: POEQ13       TP0 Q#4/#0/#454       P 0     S 96          16
After #00000/0: #000         TR0 Q#1/#0/#17777     P 0     S 0           1
After the start of traced execution.
Total time waiting for unoverlapped I/O to finish = 3980ms.
```

In the **retro** trace, instructions are listed in order, starting with the most recently executed. The trace includes the instruction itself, and its most relevant operand; 'ND' and 'SD', the Nest and SJNS Depths; 'V' and/or 'T' showing whether overflow and/or the test register is set; the **CPU** time of occurrence of the event; and the value of `ICR`.

In the case of a store order, the traced operand is the value written to store. In the case of a fetch order, it is the value fetched. For a Q-store order, it is the content of the relevant Q register. For a conditional jump it is the determining value. For subroutine jump or exit, it is the relevant value in the SJNS. For a 1-syllable or 2-syllable ALU order, it is the value left in the top of the nest. And so on. See, e.g.:

```
Retrospective trace of all instructions.
                                                            ND SD VT  CPU TIME    ICR
Ended #00023/4: OUT            0                            3  0 V   1650324654  306451955
After #00023/3: ZERO          #0000000000000000             4  0 V   1650324641  306451954
After #00023/0: OUT            4                            3  0 V   1650324639  306451953
After #00022/3: SETB6         #0000000000000006             5  0 V   1650324626  306451952
After #00022/1: C13           #0000000000000004             4  0 V   1650324615  306451951
After #00021/5: POEQ13        Q#4/#0/#454                   3  0 V   1650324610  306451950
After #00136/5: EXIT 2        #00020/5                      3  0 V   1650324591  306451949
After #00136/2: J#00137/2NE   #0000000000000035             3  1 V   1650324572  306451948
After #00135/5: SETB35        #0000000000000035             4  1 V   1650324567  306451947
After #00135/2: J#00133/5LTZ  #0000000000000035             3  1 V   1650324563  306451946
After #00135/1: DUP           #0000000000000035             4  1 V   1650324559  306451945
After #00135/0: -             #0000000000000035             3  1 V   1650324557  306451944
After #00134/3: SETB40        #0000000000000040             4  1 V   1650324556  306451943
After #00134/1: SHLD+6        #0000000000000075             3  1 V   1650324545  306451942
After #00134/0: ZERO          #0000000000000000             3  1 V   1650324542  306451941
After #00133/5: ERASE         #7500000000000000             2  1 V   1650324540  306451940
After #00133/4: DUP           #7500000000000000             3  1 V   1650324539  306451939
After #00133/1: E#72235M7Q    #7500000000000000             2  1 V   1650324537  306451938
After #00132/5: POBQ14        Q#4/#72235/#72432             1  1 V   1650324530  306451936
After #00132/3: PIBQ15        Q#2/#72235/#72432             1  1 V   1650324498  306451934
After #00132/1: IM15TOQ14     Q4/#72235/#72432              1  1 V   1650324466  306451932
After #00131/5: =M15          Q2/#72235/#72432              1  1 V   1650324462  306451931
After #00131/4: +             #0000000000072432             2  1 V   1650324453  306451930
After #00131/2: I15           #0000000000072235             3  1 V   1650324452  306451929
After #00130/5: SETB175       #0000000000000175             2  1 V   1650324446  306451928
...
After #00067/0: J#00075/0     #00075/0                      3  1 V   1650323316  306451720
After #00066/0: EXITAR#00066/0  1                           3  1 V   1650323308  306451719
After #00065/3: E#252M3       #0073200337002007             3  2 V   1650323296  306451718
After #00065/0: E#253M3       #0067500321201506             2  2 V   1650323290  306451717
After #00064/4: =LINK         #00001/0                      1  2 V   1650323284  306451716
After #00064/2: M1            #0000000000000001             2  1 V   1650323274  306451715
After #00064/0: =M3           Q0/#0/#3752                   1  1 V   1650323270  306451714
After #00063/5: +             #0000000000003752             2  1 V   1650323268  306451713
After #00063/4: DUP           #0000000000001765             3  1 V   1650323267  306451712
After #00063/2: M2            #0000000000001765             2  1 V   1650323265  306451711
After earlier instructions, whose tracing is now lost.
```

External trace mode is like retro mode, with additional output to the file `trace.txt`. This output has one line for each traced instruction. It contains: the instruction's address; the value of `ICR`; the CPU time; the nest depth; the SJNS depth; 'V' and/or 'T' if overflow and/or the test register is set; the value in N1, if the nest if non-empty; and the disassembled instruction. For example:

```
LOCATION  ICR      CPU TIME         ND SD VT  [N1]              |INSTRUCTION
#00000/0  1        8                 0  0                       |J#00012/0
#00012/0  2        12                1  0     #0000000000000002 |SETB2
#00012/3  3        19                2  0     #0000000000000005 |SETB5
#00013/0  4        32                1  0     #0000000000000002 |OUT
#00013/3  5        35                0  0                       |=C15
...
#00236/2  3439084  19706693          1  2 V   #0000000000000004 |JS#00063/2
#00063/2  3439085  19706697          2  2 V   #0000000000001243 |M2
#00063/4  3439086  19706699          3  2 V   #0000000000001243 |DUP
#00063/5  3439087  19706703          2  2 V   #0000000000002506 |+
#00064/0  3439088  19706705          1  2 V   #0000000000000004 |=M3
#00064/2  3439089  19706709          2  2 V   #0000000000000001 |M1
#00064/4  3439090  19706715          1  3 V   #0000000000000004 |=LINK
#00065/0  3439091  19706721          2  3 V   #0000000000000000 |E#253M3
```

(*etc*)

When tracing, and if requested, **ee9** will tally the number of traced executions of each type of KDF9 instruction. On termination a HISTOGRAM of dynamic instruction-type frequencies is logged, grouped according to their first syllable, but with jump instructions further analysed according to bits 0:3 of their second syllable. Output is along these lines:

```
Histogram of 74907338 executed instructions.
001: VR                            1349842    1.80% |##
002: =TR                           141        0.00% |
003: BITS                          140        0.00% |
004: ×F                            54287      0.07% |
007: ×+F                           3200       0.00% |
011: OR                            162724     0.22% |
012: PERM                          339174     0.45% |
013: TOB                           4          0.00% |
015: NEV                           220963     0.29% |
016: ROUND                         996        0.00% |
017: DUMMY                         117798     0.16% |
020: ROUNDF                        640        0.00% |
024: FLOAT                         10301      0.01% |
025: FLOATD                        640        0.00% |
026: ABS                           228        0.00% |
027: NEG                           669831     0.89% |#
030: ABSF                          70         0.00% |
031: NEGF                          1302       0.00% |
033: NOT                           81859      0.11% |
034: ×D                            14918      0.02% |
035: ×                             5668       0.01% |
036: -                             942288     1.26% |#
041: ZERO                          357578     0.48% |
042: DUP                           4075828    5.44% |#####
043: DUPD                          444854     0.59% |#
044: DIVI                          63         0.00% |
045: FIX                           14443      0.02% |
047: STR                           1276       0.00% |
050: CONT                          16037      0.02% |
051: REVD                          46657      0.06% |
052: ERASE                         1592562    2.13% |##
054: AND                           2435234    3.25% |###
056: +                             2421985    3.23% |###
060: DIV                           17040      0.02% |
062: DIVF                          12055      0.02% |
065: REV                           2586523    3.45% |###
066: CAB                           525540     0.70% |#
067: FRB                           99         0.00% |
074: +F                            45208      0.06% |
075: -F                            48273      0.06% |
077: SIGNF                         5118       0.01% |
100: MkMq                          1216889    1.62% |##
101: =MkMq                         555        0.00% |
102: MkMqQ                         19153      0.03% |
103: =MkMqQ                        919        0.00% |
104: MkMqH                         34800      0.05% |
105: =MkMqH                        1          0.00% |
110: MkMqN                         1214714    1.62% |##
111: =MkMqN                        321        0.00% |
113: =MkMqQN                       791        0.00% |
115: =MkMqHN                       1          0.00% |
121: PARQq                         23         0.00% |
125: {PIB|PID}Qq                   23         0.00% |
140: M+Iq                          903910     1.21% |#
141: M-Iq                          190303     0.25% |
142: NCq                           2427858    3.24% |###
143: DCq                           1111869    1.48% |#
144: Iq=+1                         60709      0.08% |
145: Iq=-1                         22         0.00% |
146: Iq=+2                         60499      0.08% |
151: MqTOQk                        137796     0.18% |
152: IqTOQk                        736        0.00% |
153: IMqTOQk                       157        0.00% |
154: CqTOQk                        45468      0.06% |
155: CMqTOQk                       65         0.00% |
156: CIqTOQk                       153        0.00% |
157: QqTOQk                        1629       0.00% |
161: SHA                           88702      0.12% |
162: SHAD                          2732       0.00% |
164: SHL                           3901990    5.21% |#####
166: SHLD                          1315112    1.76% |##
167: SHC                           1197056    1.60% |##
```

```
170: =[R]{Q|C|I|M}q            3830292    5.11% |#####
171: {Q|C|I|M}q                3807638    5.08% |#####
172: =+{Q|C|I|M}q              2864012    3.82% |####
173: LINK                      1181086    1.58% |##
174: =LINK                     1295344    1.73% |##
177: JCqNZS                    1130       0.00% |
201: JrNE                      490689     0.66% |#
202: JrGEZ                     34643      0.05% |
204: JrLEZ                     114912     0.15% |
206: JrNEZ                     571088     0.76% |#
210: JrNV                      45523      0.06% |
211: OUT                       65         0.00% |
212: JrNEN                     1180184    1.58% |##
213: Jr                        3286319    4.39% |####
215: JSr                       2401324    3.21% |###
216: JrNTR                     163        0.00% |
217: EXIT                      2515582    3.36% |###
221: JrEQ                      162658     0.22% |
222: JrLTZ                     1882578    2.51% |###
224: JrGTZ                     1316461    1.76% |##
226: JrEQZ                     1617383    2.16% |##
230: JrV                       125134     0.17% |
232: JrEN                      855        0.00% |
234: JrEJ                      98         0.00% |
240: JrCqZ                     90496      0.12% |
260: JrCqNZ                    299249     0.40% |
300: EeMq                      4499638    6.01% |######
301: =EeMq                     1920821    2.56% |###
302: EeMqQ                     8998       0.01% |
303: =EeMqQ                    57313      0.08% |
304: SET                       6837815    9.13% |#########
```

At option, all tracing modes can compute a digital SIGNATURE of the execution: a 48-bit cumulative hash, displayed in octal, of the contents of all the relevant KDF9 registers (nest, SJNS and Q stores) at the end of each traced instruction. Known values for this hash can be used as a digital signature to verify the proper operation of an implementation of **ee9**. (When the signature is enabled, the time-of-day is forced to midnight, to produce a repeatable hash value.)

5: The Mode Settings Files and Miscellany Parameter

The emulator has default settings for all of its options, but they may be over-ridden by settings specified in files that the emulator attempts to read as part of its initialization, and/or by specifying a miscellany parameter on the command line.

5.1: Settings Files

The file `settings_1.txt` applies to a first or sole program to be run, and `settings_2.txt` applies to a second program overlaid by it (e.g. the Whetstone Controller, overlaid after a successful compilation by the Translator). A setting specified by the command line over-rides a similar option specified in the `settings_1.txt` file.

The settings files contain a line for each option to be set, beginning with a letter that specifies the option concerned. This may be followed by one or two parameters. Address parameters may be given either in octal—preceded by a hash sign ('#')—or in decimal; and this convention is also used systematically in output messages from the emulator. The options are presented here in upper case, but lower/mixed case is also accepted. The available options are as follows:

A LAX_MODE | STRICT_MODE | AUTHENTIC_TIME_MODE

The **A** flag sets aspects of the AUTHENTICITY of execution. It takes one symbolic parameter.
It is possible to set the strictness that **ee9** applies to checking for misused register operands, with a parameter that is either `LAX_MODE` or `STRICT_MODE`. In `STRICT_MODE` an operation with *n* operands always fails if the nest depth is less than *n*. In `LAX_MODE` such an operation fails if, and only if, it further reduces the nest depth. The latter more closely approximates the behaviour of the KDF9 nest hardware. This mode also affects instructions that attempt to change the value of Q store 0, which was hardwired to a constant 0 and ignored updates. In `LAX_MODE` an assignment to Q0 is suppressed, which is what the hardware did; in `STRICT_MODE` it is treated as an execution error (unless running in Director state), which is a diagnostic more likely to be useful.
It is also possible to set authentic elapsed timing (see §2.4), with the `AUTHENTIC_TIME_MODE` parameter.

B *start* [*end*]

This flag has either one or two parameters, which are instruction-word addresses. It sets a BREAKPOINT on every instruction word in the given range of addresses.

C *l h*

This flag is used to set two COUNT values, say *l* and *h*, that determine when tracing is done. No breakpoint or watchpoint fires, and no instruction is traced, unless $l \le i \le h$ is satisfied; where *i* is the current value of ICR. With suitable *l* and *h* values, tracing can confined to a set time during execution (for example, the last few instruction executions before a program fails). The values *l* and *h* are given as unsigned decimal integers.

D FAST_MODE | TRACE_MODE | PAUSE_MODE | EXTERNAL_MODE

This flag sets the DIAGNOSTIC mode, specifying the type of tracing and the kind of logging that may be generated.

F *start* [*end*]

This flag has either one or two parameters, which are data-word addresses. It sets a FETCHPOINT on every word in the given range of addresses.

G [*colour* [*tip size*]]

This flag allows one or two optional symbolic parameters. The first, if given, sets the GRAPH PLOTTING pen colour from the list: `BLACK` (the default), `BLUE`, `BROWN`, `CYAN`, `DARK_BLUE`, `DARK_CYAN`, `DARK_GREEN`, `DARK_GREY`, `DARK_MAGENTA`, `DARK_RED`, `GREEN`, `GREY`, `MAGENTA`, `RED`, `WHITE`, `YELLOW`. If a colour is given, a second parameter may be given to set the pen tip size from the list: `EXTRA_EXTRA_FINE` (the default, 1 plotter step wide), `EXTRA_FINE` (2 steps wide), `FINE` (4 steps), `MEDIUM` (6), `MEDIUM_BROAD` (8), `BROAD` (10), `EXTRA_BROAD` (12).
In any case, the shared buffer is switched from `TP1` to `GP0`.

I*i* *start end*

P*i* *start end*

These flags have two parameters, which are word addresses. They request that the contents of that range of addresses be output in a specified interpretation, INITIALLY, or POSTMORTEM (i.e. after the end of the run).
For both `I`*i* and `P`*i*, the interpretation is given by the string of letters *i*, each letter of which must be one of: **A**, for strings in ASCII/LATIN-1 code; **C**, for strings in card code; **L**, for strings in LINEPRINTER code; **N**, for strings in paper tape code, with case NORMAL shown; **S**, for strings in paper tape code, with case SHIFT shown; **T**, for strings in paper TAPE code, with both cases shown; **O**, for syllabic OCTAL/ ORDERS; **U**, for orders in pseudo-USERCODE format; and **W**, for data WORDS in octal, syllabic octal, line printer characters, Q store format, and signed decimal.
When **U** is specified, **D** can also be given to display machine code addresses in DECIMAL instead of octal. For an example of pseudo-Usercode format, see Appendix 4.
The `PIC`, `PID`, `POC` and `POD` instructions for cards and paper tape permit the processing of data in arbitrary character codes. The **A** format for core-store printing is provided to facilitate the debugging of modern KDF9 programs that process data in ASCII/Latin-1, the native character set of **ee9**.

L *t*

This flag is used to set a value, say *t*, that specifies an execution time LIMIT. This determines how long the KDF9 program is allowed to execute before being terminated. The limit is specified in instruction executions rather than seconds, so the program is terminated if `ICR` > *t* at the end of any instruction execution. The value *t* is given as an unsigned decimal integer.

N [*t*]

The **N** flag has one optional parameter, with the same meaning at the *t* parameter of the **L** flag. It makes **ee9** run in NON-INTERACTIVE mode, suitable for invocation from a command script. In this mode it is not possible to supply responses to prompts, whether from the KDF9 program or from **ee9** itself; so if an interactive input is requested in non-interactive mode, **ee9** terminates with a suitable diagnostic message. If the **N** flag is given without a parameter, or on the command line, the time limit is taken to be the default time limit for non-interactive mode (see §6).

R *a b*

This flag is used to set two addresses, say *a* and *b*, that delimit the RANGE of instructions where tracing is done. No breakpoint or watchpoint fires, and no instruction is traced, unless $a \le i \le b$ is satisfied; where *i* is the address of the word containing the instruction to be executed. With suitable *a* and *b* values, instruction tracing can confined to the sequence of instructions that you are currently debugging.

S *start* [*end*]

This flag has either one or two parameters, which are data-word addresses. It sets a STOREPOINT on every word in the given range of addresses.

T BOOT_MODE | PROGRAM_MODE | TEST_PROGRAM_MODE

This flag is used to set the TEST mode, specifying the kind of run.

V [–] {**ADEFHIPRSTZ**}

The **V** flag is used to set the VISIBILITY of diagnostic output, by stating the set of traces that **are** to be **suppressed**. It takes a parameter which optionally starts with '`–`', followed by a selection of the letters: '`A`' to suppress Director API messages, '`E`' to suppress confirmatory or warning messages, but not error messages, from **ee9**, '`F`' to suppress the FINAL STATE of the KDF9 at the end of a run, '`H`' for the HISTOGRAM, '`I`' for the INTERRUPT trace, '`P`' for the PERIPHERAL I/O trace, '`R`' for the RETRO trace, and '`S`' for the digital SIGNATURE. '`Z`' combines the effects of all the output-suppression options.
A trace is output if it is provided by the requested diagnostic mode, **and** its output is not suppressed.
The default is that all traces provided by the diagnostic mode are to be output, i.e. **not** suppressed.

The option 'D' can be given with the **V** flag, to **enable** the output of any optional DEBUGGING output.
The option 'T' can be given with the **V** flag, to **enable** execution with authentic TIMING (see §2.4).

W *start* [*end*]

This flag has either one or two parameters, which are data-word addresses. It sets a WATCHPOINT (i.e., a FETCHPOINT **and** a STOREPOINT) on every word in the given range of addresses.

5.2: THE MISCELLANY PARAMETER

The options permitted with the miscellany parameter are as follows: `adefghilnprstz123456789`. The letters `gln` correspond with settings file commands `G`, `L` and `N`, with the defaults stated above for their optional parameters. The letters `adefhiprstz` correspond with settings file visibility options. A digit *d* requests an execution time limit of *d*0_000_000 instructions. The miscellany parameter is scanned and put into effect from left to right.

6: IMPLEMENTATION CHARACTERISTICS AND CAVEATS

The defaults for the settable options in the present implementation of **ee9** are as follows:

- the default test mode is `PROGRAM_MODE`
- the default diagnostic mode is `FAST_MODE`
- the default diagnostic visibility generates all traces, the digital signature, and the histogram
- the default is to run interactively; that is, with non-interactive mode disabled
- the default register checking mode is `STRICT_MODE`
- the default elapsed time mode is **not** `AUTHENTIC_TIME_MODE`
- the default time limit allows for effectively unlimited execution
- the default time limit in non-interactive mode is 100 million instruction executions.
- the default count bounds, *l* and *h*, are `0` and the time limit, respectively
- the default range bounds, *a* and *b*, are `#0` and `#7777`, respectively
- no breakpoint, fetchpoint, storepoint, or core dump is pre-set
- the shared buffer is switched by default to `TP1`, not `GP0`
- the default graph plotter pen colour is `BLACK`
- the default graph plotter pen tip is `EXTRA_EXTRA_FINE` (1 plotter step wide)

The following features of KDF9 remain to be implemented:

- all I/O instructions for the `DR` and `ST` device types
- the `PIE`, `PIF`, `PIG`, `PIH`, `PMH`, `POG`, and `POH` instructions for the `FD` device type
- the `PMG`, `PMK`, `PML`, `POK`, and `POL` instructions (for all device types other than `CP`, which has `POK` and `POL`)
- Time Sharing Director OUTs other than OUTs 0 through 10, and OUT 17

KDF9's nest-depth checking caused a `NOUV` interrupt **after** the maximum or minimum depth had been transgressed. Presently, **ee9** checks for all of these violations **before** the offending instruction is executed. This makes little difference in practice. KDF9 had 'imprecise' interrupts, which made recovery from a `NOUV` error impossible: Director could do no more than terminate (or perhaps restart) the offending program. (See also the `A` option setting, §5.)

There is some doubt as to the semantics of the various division instructions, particularly with respect to rounding, and their behaviour on overflow and on division by zero (other than setting the overflow bit).

All of the I/O instructions that apply to EE model 1081 magnetic tape decks (the most common kind) have been implemented, with the important restriction that data blocks are limited to at most 512 words (4K bytes) in length. I hope to lift this restriction in a future release.

There is considerable doubt as to the correct instruction encoding, and precise effects, of the `PMG`, `PMH`, `PMK`, `PML`, `POK`, and `POL` orders, which are listed in the KDF9 Programming Manual but not well defined therein, nor in any other source presently known.

It is assumed that the `POF` order for the `TP` device type has exactly the same functionality as the `POE` order.

It is assumed that the `POC` and `POD` orders for the Flexowriter change from writing to reading after the output of any word that has the KDF9 paper tape code for a semicolon (34_8) in its least significant six bits.

It is assumed that the `POB` and `POD` orders for the graph plotter have the same effect as `POA` and `POC`, respectively, as the Manual says that the plotter did not respond properly to an End Message character.

It is assumed that the device type code for the graph plotter to be used with OUT 5 is octal 20, i.e. 16.

It is assumed that the graph plotter pen tip sizes are the same as those of pens currently on sale.

It is assumed that the fixed-head area of the `FD` device type is platter 0, seek area 0.

Many other hypotheses have been put into effect in the implementation of the `FD` device; it remains to be seen whether these are justified.

ACKNOWLEDGEMENTS

I am grateful to the group of supporters, all enthusiastic former KDF9 engineers, programmers, or satisfied users, for their encouragement during this project; and for their superb work in recreating a software ecosystem for **ee9** to run. I thank in particular David Hawley and Brian Randell, for their crucial caches of EE documents; David Holdsworth for his Usercode compilers and hardware insight; David Holdsworth, Brian Wichmann, Graham Toal, and Roderick McLeod for resurrecting the Whetstone Algol system; and David Holdsworth, Mike Hore, and Bill Gallagher for compiling and testing **ee9** ports. Others, too numerous to list, know who they are: to them also, my thanks.

The plotter command file `wabbit_kdf9.txt`, used to create the example plot in §3.5, I code-converted to KDF9 plotting code from an ICL 1900 Series plotter test file made available by Bill Gallagher.

REFERENCES

Available at: `http://www.findlayw.plus.com/KDF9`

The English Electric KDF9; W. Findlay, 2011.

The Hardware of the KDF9; W. Findlay, 2010.

The Software of the KDF9; W. Findlay, (in preparation).

KDF9 Programming Manual, Publication 1002 mm (R); International Computers Ltd., 2nd Edition, October 1969.

KDF9 ALGOL programming, Publication 1002 mm (R) 1000565; J.S. Green, English Electric-Leo-Marconi Computers Ltd.

See also: `http://www.findlayw.plus.com/KDF9/#Emulator` which is updated periodically with **ee9** news; and the `README` and `HOWTO` files, included in the distribution of **ee9**.

APPENDIX 1: USING ee9 MORE CONVENIENTLY

To reduce the typing required to invoke **ee9** correctly, I provide a set of `bash` command files, namely **nine_test**, **nine**, **whet** and **tsd**. These are kept in the `Testing` directory of the **ee9** distribution, along with a number of auxiliary command files. Usercode source programs and their data files are kept in `Testing/Assembly`; Algol source programs and data files are kept in `Testing/Algol`; and compiled Usercode programs are kept in `Testing/Binary`. To facilitate the assembly of Usercode programs, I also supply the **ucc** command. This provides a convenient harness for **kal3**, David Holdsworth's new Usercode compiler; it takes the source program from `Assembly` and places the object program in `Binary`. All of these programs expect to be executed from the `Testing` directory. Using these commands on Microsoft Windows requires a `bash`-compatible shell, such as the one included in Cygwin.

ucc *prog*

This command compiles a Usercode source program using the **kal3** assembler. The source code is taken from `Assembly/`*prog*`.k3` and the object program is placed in `Binary/`*prog*`.kdf9`, while a compilation listing is stored in `Assembly/`*prog*`-listing.txt`.

EXAMPLE

• To compile `Assembly/HiGuys.k3`:

```
./ucc HiGuys
```

nine *prog* {*data* | – } [{*mode* | – } [*miscellany*]]

This command runs a binary KDF9 problem program, as previously compiled using **ucc** or **kal3**. The program is taken from `Binary/`*prog*`.kdf9`; the data file for `TR0`, if required, is taken from `Assembly/`*data*`.txt`. The *mode* and *miscellany* parameters are as described for *f* and *m* in §1 (but note the different format); the defaults are as for **ee9** itself. If fast mode is explicitly requested, **nine** measures the time taken by the execution of the program.

EXAMPLE

To run `Binary/Leech.kdf9` in default mode; but with all logging output suppressed, taking its `TR0` data from `Assembly/Leech_data9.txt`, in non-interactive mode so that reading past the end of data forces termination, and with the shared buffer (pointlessly) switched to `GP0`:

```
./nine  Leech  Leech_data9  -  gnz
```

Result:

```
TP0:
===
AEFDBC
|
A
B
C
|
CCC
FACA
DFDC
DBDB
FBCEE
BBBBBBB
AEABAEAB
AEEAEABBB
ADADADADAD
ABCDEABCDEABCDEABCDEABCDEABCDEABCDE
|
  1045|
===
```

nine_test *prog* {*data* | – } [{*mode* | – } [*miscellany*]]

This command operates exactly like the **nine** command, except that the program is executed in **test_program** mode.

whet *prog* [{*mode* | – } [*miscellany*]]

This command runs the Whetstone Algol system on the Algol 60 program and its 'stream 10' input data, both held in the file `Algol/`*prog*`.a60` and read in turn by the Translator and the Controller. The *mode* and *miscellany* parameters are as described for **nine**.

Example

To compile and run the historical Whetstone Benchmark, `Algol/Whetstone.a60`, in the (timed) fast mode:

```
./whet Whetstone f
```

Result:

```
Welcome to ee9 V2.0q, the GNU Ada KDF9 emulator.
Running a problem program in fast mode (without diagnostics).
________________________________________________________________
OUT;N.|

ee9: OUT 5: requests a device of type #02; gets TR1.
WHETSTONEBMK|
RAN/EL/000M05S/000M15S  SIZE    603
ee9: OUT 8: closes stream #10.
ee9: OUT 6: releases TR1.

ee9: OUT 1: ICR = 813081; RAN/EL = 4816503 / 20242121 KDF9 us.
ee9: OUT 1: the Whetstone Controller overlays the Translator.
Running a problem program in fast mode (without diagnostics).
________________________________________________________________
STREAM;30.|
AD     - S
ee9: OUT 8: closes stream #30.

AD  30 CLOSED
RAN/EL/006M56S/006M57S
ee9: OUT 1: ee9 will not return to the Whetstone Translator.
________________________________________________________________
Final State:

At #03016/1 (1550/1); ICR = 74907395; the instruction was #200:220:000, i.e. OUT
CIA:          #03016/1 (1550/1)
NIA:          #03016/4 (1550/4)
ORDERS:       74907395 executed (ICR)
CPU TIME:    420605422 KDF9 us. (RAN)
CLOCK TIME:  443794712 KDF9 us. (EL)

The SJNS is empty.

Q store:
 Q1: Q #064773/ #000002/ #001125  = Q  27131/      2/    597
 Q2: Q #000040/ #000015/ #001124  = Q     32/     13/    596
 Q3: Q #000001/ #001124/ #011443  = Q      1/    596/   4899
 Q4: Q #000040/ #000001/ #001133  = Q     32/      1/    603
 Q6: Q #000000/ #007217/ #007223  = Q      0/   3727/   3731
 Q7: Q #000000/ #000000/ #000024  = Q      0/      0/     20
 Q8: Q #000030/ #177777/ #007203  = Q     24/     -1/   3715
 Q9: Q #000171/ #077640/ #077600  = Q    121/  32672/  32640
Q10: Q #000000/ #000301/ #007203  = Q      0/    193/   3715
Q11: Q #000036/ #000000/ #000000  = Q     30/      0/      0
Q12: Q #000030/ #177777/ #077700  = Q     24/     -1/  32704
Q13: Q #000037/ #077700/ #077705  = Q     31/  32704/  32709
Q14: Q #000000/ #000001/ #007202  = Q      0/      1/   3714
Q15: Q #000003/ #000002/ #007203  = Q      3/      2/   3715

The NEST is empty.
________________________________________________________________
End of Run.
FW0 on buffer #00 typed 121 bytes.
TR1 on buffer #02 read 5416 bytes.
TP0 on buffer #04 punched 552 bytes.
LP0 on buffer #05 printed 13 lines.
________________________________________________________________

real    0m1.743s
user    0m1.723s
sys     0m0.005s
```

```
TP0:
===

LINE   18 REL LINE   8 POSITION IDENTIFIER lab
END COMMENT
LINE   24 REL LINE   5 POSITION IDENTIFIER p0
END COMMENT
LINE   32 REL LINE   7 POSITION IDENTIFIER p3
END COMMENT
LINE   46 REL LINE  13 POSITION IDENTIFIER pout
END COMMENT
IDENTIFIER a NOT USED
DECLARED ON LINE   2
IDENTIFIER b NOT USED
DECLARED ON LINE   2
IDENTIFIER c NOT USED
DECLARED ON LINE   2
RAN/EL/000M05S/000M15S  SIZE    603
===

LP0:
===

N=   0 J=   0 K=   0 X1=+1.00000000 X2=-1.00000000 X3=-1.00000000 X4=-1.00000000
N= 120 J= 140 K= 120 X1=-0.06834220 X2=-0.46263766 X3=-0.72971839 X4=-1.12397907
N= 140 J= 120 K= 120 X1=-0.05533645 X2=-0.44743656 X3=-0.71097339 X4=-1.10309806
N=3450 J=   1 K=   1 X1=+1.00000000 X2=-1.00000000 X3=-1.00000000 X4=-1.00000000
N=2100 J=   1 K=   2 X1=+6.00000000 X2=+6.00000000 X3=-0.71097339 X4=-1.10309806
N= 320 J=   1 K=   2 X1=+0.49040732 X2=+0.49040732 X3=+0.49039250 X4=+0.49039250
N=8990 J=   1 K=   2 X1=+1.00000000 X2=+1.00000000 X3=+0.99993750 X4=+0.99993750
N=6160 J=   1 K=   2 X1=+3.00000000 X2=+2.00000000 X3=+3.00000000 X4=-1.10309806
N=   0 J=   2 K=   3 X1=+1.00000000 X2=-1.00000000 X3=-1.00000000 X4=-1.00000000
N= 930 J=   2 K=   3 X1=+0.83466552 X2=+0.83466552 X3=+0.83466552 X4=+0.83466552

RAN/EL/006M56S/006M57S
===
```

tsd [{*mode* | - } [*miscellany*]]

tsd runs the Time Sharing Director (the original KDF9 operating system from English Electric) in the specified manner. The *mode* and *miscellany* parameters are as described for **nine**. This is how it boots, with the supplied `FW0` file:

```
Welcome to ee9 V2.0q, the GNU Ada KDF9 emulator.
Performing a cold boot in fast mode (without diagnostics).
________________________________________________________________________

P
KKT40E007UPU
TIME SHARING DIRECTOR  2464 WDS|
02U01
02U02
05U03
01U04
03U05
10U07
10U10
10U11
10U12
10U13
10U14
CORE MODULES;8.|

OUT 8 REEL NO;9.|

A-PROGRAM DETAILS|
LEVELS;N.|

DATE  D/M/Y;4/5/67.|

TIME ON 24-HOUR CLOCK
HOURS/MINS;1/2.|^C
ee9: Breakpoint: (f:ast | t:race | p:ause or q:uit)?

TINT;G0.|
```

```
10L14 /Iden<_J_U_N_K>,TSN -00-2339
10L13 /Iden<_J_U_N_K>,TSN 77777777
10L12 /Iden<EFPBEAAG>,TSN -00-0552
10L11 /Iden<WHETLIST>,TSN -00-1498
10L10 /Iden<_Z_E_R_O>,TSN -00-1478
10L07 /Iden<PRINTEND>,TSN 0-00-929
02U01
02U02
05U03
01U04
03U05
10L07 PRINTEND
10L10 _Z_E_R_O
10L11 WHETLIST
10L12 EFPBEAAG
10L13 _J_U_N_K
10L14 _J_U_N_K^C
ee9: Breakpoint: (f:ast | t:race | p:ause or q:uit)? q
Run stopped by user!
________________________________________________________________________
Final State:

At #00074/2 (60/2); ICR = 13666251474; the instruction was #042, i.e. DUP
CIA:          #00074/2 (60/2)
NIA:          #00074/3 (60/3)
ORDERS:     13666251474 executed (ICR)
CPU TIME:   70696387070 KDF9 us. (RAN)
CLOCK TIME: 70748738953 KDF9 us. (EL)

The CPU is in DIRECTOR_STATE
CONTEXT:  0
PRIORITY: 0
BA:       #000000
NOL:      #077777
CPDAR:    AAAAAAAAAAAAAAAA
RFIR (Interrupt Flags):
PR:       FALSE
FLEX:     FALSE
LIV:      FALSE
NOUV:     FALSE
EDT:      FALSE
OUT:      FALSE
LOV:      FALSE
RESET:    FALSE

The SJNS is empty.

Q store:
 Q5: Q #000003/ #177073/ #136511  = Q      3/   -453/ -17079

V is set. T is clear.
NEST:
 N1:
#4040400000000000 = -138504105361408 = -4.40810381558E-38
     = Q #101010/ #000000/ #000000   = Q -32248/      0/      0
     = #202 #010 #000 #000 #000 #000 = "ØØØ      "
 N2:
#0000000000000000 =                0 = 0.00000000000E+0
     = Q #000000/ #000000/ #000000   = Q      0/      0/      0
     = #000 #000 #000 #000 #000 #000 = "         "
________________________________________________________________________
End of Run.
FW0 on buffer #00 typed 595 bytes.
MT0 on buffer #07 is at BOT, after 2 inter-block gaps and 48 bytes.
MT1 on buffer #10 is at BOT, after 2 inter-block gaps and 48 bytes.
MT2 on buffer #11 is at BOT, after 2 inter-block gaps and 48 bytes.
MT3 on buffer #12 is at BOT, after 2 inter-block gaps and 48 bytes.
MT4 on buffer #13 is at BOT, after 2 inter-block gaps and 48 bytes.
MT5 on buffer #14 is at BOT, after 2 inter-block gaps and 48 bytes.
```

Appendix 2: KDF9 Characters and their Latin-1 Transcriptions

Line printer	SP		LF	FF	HT		%	'
Card Reader	SP	ISO:"	LF	FF	HT	ISO:#	%	'
Normal Case	SP	ISO:"	LF	FF	HT	ISO:#	CS ISO:ß	CN ISO:ñ
Shift Case	SP	ISO:"	LF	FF	HT	ISO:#	CS ISO:ß	CN ISO:ñ
Octal code	00	01	02	03	04	05	06	07

Line printer	:	=	(	)	£	*	,	/
Card Reader	:	=	(	)	£	*	,	/
Normal Case	ISO:&	ISO:?	ISO:!	ISO:%	ISO:'	ISO:$	ISO:~	/
Shift Case	ISO:&	ISO:?	ISO:!	ISO:%	ISO:'	ISO:$	ISO:~	:
Octal code	10	11	12	13	14	15	16	17

Line printer	0	1	2	3	4	5	6	7
Card Reader	0	1	2	3	4	5	6	7
Normal Case	0	1	2	3	4	5	6	7
Shift Case	↑ ISO:^	[	]	<	>	=	×	÷
Octal code	20	21	22	23	24	25	26	27

Line printer	8	9		$_{10}$ ISO:º	;	+	-	.
Card Reader	8	9	_ ISO: _	$_{10}$ ISO:º	;	+	-	.
Normal Case	8	9	_ ISO: _	$_{10}$ ISO:º	;	+	-	.
Shift Case	(	)	_ ISO: _	£	;	≠ ISO:±	*	,
Octal code	30	31	32	33	34	35	36	37

Line printer		A	B	C	D	E	F	G
Card Reader	ISO:@	A	B	C	D	E	F	G
Normal Case	ISO:@	A	B	C	D	E	F	G
Shift Case	ISO:@	a	b	c	d	e	f	g
Octal code	40	41	42	43	44	45	46	47

Line printer	H	I	J	K	L	M	N	O
Card Reader	H	I	J	K	L	M	N	O
Normal Case	H	I	J	K	L	M	N	O
Shift Case	h	i	j	k	l	m	n	o
Octal code	50	51	52	53	54	55	56	57

Line printer	P	Q	R	S	T	U	V	W
Card Reader	P	Q	R	S	T	U	V	W
Normal Case	P	Q	R	S	T	U	V	W
Shift Case	p	q	r	s	t	u	v	w
Octal code	60	61	62	63	64	65	66	67

Line printer	X	Y	Z					
Card Reader	X	Y	Z	ISO:{	ISO:}	→ ISO:|	ISO:\	see note
Normal Case	X	Y	Z	ISO:{	ISO:}	→ ISO:|	ISO:\	see note
Shift Case	x	y	z	ISO:{	ISO:}	→ ISO:|	ISO:\	see note
Octal code	70	71	72	73	74	75	76	77

NOTES

The transcription provides a Latin-1 representation for every KDF9 internal character code.

- SP is a blank space; LF is Line Feed; FF is Form Feed; HT is Horizontal Tab; CS is Case Shift; CN is Case Normal.
- '*y* ISO:*x*' indicates that *x* is the ISO Latin-1 transcription of the non-Latin-1 KDF9 character *y*.
- 'ISO:*x*' indicates that *x* is the ISO Latin-1 external representation of a non-legible KDF9 character.
- Fast devices (e.g. magnetic tapes) always use the Normal Case representation.
- Code 77_8 is represented by Ø; on two-shift devices (such as the Flexowriter) for 'character mode' transfers only, and on punched card devices and fast devices invariably.
- If a cell is empty, that code is completely suppressed by the line printer.
- Except for 'character mode' output, ß and ñ are acted upon by the Flexowriter and paper tape punch, not transferred literally, so that output is presented in the correct case.

APPENDIX 3: KDF9 GRAPH PLOTTER CODES

Plotting action	none	step paper back	step paper forward	step pen right	step pen left	step pen right, paper back	step pen left, paper forward	lower pen	raise pen
Normal Case	SP	ISO:"	LF	HT	ISO:&	ISO:?	ISO:!	0	ISO:@
Octal code	00	01	02	04	10	11	12	20	40

All other 6-bit character codes represent invalid plotter commands.

APPENDIX 4: DISASSEMBLED MACHINE CODE

The **U** format core printing routine traces the program's control flow, and uses simple heuristics, to determine whether a given word represents data or instructions. These do a good job, but cannot always be correct. Words thought to be data are output in a variety of formats. Instructions are shown with octal or decimal operand addresses, at option. An instruction that is the target of a jump starts a new line labelled by its address. An address is also given for the instruction sequentially following a subroutine jump (`JS`…), as that is the link value stored in the SJNS, and may be useful for following the course of execution.

Here is an example for which the heuristics work well. Lines of the form `V`*n*`=`*value*`;` are local static data declarations; and the executable code begins with '`V14; =V13;`':

```
P51V15;
         V0=F0.019042127887;
         V1=F0.019042129240;
         V2=F0.038082414120;
         V3=F0.076666493927;
         V4=F0.121226383896;
         V5=F0.725940450930;
         V11=Q6/1/0;
         V12=Q4/1/0;
         V14=F1.0;
         V15=F0.5;

         V14; =V13;
         DUP; DUP; ×F; V14; +F; JSP40; =V6;
         V12; =Q13;

2;       V13; V6M13; +F; V15; ×F; =V13;
         V13; V6M13Q; ×F; JSP40; =V6M13; J2C13NZ;
         V11; =Q13;
         V0M13Q; ZERO; REV; FIX; FLOATD;

1;       V0M13; V5M13Q; ×+F; J1C13NZ;
         ROUNDF; ÷F; EXIT1;
```

And here is its **DU**-format output:

```
Core store interpreted as instructions.

1393/0:     #1742337743622052 =   68337217250346 = 1.90421278869E-2
 =          #076 #046 #377 #217 #044 #052 = "/BºØCR0J"
 =            Q   15910/ #177617/  #22052 = Q 15910/  -113/  9258

1394/0:     #1742337743651250 =   68337217262248 = 1.90421292400E-2
 =          #076 #046 #377 #217 #122 #250 = "/BºØCU(H"
 =            Q   15910/ #177617/  #51250 = Q 15910/  -113/ 21160

1395/0:     #1744677611614626 =   68504697379222 = 3.80824141200E-2
 =          #076 #115 #376 #047 #031 #226 = "/DWØ=QF6"
 =            Q   15949/ #177047/  #14626 = Q 15949/  -473/  6550
...
1409/0:
                E1407; =E1406;
                DUP; DUP; ×F; E1407; +F;
            JSE1263/0;
1411/4:
                =E1399;
                E1405; =Q13;
1413/0:
                E1406; E1399M13; +F; E1408; ×F; =E1406;
                E1406; E1399M13Q; ×F;
            JSE1263/0;
1417/0:
                =E1399M13;
            JE1413/0C13NZ;
                E1404; =Q13; E1393M13Q; ZERO; REV; FIX; FLOATD;
1420/0:
                E1393M13; E1398M13Q; ×+F;
            JE1420/0C13NZ;
                ROUNDF; DIVF;
            EXIT 1;
```