ERSATZ-11 DEMONSTRATION VERSION 2.1
PDP-11 SYSTEM SIMULATOR
FOR 30-DAY COMMERCIAL EVALUATION ONLY

Release date: 01-Apr-1999

Ersatz-11 is a full system emulator of the PDP-11. This demo version runs on any AT-class computer with an 80186 or later CPU with DOS V2.0 or later, and requires approximately 400–600 KB of free memory (depending on peripheral configuration and emulated memory size). operating system. It has been tested with RT-11 (all flavors), RSX-11M, RSX-11M-PLUS, RSTS/E, IAS, TSX-Plus, 2.9BSD and 2.11BSD UNIX, DSM-11, DOS/BATCH, Fuzzball (BOS), and XXDP+.

Emulated configuration:

- PDP-11/23, PDP-11/24, PDP-11/34a, PDP-11/40, PDP-11/44, PDP-11/45, PDP-11/53, PDP-11/70, PDP-11/73, PDP-11/83, PDP-11/84, PDP-11/93, or PDP-11/94 CPU with individually selectable features
- FP11/FPF11/FPJ11 (etc.) floating point processor
- 248 KB–400 KB main memory (approx.)
- UDA50/RQDX3,RAxx/RDxx/RXxx MSCP disks; number limited by available memory
- KLESI/TQK50/TQK70,TU81/TKxx TMSCP tapes; number limited by available memory
- RK11D,RK02/RK05 disk drives; up to 8 per controller
- RL11/RLV12,RL01–02 disk drives; up to 4 per controller
- RK611,RK06–07 disk drives; up to 8 per controller
- RX211,RX02, RX11,RX01, or RXT11,RX01 dual 8” floppies
- RP11C,RP02–03 pack drives; up to 8 per controller
- RC11,RS64 fixed-head disk drives; up to 4 per controller
- RF11,RS11 fixed-head disk drives; up to 8 per controller
- RH11/RH70,RS03–04 fixed-head disk drives; up to 8 per Massbus
- DL11 terminal ports; console and up to 15 others (VT100, and/or uses COM, LPT, Sound Blaster MIDI ports)
- LP11 printer ports; up to 4 (same devices as DL11)
- PC11 paper tape reader/punch (using files)
- TA11,TU60 dual cassette tape drive
- TC11,TU56 DECtape drives; up to 8 per controller
- TM11,TU10 magtape drives; up to 8 per controller
- TS11/TSU05/TSV05/M7454/TQK25,TS04/TS05/TU80/TK25 magtape drives; normally one per controller
- RH11/RH70,TM03,TE16/TU45/TU77 magtape drives; up to 64 per Massbus
- DELUA Ethernet ports; up to 4 (using PC Ethernet cards)
- KW11L line clock (50/60 Hz, settable)
- Display register (using special hardware)

PC hardware support:

- Block devices: DOS disk image files, raw floppy drives
- Tape devices: DOS tape image files
- Character devices: Emulated VT100 on SVGA (can flip between up to 12 virtual screens), COM, LPT, Sound Blaster MIDI ports
- Network devices: NE2000 clones, packet drivers

This is a stripped-down demonstration version of Ersatz-11, which when used for commercial purposes may only be installed for an evaluation period limited to 30 days; after this time, commercial users must either buy E11 (either the "Lite" or full version), or delete all copies of the demo version in their possession. *There is no limitation on hobby/personal use of this demo package.* Commercial use is defined as anything having to do with the operation of a for-profit business. Previous versions of Ersatz-11 (V1.1A and earlier) had no such limitation on use, so this notice does not apply to them, however they are no longer supported by D Bit.

This demo version of Ersatz-11 is available by anonymous FTP from `FTP.DBIT.COM` (199.181.141.53), which was formerly known as `TATS.WIZVAX.NET`. The directory is `pub/e11`, there's a `README` file that lists which files are ASCII and which are binary.

The emulator speed depends on the application, but in general E11 on any speed Pentium II runs very significantly above the speed of a PDP-11/93. In real mode versions of E11, writing MMU registers is an expensive operation which slows down multiuser OSes, compared to RT-11FB for example, but this is far less of a factor in the full version of E11 (which has an entirely different MMU implementation due to running in protected mode). Meanwhile, E11's disk I/O is much faster than that of real PDP-11s.

The FP11 emulation currently requires a math coprocessor. If the PC has none, then the emulated PDP-11 will have no FPP either. Intel Pentium CPUs that have the floating point divider bug are detected and a workaround is used to get correct results at a slight speed penalty (for `DIVF`/`DIVD` only).

The system has been tested under the XXDP+ diagnostic monitor. It passes the KD11EA diagnostics DFKAA, DFKAB, and DFKAC, and the FP11A diagnostics DFFPA, DFFPB, and DFFPC. It does not work with MMU diagnostics due to the absence of the maintenance mode.

INSTALLATION

Ersatz-11 installation is straightforward. Simply mount the distribution disk on a floppy drive and type:

`C\>`*x*`:install`

where *x* is the name of the floppy drive. The installation program will prompt for a directory into which to install, and will then extract all the E11 files from the distribution disk. The files are as follows:

file	*contents*
`E11.EXE`	executable
`E11.HLP`	"HELP" data file
`E11.PDF`	this document, readable with Adobe Acrobat
`HD*.*`	source and binaries for RT-11 HD: driver
`DO*.*`	source and binaries for RT-11 DO: driver
`KED.CMD`	keyscripts for using cursor keys with KED/EDT

E11's home directory is also a sensible place to put disk image files (*.DSK) and the E11.INI initialization file (see below), since E11 will look there if these files aren't found in the current working directory.

INITIALIZATION FILE

When E11 is first started, it looks for a file named "`E11.INI`," using E11's usual search rules (see the section on filenames). If this file exists then a command is read from the file each time E11 would otherwise prompt for a user command; input for the console DL11 still comes from the keyboard, so the init file may be used both to start up the system and to shut it down (E11 will continue after the `BOOT` command if you type Alt-SysReq or Shift-Enter). If a line's first nonblank character is "`;`" or "`!`", it is treated as a comment. Processing of the init file can be disabled by starting E11 with the "`/NOINIT`" switch, and a different file may be specified using the "`/INITFILE:`*file*" switch; the default extension is "`.INI`" and the usual search rules apply.

Normally, the initialization file contains "`ASSIGN`" commands to define all the emulated character devices (including network interfaces), "`MOUNT`" commands to define all the emulated block devices (disks/tapes), a "`SET CPU`" command to define the emulated CPU model if the default PDP-11/34a is not desired, and a "`BOOT`" command to start up the PDP-11 operating system.

Typical `E11.INI` file:

```
mount dl0: rt11.dsk
; uncomment the following commands to use a COM port for console I/O
;set com3: irq5
;assign tt0: com3: /mode:1200,n,8,1
set cpu 44
boot/rt11 dl0:
; control returns to the next line when the user presses Alt-SysReq
quit
```

There is also a "`/MEMORY:`*nnn*" switch that may be specified on the DOS command line when E11 is started. It sets the maximum possible emulated memory size of the PDP-11 to *nnn* (decimal) kilobytes. By default this maximum is 384 KB. If this amount is not available, E11 settles for whatever it can get (rounded down to a multiple of 8 KB) as long as it's at least 248 KB. The reason this switch exists is so that you can enlarge PDP-11 memory past the default in real mode versions, (how much depends on what device drivers and TSRs you have loaded), or reduce it if memory is so tight that `ASSIGN`, `LOG`, or `MOUNT` commands fail for lack of it (they will give error messages if this is the case).

DEVICE NAMES

With the exception of DOS files, just about every I/O device used by Ersatz-11 has a device name ending in a colon ("`:`"). This applies to both emulated PDP-11 devices and real PC hardware devices. Each device (disk unit, serial line, etc.) has a name that normally conforms to the following prototype:

dev[*c*][*u*]`:`

dev Alphabetic device name identifying the device type: always two letters for emulated PDP-11 devices, variable for PC hardware devices.

c Optional letter (A–Z) identifying which of the (potentially) multiple controllers of the same type is controlling this particular device; only used with controllers that can support multiple devices, a reasonable default is used if it is omitted. The controller letter is always displayed in output from the SHOW command for devices where it is meaningful.

u Optional unit number identifying the device; default is the first unit on the controller. The unit number should be omitted when referring to the controller as a whole (e.g. SET commands).

The two-letter device names for emulated PDP-11 devices are taken from the names used by the popular PDP-11 operating systems. Where possible, synonyms are available to ensure that the device names will be familiar to users of each operating system; for example, DELUA ports may be referred to using either the RSTS/RSX name ("`XE:`"), or the RT-11 name ("`NU:`"), and similarly, RK05 disk units may be called "`DK:`" or "`RK:`". However E11 uses a more consistant naming system than these operating systems do, which in some cases leads to differences, for example "`TT`*u*`:`" refers specifically to a DL11/DLV11 serial port, a serial port located on a DZ11/DZV11 is always "`YZ`*cu*`:`". As a result, the device names used by E11's command language may not necessarily be identical to those used by the operating system for the same devices.

The device names for PC hardware are the same as those used by DOS for those devices that actually have names in DOS. So `CON:` refers to the first video session, `COM1:`–`COM4:` are serial ports (`AUX:` is a synonym for `COM1:`), and `LPT1:`–`LPT4:` are parallel ports (`PRN:` is a synonym for `LPT1:`). For other devices a short mnemonic name is used, with an optional letter identifying the controller for devices like multi-port serial interfaces where a port number alone isn't enough to uniquely identify the device. Note that PDP-11 unit numbers always start at 0 (TT0:, DU0:, MU0: etc.) while PC unit numbers generally start at 1 (COM1, LPT1).

For a very few devices (both real and emulated), identifying the controller and unit isn't enough since there may be multiple slaves attached to the same master unit. In this case the device name is expanded to look like "*dev*[*c*][*u*][_*s*]`:`", where *s* is the optional slave number within unit *u*. This form is rarely used since it only makes sense on emulated Massbus tape drives with multiple slaves attached to the same formatter, or on SCSI devices with multiple LUNs within the same target such as the old SCSI/MFM bridge boards. In any case if *s* is omitted (as well as the preceding underscore) a reasonable default is used.

FILENAMES

A few rules apply to DOS files referenced using E11 commands. When a reference is made to an existing file, without a drive name or directory path in the file specification, E11 searches for it first in the current directory, then in the directory where `E11.EXE` is located (under DOS V3.0 and later), then in the directories listed in the `PATH` environment variable. A filename may be enclosed in single (') or double (") quotes to allow lower case or special characters in the name, however this is only useful when running on a system that supports the "long filenames" extended DOS calls. Path elements must be separated by "\" characters, except when the file specification is enclosed in single or double quotes in which case either forward slashes or backslashes may be used. Each command that takes a filename has a default extension which is added when none is supplied by the user.

Typical defaults are:

.ext	*type of file*
`.CMD`	command file, used by @ command
`.DSK`	disk image file
`.INI`	init file, used by /INITFILE switch
`.LOG`	log file
`.PAP`	paper tape image file
`.PDP`	binary memory image
`.TAP`	tape image file

COMMANDS

Ersatz-11 recognizes many keyboard commands. These are entered at the "`E11>`" prompt, which appears whenever the PDP-11 is halted (e.g. at startup) but may be brought up at any time by pressing Shift-Enter or Alt-SysReq (or by pressing the BREAK key on a serial terminal if the console terminal (TT0:) has been `ASSIGN`ed to a COM port). Some versions of the Windows DOS box intercept Alt-SysReq, but Shift-Enter seems to work everywhere. Commands (and parameters and switches) may generally be shortened to any unique abbreviation. Note that E11 is multithreaded and PDP-11 code continues to be executed while you are entering commands at the prompt, if you haven't `HALT`ed it.

`@`*file*[`.CMD`]

Accepts input from the specified file as if it had been typed at the E11 prompt. The default extension is "`.CMD`", and search rules are the same as for the `E11.INI` initialization file. Lines read from the file are not echoed, and indirect files may not be nested.

`ASSIGN` *pdp11dev*: *pcdev*: [*pcswitches*]

Assigns a physical PC device to emulate the specified PDP-11 character device. Where appropriate, the PDP-11 multiplexer of the appropriate type is created if it did not already exist. The PDP-11 device name consists of two letters that give the controller type, an optional third letter explicitly specifying the multiplexer in cases where there are multiple multiplexers of that type, and a decimal unit number. For example "`YZC3:`" refers to line 3 on the third DZ11/DZV11 serial multiplexer. If the multiplexer letter is omitted, the default controller (the first or only one) of that type is assumed. Only single-line ports are present in the demo version of E11, so no letter is needed.

Supported PDP-11 character devices:

name	*units*	*controller*	*notes*
`TT`*u*:	0–15	DL11/DLV11	`KB`*u*: and `YL`*u*: are synonyms
`LP`*u*:	0–3	LP11/LPV11	

Supported PC character devices:

name	*units*	*type*	*switches*
`CON`*u*:	1–12	video console	(none)
`COM`*u*:	1–4	PC COM port	`/FIFO`[`:`*n*], `/NOFIFO`, serial switches
`LPT`*u*:	1–4	PC LPT port	
`SBMIDI:`	none	Sound Blaster MIDI port	

`CON`*u*: The specified PDP-11 port is connected to a simulated VT100 that can normally be put up on the screen by pressing Alt and the function key corresponding to the screen number (F1–F12). Note that the screens assigned to F11 and F12 are not accessible if you have the old 84-key AT keyboard, unless you redefine other keys to reach them. When one screen is being displayed on the PC screen, the others (up to 11) are maintained invisibly in memory, so they will be up to date when you switch the display to them by pressing Alt and the function key for the screen you want. Note that this is only the default behavior of the function keys, if you redefine them then it's up to you to define keys to switch displays (using the "`PRIMARY` *n*" and "`SECONDARY` *n*" keyboard commands).

If there are two video adapters on the PC (e.g., an SVGA and a Hercules monochrome card), then one pseudo VT100 session may be displayed on each. Using the default key scripts, the Alt-function keys choose which of the 12 possible screens is displayed on the primary monitor, and the Ctrl-function keys choose which is on the secondary monitor (unless these keys have been redefined). Note that it is not possible to display the same port on both monitors at once; if this is attempted then whichever monitor was previously showing the port, switches to displaying the lowest-numbered available screen not already being displayed.

`COM`*u*: The PDP-11 port is connected to the specified PC COM port. Any changes to the default I/O port address and IRQ settings must be made with a `SET COM`*n*`: PORT=`*nnnn* `IRQ`*n* command before assigning a PDP-11 port to the COM port. The baud rate, number of data bits, etc. for a COM port may be set with a `/MODE` switch, see "serial switches" below for a description. These parameters may also be changed using a SET command once the port is assigned to a PDP-11 device.

The "`/NOFIFO`," "`/FIFO`," and "`/FIFO:`*n*" switches control usage of the receive FIFOs on the 16550A (etc.) UART chips used in almost all current PC COM ports. These FIFOs greatly reduce interrupt traffic and enabling them normally increases the maximum throughput of the system, however they can make input appear "bursty," especially at low baud rates. "`/NOFIFO`" disables the receive FIFOs, "`/FIFO`" enables them, and "`/FIFO:`*n*" enables them only when the baud rate is at least *n*. The setting is "`/FIFO:4800`" by default for ports that have FIFOs, ports that don't are always set to "`/NOFIFO`" regardless of the switch given.

`LPT`*u*: The PDP-11 port is connected to the specified PC LPT port. Any changes to the default I/O port address and IRQ settings must be made with a `SET LPT`*n*`: PORT=`*nnnn* `IRQ`*n* command before assigning a PDP-11 port to the LPT port. Some LPT ports do not work well with interrupts — for example, if you have multiple LPT ports they may all try to drive IRQ7 at once, or it's possible that a printer may not generate the `ACK` signal correctly. If yours is one of these then use the `SET LPT`*n*`: NOIRQ` command before assigning the port. This tells E11 to use timers and polled I/O for printer output, which works well with most late model printers, but some printers may experience very poor performance with this (printing only about 18 characters per second), if this happens it is best to resolve the IRQ problem and use interrupts.

Printers may also be accessed using the PC BIOS, by issuing a `SET LPT`*n*`: BIOS` command before assigning the port. This is mainly useful for remote printing using network software that intercepts BIOS output, for a locally attached printer it's better to use the default `NOBIOS` setting.

LPT output is passed transparently, so you'll need to make sure that your OS and printer agree on whether lines end in <`CRLF`> or just <`LF`>, and on whose responsibility it is to expand tabs and form feeds.

`SBMIDI:` The PDP-11 port is connected to the Sound Blaster MIDI port. This allows connecting electronic musical instruments (synthesizers, etc.) to the sound board using a special cable. The default I/O port address and IRQ settings are taken from the BLASTER environment variable, 330/IRQ5 are used if this variable is not defined. Any changes to these defaults must be made with a `SET SBMIDI: PORT=`*nnnn* `IRQ`*n* command before assigning a PDP-11 port to the MIDI port. The communications parameters are not changeable on a MIDI port.

The Sound Blaster MIDI port supports interrupt-driven I/O only when receiving characters, so E11 must use polled I/O to transmit. This means that PDP-11 emulation may stall for up to one character time while the port driver is waiting to start a new character transmission, so the SBMIDI: port should not be used on systems where there are other real-time tasks running. This has nothing to do with emulated interrupts provided for PDP-11 software, those work in both directions, but transmitting a character can stop all PDP-11 instruction execution for several hundred microseconds.

Serial switches

Serial switches may be used with COM ports, or any other type of serial port:

`/LOCK` Locks communication mode parameters (baud rate, data bits etc.) against being changed by the PDP-11 port. This is useful if you want to set a port with programmable parameters (e.g. a DZ11 line) to a non-standard speed without having the PDP-11 OS set it back to a lower speed.

`/MODE:`*string* Sets communication mode parameters to the values specified in the *string*, which has the form "*bps*`,`*par*`,`*dbits*`,`*sbits*". *Bps* is the number of bits per second (range of allowable values depends on the specific serial hardware). *Par* is the parity specified as one letter: Even, Odd, None, Mark, Space (not all serial hardware supports all types). *Dbits* is the number of data bits, generally 5–8 although some port types can only do 7–8. *Sbits* is the number of stop bits, 1–2 (on most ports, "2" actually means 1.5 if *dbits* is 5).

`/TXMAX:`*n* This switch sets the maximum number of transmitted characters that will be buffered for transmission by that port. Once the port has accepted this number of characters from the PDP-11, it waits for all of them to be transmitted before accepting more characters. This value should be set low enough to get acceptable response to XON/XOFF characters (it may take up to *n* characters for the PDP-11 to react and suspend output), but high enough to get adequate throughput. The default value is 16.

`/UNLOCK` The opposite of the `LOCK` switch, unlocks the port's `MODE` parameters so that they may be altered by the PDP-11 port (on port types where this is possible such as the DZ11 and DHU11). This is the default so there's normally no reason to use it as an ASSIGN switch, however it may be useful as a SET keyword once the port has been assigned.

Note that E11 does not flag an error if you `ASSIGN` a `TT:` port to a printer, or `ASSIGN` an `LP:` port to a screen, even though these are not likely to be useful combinations. The reason both port types use the same pool of devices is so that they can both access COM ports, since serial terminals and serial printers are both reasonable devices. `LP:` ports attached to COM ports or video screens respond to `XON/XOFF` flow control. Actually, one good reason to `ASSIGN` an `LP:` port to a screen is that you can then issue a `LOG` command to capture the output to a file, without necessarily having to watch the output go by on the screen.

The `ASSIGN` command fails if the specified PC port doesn't exist, or if the device's currently `SET` IRQ is already in use and not `/SHARE`ed, or if you're trying to steal `TT0:`'s device for some other port (there must always be something attached to `TT0:` since that's E11's console terminal).

`ASSIGN XE`*n*`:` *pcdev*`:` [`/`*switches*] [$proto_1$ $proto_2$ $proto_3$ …]

Assigns a PC network device to emulate the specified DELUA (XE:) Ethernet port. *Pcdev* may be one of the following:

name	*type*	*switches*
`NE2000`*c*`:`	NE2000 clone	(none)
`PKTD:`	Packet Driver	`/INT:`*vv*

The "`NE2000`*c*`:`" device refers to a Novell NE2000 (or clone) Ethernet board, accessed using E11's own internal driver. This typically requires less software overhead than using a DOS packet driver, and common PCI boards are supported as well as the older ISA ones. The "*c*" parameter is an optional letter specifying which NE2000 board to use (default is "A", meaning the first or only one). The port/IRQ values and controller type should be set with "SET NE2000c: …" commands prior to issuing the ASSIGN command. By default the controller type is "AUTO", which means that E11 should first try to locate a PCI board and failing that, should assume that there's an ISA board located at the specified port/IRQ values. Setting the controller type to "PCI" tells E11 that it's definitely a PCI board, so the port/IRQ values are ignored (the values obtained from the PCI BIOS are used instead), and an error message is given if no PCI board exists. A controller type of "ISA" tells E11 to skip the PCI probe and definitely use the supplied port/IRQ values.

The "PKTD:" device uses a DOS Packet Driver to access the physical Ethernet board. If specified, "`/INT:`*vv*" gives the hexadecimal 80x86 interrupt vector to which the packet driver is attached. If the interrupt vector is not given, then the range of vectors from 20 to FF (hex) is searched until a packet driver is found that isn't already in use by another simulated Ethernet port. A list of hex 16-bit *DIX* protocol numbers, up to ten total, may be specified, in which case E11 asks the packet driver to pass only frames of those types. This may allow E11 to coexist with other protocol stacks running on the same PC, as long as they each use different sets of protocol numbers and neither one needs to change the port's physical address after the other is loaded, and in any case it may be helpful for shielding the PDP-11 operating system from unwanted broadcast traffic.

DECnet requires the ability to change the address (it wants the physical address to be based on the protocol address to avoid the additional overhead of performing address resolution over the network, as TCP/IP does), so in general running DECnet under E11 will require either that no other network software be running on the PC, or that you install a second Ethernet board for E11's exclusive use. This is because the packet driver refuses to change the physical address once the board is already in use so as not to surprise protocol stacks that were already running when the change was made and thought they knew the address.

Freeware packet drivers for a wide variety of Ethernet interfaces are available via anonymous FTP from many sources including `FTP.FTP.COM`, and are typically included on the driver disk that comes with the adapter. Ersatz-11 requires packet drivers that conform to version 1.09 or later of the packet driver specification as published by FTP Software.

`BOOT` *ddu*`:` [`/`*switches*]

Boots the system from the specified disk, magnetic tape, DECtape, or paper tape. The disk/tape must have been mounted with the `MOUNT` command. The optional switch is an OS name; for now the only meaningful ones are `/RT11` and `/RSTS`. `/RSX` is accepted too but has no special effect. This has to do with the method used to pass time and date information to a newly booted monitor. RT-11 ignores the time and date passed at 005000 unless the NOP in word 000000 of the bootstrap is cleared to 0 (HALT) and the bootstrap is entered at 000002. RSTS uses the time and date at 001000 (in a different format from RT-11) regardless of whether its NOP was cleared, but later versions of RSTS save the first word of the bootstrap and execute it later, so they will halt if the system was booted the RT-11 way. Hence the need for the switch. Note that the OS switches are meaningful only on block devices. If you like typing the time and date manually (or your PC has no RTC) then don't worry about the switch. RSX doesn't appear to have a way to pass the time and date to a fresh monitor, so you'll have to use "F12" or else write a privileged program to read the TOY clock (recent RSX-11M-PLUS versions have a built in `TIM /SYN` command to do this).

There is also a `/HALT` switch, which means to go as far as loading block 0 into core and setting up the registers, but to stop there. This can be handy for debugging boot blocks.

The `BOOT PR:` command expects a tape image in absolute binary format, as produced by the "`LINK /LDA`" command under RT-11.

`CALCULATE` *expr*
`&` *expr* (synonym)

Calculates the value of a 32-bit octal expression and displays the result in octal, decimal, hex, ASCII and radix-50. The operators are `* / + -`, unary `+ - ^C` (where "ˆC" means logical complement), and `( )`, with the usual precedence. Numbers are either octal digit strings, or decimal if they contain `8` or `9` or end in "`.`", or hex if preceded by "`^X`", or radix-50 triplets if preceded by "`^R`"; or general register contents may be specified using the names `R0`–`R5` (with a "`'`" suffix to indicate the other register set, when emulating a PDP-11 with dual register sets) or `SP` or `PC`, `R$` or `PS` means the processor status word, and something of the form "`'a`" means the ASCII value of the character "`a`".

`DEFINE KEYPRESS` *keyname* `=` *statement*
`DEFINE KEYRELEASE` *keyname* `=` *statement*

Defines the action taken when the specified key is pressed or released. Keyboard operation is defined using a simple script language, which allows the user to bind a small script to any possible keypress or keyrelease, which is executed whenever that key is pressed (`DEFINE KEYPRESS`) or released (`DEFINE KEYRELEASE`). When E11 is first started, the keyboard is initialized with a set of scripts which define the action of a VT100-like keyboard with a US English layout. Just like user definitions, these default scripts may be displayed with the `SHOW KEYPRESS` and `SHOW KEYRELEASE` commands; by default most keys do nothing on release and have no `KEYRELEASE` definition, except for the Alt, Ctrl, and Shift keys. Using the "`E11.INI`" initialization file, the user may easily redefine some or all of the keyboard as required.

Keyboard script language

The keyscript language is very simple and is somewhat similar to BASIC. When defining a new keyscript, it may be helpful to use a `SHOW` command to display the existing default keyscript for that or a similar key, and use it as the basis for the new keyscript. To make the keyboard send a string, simply enclose that string in single or double quotes. `IF` statements can be used to send different strings depending on the state of a variety of flags (including the current state of the Alt/Ctrl/Shift keys). `NONREPEATING` and `NOREPEATS` are used in many of the default keyscripts to precisely emulate the autorepeat behavior of the VT100 keyboard; `NONREPEATING` tells E11 that this keyscript should not be autorepeated even if the key is held down, and `NOREPEATS` means that while this key is pressed, all other keys should be prevented from autorepeating (which is what the VT100 Ctrl key does). There are numerous other special-purpose commands that allow such things as sending the current date and/or time in a variety of formats, and popping up the E11 command prompt.

Multiple statements in a single key definition may be separated by ":" or "\" characters and count as one statement (for the purposes of the `IF/ELSEIF/ELSE/ENDIF` construct). If a line ends with "&" (with no white space following) it is continued on the next line, and any characters after the first "!" that is not inside single or double quotes are considered a comment and are ignored (up until the trailing "&" if one is present). This should be familiar to

BASIC-PLUS users.

Keyboard script statement descriptions

string

Sends the specified string. The string may be any combination of double quoted strings (``*string*''), single quoted strings (`*string*'), and single ASCII characters (`CHR$(`*n*`)`), concatenated with plus signs (`+`). Note that PDP-11 serial ports normally have only one or two characters worth of input buffering, and E11 currently buffers 32 characters per port in addition to that (this number may increase in the future), so it is not possible to send arbitrarily long strings.

`AMPM`

Sends "`AM`" or "`PM`" depending on whether the time read by `GETTIME` is before or after noon.

`CLEAR` *flag*

Clears a read/write flag.

`DAY1`

Sends the 1- or 2-digit day of the month (1–31) as read by `GETTIME`.

`DAY2`

Sends the 2-digit day of the month (01–31) as read by `GETTIME`.

`GETTIME`

Reads the current date and time (as an atomic operation to avoid race conditions) and stores it internally for use by `HOUR2`/`MINUTE2`/`SECOND2`) etc. Without a preceding `GETTIME` statement, the statements that send the individual parts of the date/time will send garbage.

`HOUR1`

Sends the 1- or 2-digit hour of the day (0–23) as read by `GETTIME`.

`HOUR12`

Sends the 1- or 2-digit hour (1–12) as read by `GETTIME`.

`HOUR2`

Sends the 2-digit hour of the day (00–23) as read by `GETTIME`.

`HUNDREDTH2`

Sends the 2-digit hundredth of a second (00–99) as read by `GETTIME`.

`IF <expr1> THEN`

```
 [statement1]
ELSEIF <expr2> THEN
 [statement2]
 ...
ELSE
 [statement3]
ENDIF
```

Executes statements conditionally. The expressions may be made up of any combination of read-only and read/write flags (see below), the operators `AND`, `NOT`, `OR`, and `XOR`, and parentheses (to override the default binary operator precedence, which is `NOT`, `AND`, and `OR`/`XOR` from highest to lowest with `OR` and `XOR` being equal). If the expression after the `IF` is true, the statement (which may be multiple statements separated by ":" or "\" characters) following the `THEN` is executed, and execution then skips to after the `ENDIF`. Otherwise the expression following the `ELSEIF` (if any) is similarly tested, followed by any successive `ELSEIF`s if the first fails. Finally the `ELSE` clause (if any) is executed if no (`ELSE`)`IF` was true.

The `ELSEIF` keyword is provided as a convenience to avoid excessive nesting:

```
IF a THEN
 x
ELSEIF b THEN
 y
ELSE
 z
ENDIF
```

is equivalent to:

```
IF a THEN
 x
ELSE
 IF b THEN
  y
 ELSE
  z
 ENDIF
ENDIF
```

`LETTER` *string1*

Acts as a normal alphabetic ("letter") key. *String1* is a one-character string; if `CTRL` is true, *string1* is sent with the high 3 bits set to 0. Otherwise if `CAPS` or `SHIFT` is true then *string1* is sent with bit 5 set to 0. Otherwise *string1* is sent with bit 5 set to 1.

`MINUTE2`

Sends the 2-digit minute (00–59) as read by `GETTIME`.

`MONTH1`

Sends the 1- or 2-digit month (1–12) as read by `GETTIME`.

`MONTH2`

Sends the 2-digit month (01–12) as read by `GETTIME`.

`MONTH3`

Sends the 3-letter English month abbreviation (`Jan`–`Dec`) as read by `GETTIME`.

`NONDATA`

Specifies that the current key is not a data key and should not generate keyclick (if E11 supports it in the future) or count from a "`SET` *flag* `FOR` *n*" prefix.

`NONREPEATING`

Specifies that the current key should not auto-repeat.

`NOREPEATS`

Specifies that the current key should prevent all other keys from auto-repeating until it is released.

`NUMBER` *string2*

Acts as a normal numeric ("number") key. *String2* is a two-character string; if `CTRL` is true, nothing is sent. Otherwise if `SHIFT` is true then the second character of *string2* is sent. Otherwise the first character of *string2* is sent.

`PRESS` *keyname*

Executes the "keypress" script for the specified key.

`PRIMARY` *n*

Changes the screen on the primary video display to screen *n* (1–12).

`PROMPT`

Pops up an E11 command prompt.

`RELEASE` *keyname*

Executes the "keyrelease" script for the specified key.

`RSTSAMPM`

Sends "`AM`" or "`M`" or "`PM`" depending on the time read by `GETTIME`, using the unusual rules used by pre-V9.0 versions of RSTS/E:
00:00–00:00:59.99 is PM (the minute starting at midnight)
00:01–11:59:59.99 is AM as usual

12:00–12:00:59.99 is M (the minute starting at noon)
12:01–23:59:59.99 is PM as usual

`SECOND2`

Sends the 2-digit second (00–59) as read by `GETTIME`.

`SECONDARY` *n*

Changes the screen on the secondary video display to screen *n* (1–12).

`SET` *flag* [`FOR` *n*]

Sets a read/write flag. If the "`FOR` *n*" modifier is given, it means that the flag is set for the specified non-zero number of data keystrokes, and then automatically clears after the script for the *n*th keystroke is executed. This is used for the prefix keys common on non-English keyboards, and can also be useful for handicapped users. Note that the current key counts from the total unless a `NONDATA` statement is part of its definition.

Example using `SET` to redefine the right-hand `Alt` key as an "acute accent" prefix key, which makes the vowel keys send the ISO Latin-1 codes for the same letters with acute accents when pressed immediately afterwards:

```
DEFINE KEYPRESS RALT = SET FLAG1 FOR 1 : NONDATA
DEFINE KEYPRESS A = IF FLAG1 THEN LETTER CHR$(193) ELSE LETTER ‘A’ ENDIF
DEFINE KEYPRESS E = IF FLAG1 THEN LETTER CHR$(201) ELSE LETTER ‘E’ ENDIF
DEFINE KEYPRESS I = IF FLAG1 THEN LETTER CHR$(205) ELSE LETTER ‘I’ ENDIF
DEFINE KEYPRESS O = IF FLAG1 THEN LETTER CHR$(211) ELSE LETTER ‘O’ ENDIF
DEFINE KEYPRESS U = IF FLAG1 THEN LETTER CHR$(218) ELSE LETTER ‘U’ ENDIF
```

`TOGGLE` *flag*

Toggles a read/write flag.

`UCMONTH3`

Sends the 3-letter upper case English month abbreviation (`JAN–DEC`) as read by `GETTIME`.

`YEAR2`

Sends the 2-digit year (00–99) as read by `GETTIME`.

`YEAR4`

Sends the 4-digit year (1980–2099) as read by `GETTIME`.

FLAGS

The keyboard script language has a number of boolean flags, which may be used in key scripts and `DEFINE LED` commands. They are broken into two groups: read/write flags, and read-only flags.

Read/write flags
Can be used in `IF` expressions or `DEFINE LED` commands, or set using `SET`, `CLEAR`, and `TOGGLE` statements. These

descriptions are only defaults used by E11's initial keyboard definitions, the user is free to redefine them at will.

`CAPS`	Caps Lock state
`FLAG`n	User-defined flags (n=1–4), reserved for user key scripts
`LALT`	Left (or only) Alt key state
`LCTRL`	Left (or only) Ctrl key state
`LSHIFT`	Left Shift key state
`NUM`	Num lock state
`RALT`	Right Alt key state
`RCTRL`	Right Ctrl key state
`RSHIFT`	Right Shift key state
`SCROLL`	Scroll lock state

Read-only flags
Can be used in `IF` expressions or `DEFINE LED` commands only, values are maintained by E11 itself.

`ALT`	OR of `LALT` and `RALT`
`APPKEYPAD`	Applications keypad mode (ESC =)
`CHARSETA`	Char set A (UK) is currently selected
`CHARSETB`	Char set B (US) is currently selected
`CHARSET0`	Char set 0 (graphics) is currently selected
`CHARSET1`	Char set 1 (undefined) is currently selected
`CTRL`	OR of `LCTRL` and `RCTRL`
`CURSORKEY`	Cursor key mode (ESC [?1h)
`EKB`	True: 101-key Enhanced (or 104-key W95) keyboard, false: 84-key AT keyboard
`G0`	G0 character set is selected (SI)
`G1`	G1 character set is selected (SO)
`L1`	keyboard LED 1 is lit (ESC [1q)
`L2`	keyboard LED 2 is lit (ESC [2q)
`L3`	keyboard LED 3 is lit (ESC [3q)
`L4`	keyboard LED 4 is lit (ESC [4q)
`NEWLINE`	Newline mode (ESC [20h)
`SHIFT`	OR of `LSHIFT` and `RSHIFT`
`VT52`	VT52 mode (ESC [?2l)

Key names, used for `DEFINE KEYPRESS`, `DEFINE KEYRELEASE` commands, and `PRESS`, `RELEASE` statements. Key descriptions are for US English keyboards and may differ on keyboards designed for other languages, for most of these keyboards the physical layout is close to the US layout so name of the key that would be in the same position on a US keyboard should be used in script language definitions. Keys labeled "EKB only" exist only on the 101-key "Enhanced" keyboard and the 104-key "Windows 95" keyboard. It is not an error to bind key scripts to them even when only an 84-key AT keyboard is present, but scripts for keys that are missing will never be executed.

`'`	`'/"`
`*`	`*` key on keypad, or `*/PrScr` key
`,`	`,/<`
`-`	`-/_`
`.`	`./>`
`/`	`//?`

`0–9`	Numeric keys (top row of main keyboard)
`;`	`;/:`
`=`	`=/+`
`A–Z`	Alphabetic keys
`BACKSPACE`	Backspace key (top right of main keyboard)
`CAPSLOCK`	Caps Lock key
`CONTEXT`	Context Menu key (104-key keyboard only)
`DARROW`	Down arrow key (EKB only)
`DEL`	Del (EKB only)
`END`	End (EKB only)
`ENTER`	Enter
`ESC`	Esc
`F1–F12`	Function keys (`F11`, `F12` on EKB only)
`HOME`	Home (EKB only)
`INS`	Ins (EKB only)
`KP0–KP9`	Numeric keys on keypad
`KPENTER`	Enter key on keypad (EKB only)
`KPMINUS`	`-` key on keypad
`KPPERIOD`	`.` key on keypad
`KPPLUS`	`+` key on keypad
`KPSLASH`	`/` key on keypad (EKB only)
`LALT`	Left (or only) Alt key
`LARROW`	Left arrow key (EKB only)
`LCTRL`	Left (or only) Ctrl key
`LSHIFT`	Left Shift key
`LWIND`	Left "Windows" key (104-key keyboard only)
`NUMLOCK`	Num Lock key
`PAUSE`	Pause key (EKB only); N.B. most keyboards send the "release" code for this key immediately after the "press" code, rather than waiting until the user actually releases the key
`PGDN`	PgDn (EKB only)
`PGUP`	PgUp (EKB only)
`PRSCR`	Print Screen key (EKB only)
`RALT`	Right Alt key (EKB only)
`RARROW`	Right arrow key (EKB only)
`RCTRL`	Right Ctrl key (EKB only)
`RSHIFT`	Right Shift key
`RWIND`	Right "Windows" key (104-key keyboard only)
`SCRLOCK`	Scroll Lock key
`SPACE`	Space bar
`SYSREQ`	Sys Req (84-key keyboard only)
`TAB`	Tab
`UARROW`	Up arrow key (EKB only)
`[`	`[`/{
\	\/|
`]`	`]`/}
`‘`	`‘`/~

In addition to the above, the following keywords define keys that don't exist on most keyboards, for completeness:

`KEY00`	Sends scan code 00 hex
`KEY55`	Sends scan code 55 hex
`KEY56`	Sends scan code 56 hex (unmarked key on some keyboards made by Focus)
`KEY59`	Sends scan code 59 hex
`KEY5A`	Sends scan code 5A hex
`KEY5E`	Sends scan code 5E hex
`KEY5F`	Sends scan code 5F hex

`DEFINE LED` *ledname* `=` *flag*

Defines which flag is tracked by each keyboard LED. LED names are `CAPS`, `NUM`, and `SCROLL`. The flag may be the name of any flag (read-only or read/write) from the keyboard script language, in which case the LED turns on when the flag is set and turns off when the flag is clear; or it may be `NONE` to turn the LED off permanently.

`DEASSIGN` *ddu*:

Disables the specified character or network device. Deassigning `TT0:` is not allowed (either explicitly, or implicitly by `ASSIGN`ing its PC device to another PDP-11 device).

`DEPOSIT` [*/switches*] *addr* val_1 val_2 ...

Deposits the word(s) val_1, val_2 etc. starting at memory address "*addr*," which is forced even. An error message is returned if an attempt is made to access a nonexistent CSR in the I/O page (bus timeout). The address space to use is specified by the switch(es), or the space used in the last `EXAMINE` or `DEPOSIT` command is used by default if none are given. See the `EXAMINE` command for a list of valid switches.

`DISMOUNT` *ddcu*:

Dismounts the specified mass storage device (see `MOUNT`).

`DUMP` [*/switches*] *filename*[`.PDP`] [s_1`:`e_1 s_2`:`e_2 ... s_n[`:`[e_n]]]

With no switches, dumps PDP-11 memory to the specified DOS file (default extension is "`.PDP`"). Any number of address ranges "s_i`:`e_i" may be given, and data will be dumped to the file from each range inthe order given in the command line. The last range may have no ending address, in which case file data are dumped until the end of memory. If no ranges are given at all the default is to dump all of PDP-11 memory starting at 000000.

With either the "`/ROM`" or "`/EEPROM`" switch, dumps a range of ROM or EEPROM to the file. The ROM/EEPROM must have been created with "`LOAD/ROM`" or "`LOAD/EPROM`". Only one address range may be specified; it must begin at the beginning of the ROM but may end before the end of the ROM. The "`/BANKED`" switch can dump all of a banked ROM if only the starting address is given (rather than dumping only up to the first 512. bytes), in this case the starting address can be omitted too, (17)773000 is the default for "`/ROM`" and (17)765000 is the default for "`/EEPROM`". If no ending address is given, the default is to dump out the whole ROM.

ROM/EEPROM page and loads its contents from the file. The ROM contains a linear copy of the file contents, unless the "`/BANKED`" switch is also given, in which case it is banked through a 512.-byte window at either (17)773000 or (17)765000, using the page control register (see the "`PCR`" option under `SET CPU`). Only one address range may be given. If "`/BANKED`" switch is specified, the address range must be exactly 512. bytes long and must begin at one of the addresses given above; if "`/EEPROM`" is specified, the starting address must be (17)765000, so this address will be used by default, otherwise (17)773000 is the default. Otherwise, if only the starting address is given, the size of the ROM depends on the size of the file. If an ending address is given, the file must be large enough to fill that address range.

`EXAMINE` [*/switches*] [*addr* [*end*]]

Examines the word at memory address *addr*, which is forced even. If *end* is specified then a range of words is displayed. If both are missing then the 8 words following the last location accessed with `EXAMINE` or `DEPOSIT` are displayed. An error message is returned if an attempt is made to access a nonexistent CSR in the I/O page (bus timeout). The address space to use is specified by the switch(es), or the space used in the last `EXAMINE` or `DEPOSIT` command is used by default if none are given.

Switches:

switch	*space*
`/CURRENT`	Current CPU mode, specified by $PSW_{15:14}$
`/PREVIOUS`	Previous CPU mode, specified by $PSW_{13:12}$
`/KERNEL`	Kernel mode
`/SUPERVISOR`	Supervisor mode
`/USER`	User mode
`/INSTRUCTIONS`	I space (within one of the above modes)
`/DATA`	D space (within one of the above modes)
`/PHYSICAL`	Physical 22-bit address space (default if MMU disabled)

Note that the address space switch(es), if any, must be given before the address expression on the command line, to avoid ambiguity since the switch character ("`/`") is used for division in expressions.

`FPREGISTER` [r v_1 v_2 [v_3 v_4]]

Sets or displays the FPP registers. r is the FP accumulator number, `0`–`5`, and v_1–v_4 are two or four 16-bit octal words to write in the register (sorry, not decimal). If no arguments are given then the octal contents of all six ACs are given, along with octal displays of the `FPS`, `FEC`, and `FEA`, and also a bit-by-bit display of `FPS`.

`GO` [*addr*]

Starts the machine at the specified address, or at the address currently in the program counter if none is given.

`HALT`

If the machine is running, halts it and displays the registers. Otherwise a no op.

`HELP` [*command*]

Explains use of Ersatz-11 commands. Just type "`HELP`" for a list.

`INITIALIZE`

Initializes all emulated I/O devices, disables the MMU, sets the CPU mode to "kernel."

`LIST` [*/switches*] [*addr*]

Disassembles eight instructions starting at the specified address if it is given, or otherwise at the first address following the last one disassembled by the most recent `LIST` or `REGISTER` command. The address space to use is specified by the switch(es), or the space from the last `LIST` command is used if none are given. The default for `LIST` is set to `/CURRENT /INSTRUCTIONS` after each register dump, either from a `REGISTER` command or from the register display from a `STEP` command or CPU halt. See the `EXAMINE` command for a list of valid switches.

`LOAD` [*/switches*] *filename*[`.PDP`] [s_1`:`e_1 s_2`:`e_2 … s_n[`:`[e_n]]]

With no switches, loads the specified DOS file into PDP-11 memory (default extension is "`.PDP`"). Any number of address ranges "s_i:e_i" may be given, and data from the file will be loaded into each range in the order given in the command line. The last range may have no ending address, in which case file data are loaded until end of file is reached. If no ranges are given at all the default is to load the file at 000000. This command may be useful with binary files produced by Strobe Data Inc.'s `PDPXASM` cross-assembler.

With either the "`/ROM`" or "`/EEPROM`" switch, creates a ROM/EEPROM page and loads its contents from the file. The ROM contains a linear copy of the file contents, unless the "`/BANKED`" switch is also given, in which case it is banked through a 512.-byte window at either (17)773000 or (17)765000, using the page control register (see the "`PCR`" option under `SET CPU`). Only one address range may be given. If "`/BANKED`" switch is specified, the address range must be exactly 512. bytes long and must begin at one of the addresses given above; if "`/EEPROM`" is specified, the starting address must be (17)765000, so this address will be used by default, otherwise (17)773000 is the default. Otherwise, if only the starting address is given, the size of the ROM depends on the size of the file. If an ending address is given, the file must be large enough to fill that address range.

`LOG` *ddcu*: [*filename*] [`/APPEND`]
(where *dd* is `LP` or `TT`.)

Logs all output to the specified character device in the specified file. If no filename is specified, any existing log file for that device is closed. The `/APPEND` switch means to append to an existing log file, rather than replacing it. The default extension is "`.LOG`".

`LOG` *ddcu*: [*filename*] [`/APPEND`]
(where *dd* is `CT`, `DC`, `DF`, `DK`, `DL`, `DM`, `DP`, `DS`, `DT`, `DU`, `DX`, `DY`, `PD`, `MM`, `MS`, `MT`, `MU`, `DO`, or `HD`.)

Logs commands sent to the TA11, RC11, RF11, RK11D, RL11, RK611, RP11C, RS03/04, TC11, MSCP, RX11, RX211, RXT11, TM03, TS11, TM11, or TMSCP controller, or the DOS file device or `HD_SYS.EXE` pseudo-controller, to the specified file. If no file is specified, the current log file, if any, is closed. The unit number is insignificant (except for Massbus devices, which effectively have separate controllers for each unit), all commands to the controller are logged regardless of the currently selected unit. The `/APPEND` switch means to append to an existing log file, rather than replacing it.

`LOG XE`*n*: [*filename*] [*switches*]

Controls logging of Ethernet events. If a filename is specified then the log file is opened. If switches are specified they specify what events are to be logged; "`/[NO]COMMANDS`" controls logging of port commands, "`/[NO]RECEIVE`" controls logging of received frames, and "`/[NO]TRANSMIT`" controls logging of transmitted frames. The switches may be specified when the log file is first opened, or later in `LOG` commands with the filename parameter missing to change what is being logged without having to open a new log file. If neither the filename nor any switches are specified, any existing log file for that device is closed. If no switches are specified when the file is first opened, the default is "`/RECEIVE /TRANSMIT`". In addition, the `/APPEND` switch means to append to an existing log file, rather than replacing it.

`MOUNT` *pdp11dev*: [*pdp11switches*] *pcdev* [*pdp11switches and/or pcswitches*]

Mounts a PC file or device as the specified PDP-11 block device. The PDP-11 disk/tape controller of the appropriate type is created if it did not already exist. The PDP-11 device name consists of two letters that give the controller type, an optional third letter explicitly specifying the controller in cases where there are multiple controllers of that type, and a decimal unit number. For example "`DTB6:`" refers to transport 6 on the second TC11 DECtape controller. If the controller letter is omitted, the default controller (generally the first or only one) of that type is assumed. In the demo version of E11, the RH11/RH70 Massbus controllers and TC11 DECtape controllers are the only types of which there may be more than one.

Switches specific to the controller type may appear either after the PDP-11 device name or after the PC device (or file) name, and are typically used to specify the drive type in case the controller supports more than one drive type. If no drive type switch is specified, the default type is usually based on the size of the PC device. All emulated controller types support the "`/RO`[`NLY`]" (syn. "`/WP`[`ROTECT`]") switch, which has the same pressing the `WRITE PROT` (etc.) button on a real drive, and works even in cases like the RX01 where the real hardware had no write protection facility. A "`/RW`" switch exists for completeness and allows read/write access to the device, which is the default behavior.

Supported PDP-11 disk (or disk-like) devices:

name	*units*	*controller*	*drive/volume switches*	*notes*
`DC:`	0–3	RC11	(none, always RS64)	
`DF:`	0–7	RF11	(none, always RS11)	(syn. `RF:`)
`DK:`	0–7	RK11D	`/RK02`, `/RK05` (syn. `/RK03`)	(syn. `RK:`)
`DL:`	0–3	RL11	`/RL01`, `/RL02`	
`DM:`	0–7	RK611	`/RK06`, `/RK07`	
`DP:`	0–7	RP11C	`/RP02`, `/RP03`	
`DS:`	0–7	RH11/RH70	`/RS03`, `/RS04`	
`DT:`	0–7	TC11	(none, always TU56 DECtape)	
`DU:`	0–65535	MSCP	`/TYPE:`*xxxyy*	
`DX:`	0–1	RX11	(none, always RX01)	
`DY:`	0–1	RX211	`/RX01`, `/RX02`, `/SS`, `/DS`	
`PD:`	0–1	RXT11	(none, always RX01)	
`HD:`	0–15	virtual	(none, file size is all that matters)	

For MSCP disks, the `/TYPE:`*xxxyy* switch sets the unit type name, which may be up to three letters and up to two decimal digits. The default is RA81 for image files, or the actual drive type (RX50 etc.) for floppies.

Supported PC virtual disk devices:

Disk image files

`MOUNT` *ddcu*: *d:path\filename*[`.DSK`] [*switches*]

A disk image file contains a byte-by-byte image of a PDP-11 disk, presumably loaded from a real PDP-11 using Kermit or DECnet or some equivalent, or built using `FLX.EXE` or `PUTR.COM` or `RT11.EXE` or a similar utility. There are two types of image files, "block" and "sector" images. "Block" images contain the disk data as it would be read in sequential block order, which for most PDP-11 disks is the same as the raw device order anyway. This is the most common format and is normally used by default. "Sector" images apply to floppy disks only; RX01, RX02, and RX50 disks are organized using a soft interleave layout to increase their speed when used with controllers that have only one sector buffer. The PDP-11 device handlers (and/or controllers) for these disks handle the soft interleave so it is normally invisible to the PDP-11 user program, so images made of these disks using something like the RT-11 "`COPY/FILE/DEV`" command will be normal block images. When a block image file is accessed as a virtual PDP-11 floppy disk, Ersatz-11 does the inverse of the soft interleave so that when the PDP-11 driver does the interleave, the blocks come out in the correct order.

However if the image is taken using special software or on a non-DEC computer, it may be more natural for the image file to be in raw sector order, i.e. starting with track 0 sector 1, then track 0 sector 2, track 0 sector 3 etc. In this case Ersatz-11 should not alter the interleave, and in fact it should do the interleave itself if the file is mounted as something other than a virtual floppy drive (since PDP-11 drivers for other devices don't do the floppy-style interleave).

By default, Ersatz-11 decides whether a file is a block or a sector image by the file size:

size (bytes)	*type*
256,256	RX01 sector image
512,512	RX02 sector image
1,025,024	“RX03” (DS RX02) sector image
(anything else)	block image

Block images of RX01/02/03 disks are slightly smaller because the interleave scheme leaves out track 0, so they can be distinguished by size alone, unless padding was added during transfer or something else altered the file size. RX50 image files are the same size either way so by default they are assumed to be block images. The defaults may be overridden with the “`/BLOCK`” and “`/SECTOR`” switches.

Since DEC’s 8” floppy interleave scheme doesn’t use track 0, data from this track do not normally appear in a block image file. However some non-standard software may need to use track 0, so the RX11, RX211, and RXT11 emulation relocates it beyond the end of the block image, if the file is enlarged by the size of one cylinder to be the same size as the equivalent sector image file. Use “`/BLOCK`” to specify that it’s still a block image.

type	*base size (bytes)*	*extra size (bytes)*
RX01	252,928	3,328
RX02	505,856	6,656
“RX03”	1,011,712	13,312

If the base file sizes are used, these files work as regular block images and track 0 does not exist. Any attempt to write track 0 is a no op, and any attempt to read track 0 returns hex E5 in every data byte, as if the disk were freshly formatted.

NOTE

> The Demo and Lite versions of E11 limit the combined size of all image files to 32 MB. E11 versions prior to 2.0 did not have this restriction, but they are no longer supported by D Bit.

Floppy disk drives

`MOUNT` *ddcu*: *d*: [*switches*]

Floppy disk drives may be used to emulate any block-replaceable device supported by E11. “*d*:” is the drive letter, i.e. `A:` or `B:`, and there are switches to specify the PC drive type:

switch	*drive type*	*size in blocks*	*disk types*
`/RX01`	5.25” or 8” SS SD	494	RX01
`/RX02`	5.25” SS DD	988	RX01, RX02
`/RX03`	5.25” DS DD	1,976	RX01, RX02, RX03
`/RX23`	3.5” 1.44 MB	2,880	RX23, RX24
`/RX24`	3.5” 720 KB	1,440	RX24
`/RX26`	3.5” 2.88 MB	5,760	RX23, RX24, RX26
`/RX33`	5.25” 1.2 MB	2,400	RX33, RX50, “RX52”
`/RX50`	5.25” SS DD	800	RX50
`/RX52`	5.25” DS DD	1,600	RX50, “RX52”
`/MY`	5.25” DS DD	1,600	`MY:` (Russian)

As shown in the table above, most drive types support one or two smaller disk formats in addition to their own. This means that any disk type supported by a given drive type may be inserted at any time with no need to re-`MOUNT` the drive. E11 will automatically detect the format of the new disk and adjust to the new geometry. Size changes are visible to PDP-11 controllers that support them, i.e. the `DY:`, `DU:`, and `HD:` controllers.

Note that confusion is possible if a floppy disk has a different total number of blocks than the device being emulated. The PDP-11 OS may try to access areas off the end of the disk (which results in a controller-specific I/O error) or may not use all of the disk, and in particular writing a blank file system (with an OS-specific “initialize volume” command) will result in a directory structure that doesn’t match the actual volume size. Care should be exercised to avoid trouble. Like disk types are of course not a problem, so for example “`MOUNT DX0: B: /RX01`” will mount a real RX01 disk to be used as an emulated RX01 disk. The `HD:` device works with any size device, so all floppy types may be mounted on `HD:` if you have the “HD.SYS” device handler (under RT-11). If the disk already has a correct directory structure for its actual size, and is mounted to emulate a device of at least that size, most operating systems (with device-independent file systems) will be able to read and write the disk correctly. For example, if you initialize an RX23 floppy with RT-11 directory structure using the PUTR utility under DOS, and then “`MOUNT DL0: A: /RX23`” in E11 (using the 1.44 MB RX23 disk to emulate a 5 MB RL01 pack), RT-11 will be able to access all files on the disk, and can write new files without data corruption, only the RT-11 `INITIALIZE` command needs to be avoided.

Most of the floppy disk formats supported by E11 are exact equivalents to PC formats. Not all of them were ever supported by DEC for use on PDP-11s, but some were supported by aftermarket controllers. The RX01/02/03 formats use 1.2 MB disks, specially formatted (by E11 or the PUTR utility) to have the exact same geometry as their 8” counterparts. The RX01 format is in fact identical to the IBM 3740 format used on DEC’s 8” disks, so if an 8” drive is attached to the PC floppy disk controller using a special cable, it will be possible to read/write/format real RX01 disks. DEC uses a modified MFM format for RX02 disks, which the PC hardware is incapable of using, so E11 substitutes the IBM System/34 compatible format for the RX02/RX03 workalike disks. The RX01 format is not guaranteed to work (on either 5.25” or 8” drives) because most PC floppy controllers don’t have a working single density mode (the PC BIOS is hard-coded for double density so manufacturers have dropped support for “unneeded” features). However some floppy controller chips made by SMC, Goldstar, and Western Digital are known to work with single density mode. These chips usually contain “37C65” in the part number and use a 16/32 MHz clock plus a 9.6 MHz clock rather than the single 24 MHz clock used by many other controllers, so it may be useful to shop for a floppy controller board that has two crystals on it instead of one, in the absence of better information. The “CompatiCard IV” controller made by Micro Solutions, Inc. works with single density disks and is supported by E11. The same company sells an adapter cable for use with 8” drives.

In addition to the PC formats and 8” workalike formats, E11 also supports several 5.25” formats. RX50 disks are supported using a 1.2 MB drive. “RX52” is E11’s name for double-sided RX50 disks, which were never sold by DEC, but may have been planned. “MY” is the device name for the DS DD 5.25” disks used on the Russian DVK-x PDP-11 clones; the low-level format is the same as the “RX52” but there is no software sector interleave.

"MX" disks are not supported by the PC disk controller so E11 can't use them.

E11 can format all the disk types that it supports. Currently the only way to get at this feature is using the "set density" command of the RX211 controller, which can be executed by running `FORMAT.SAV` under RT-11 (using commands like "`FORMAT DY0:`" or "`FORMAT DY0:/SINGLE`"), or the `SPEC%()` function under RSTS. E11 chooses which of the formats (supported by a given floppy drive type) to use based on the size of the emulated disk, it uses whichever format is the same size as the emulated volume being formatted, or the next larger format if there's no exact match.

POTENTIALLY IMPORTANT NOTE

> The simulator has no control over any caching of disk writes that DOS may do, so it is important that you `QUIT` out of the simulator to make sure all the pseudo-disk files get closed properly, rather than simply switching the computer off, after shutting down your PDP-11 OS. Otherwise there is no guarantee that all data written to disk by the PDP-11 has really made it onto the DOS disk (in practice I've had no problems, DOS and most disk cache programs don't live this dangerously, but trouble is theoretically possible so you've been warned). The floppy disk support controls the hardware directly (and it intercepts `INT 13h` to keep DOS's fingers out of things so you can't crash the machine by trying to `MOUNT` an image file located on a DOS floppy while floppy emulation is active) so as long as the PDP-11 is done with the disk it's OK to take it out of the drive.
>
> Also, the simulator has no control over "Green PC" BIOSes which spin down hard drives after a specified period of inactivity. When E11 accesses the drive after it's been shut down, the BIOS will pause several seconds while it spins the drive back up. During this period E11 is not running, so the simulated PDP-11 may drop incoming characters (but maybe not, E11 maintains a small FIFO buffer on each line) and its clock will lose a few seconds. If this is a problem you may have to disable this BIOS feature.

Supported PDP-11 tape devices:

name	*units*	*controller*	*drive/volume switches*
`CT:`	0–1	TA11	(none, always TU60 DECassette)
`MM:`	0–7	RH11/RH70, TM03	`/TE16`, `TU45`, `TU77`, `/SERIAL:`*nnnn*
`MS:`	0–7	TS11 etc.	(none, always matches controller)
`MT:`	0–7	TM11	(none, always TU10)
`MU:`	0–65535	TMSCP	`/TYPE:`*xxxyy*

For TMSCP tapes, the `/TYPE:`*xxxyy* switch sets the unit type name, which may be up to three letters and up to two decimal digits. The default is TU81.

For MS: tapes, DEC's earlier controllers support only one drive per controller so PDP-11 operating systems normally use the unit number to distinguish between separate controllers. Like later controllers, E11 allows up to 8 units per controller, so E11 uses the controller letter to distinguish between multiple controllers, the same as with most other device types. This means that when emulating a typical DEC system with one unit per controller, drives that the operating system calls MS0:, MS1:, and MS2:, will be called MSA0:, MSB0:, and MSC0: by E11, since they're the first and only slaves on three separate controllers. This only comes up with there is more than one MS: style tape drive.

Supported PC virtual tape devices:

Tape image files

`MOUNT` *ddcu*: *d:path\filename*[`.TAP`] [*switches*]

A tape image file contains a byte-by-byte image of tape data, with headers and trailers on each record to maintain the blocking data from the real tape. Each record looks like this:

```
.LONG   LEN     ;32-bit record length, LSB first, byte-aligned
.BLKB   LEN     ;LEN bytes of data
.LONG   LEN     ;the length again, for backspacing
```

A tape mark appears as a single 32-bit 0. The `MOUNT` command for an image file may include a `/MAXRECORD:`*n* switch, which sets the maximum possible record length that can be read or written on that unit; the default is 10240 bytes. E11's memory usage may be decreased by using a smaller number, but data will be lost if the PDP-11 attempts to read or write records larger than the specified maximum. Both ANSI and DOS-11 labeled tapes normally have a maximum record length of 512 bytes, but BRU and UNIX "tar" tapes use longer records. As with disks, there are also `/RONLY` (syn. `/WPROTECT`), and `/RW` switches, to optionally write lock a tape drive.

TA11/TU60 DECcassette

The TA11/TU60 cassette tape system requires a mandatory load point gap (i.e. tape mark) on all tapes. E11 simulates this internally so that the load point gap does not appear on the image file; this allows the TU60 general access to tape images that may have been created using some other device emulation, where a mandatory tape mark at BOT would violate the labeling standards.

The "`LOG CT`*n*`:`" command may be used to find out whether a PDP-11 program is trying to exceed the value set with `/MAXRECORD`, or if it is guessing incorrectly about the length of a tape record; the TA11 must be told by the program when a record is ending even when reading, unlike 9-track drives which detect record gaps automatically as part of a read operation).

RH11/RH70, TM03 magtapes

Massbus tapes are complicated slightly by the fact that each formatter supports up to 8 drives, and most systems have only one formatter even if they have multiple drives. As a result, the most common naming convention for "`MM`*n*`:`" device names uses the unit number *n* to refer to the slave number within the single formatter, rather than the Massbus unit number (which is what referred to by the unit number with all Massbus disks), and the formatter is Massbus unit 0.

E11 uses an extended `MM:` device name syntax similar to that used by RSX, where each slave's device name looks like "`MM`*cu_s*`:`". *c* is a letter indicating which RH11/RH70 controller connects to the TM03 formatter; the default in this version of E11 is "`C`", the letter may change in future versions but in any case it refers to the default tape Massbus adapter at (17)772040. *u* is the Massbus unit number of the formatter, which defaults to 0 and is in the range 0–7. *s* is the slave number (within a TM03 formatter) of the tape transport, which also defaults to 0 and is in the range 0–7. If a number is present but no "_", that number is the slave number, not the Massbus unit number. The effect is that if the controller letter and Massbus unit number are omitted leaving a device name like "`MM3:`", this name has the same meaning as the usual RT-11 or RSTS name, which is: default tape RH11 (the one at (17)772040), default formatter (0), slave 3. Meanwhile additional fields may be supplied to identify any of the 64 possible slaves on any of the (currently 3) possible Massbusses, so "`MMA2_5:`" refers to the first RH11 (which is at (17)776700 in E11 V2.0), TM03 formatter 2, slave 5. This same name format may also be used in any other command (e.g. `BOOT`, `LOG`) that takes a device name.

The `MOUNT MM:` command has switches to identify the drive model, but their only effect is to set the value of the "drive type" register. From a PDP-11 software point of view, all drives attached to a TM03 formatter look the same, the only difference is speed. There is also a `/SERIAL:`*nnnn* switch, which sets the value of the "drive serial number" register.

`MOUNT PP:` *file*

Mounts the specified file to receive output sent to the PC11 high speed paper tape punch. The default extension is "`.PAP`".

`MOUNT PR:` *file*

Mounts the specified file to provide input read from the PC11 high speed paper tape reader. The default extension is "`.PAP`".

`PROCEED` [*break*]

Continues PDP-11 execution at the address currently in the program counter. If "*break*" is specified, then it is the virtual address of a single hard breakpoint, where the PDP-11 is guaranteed to stop if an instruction fetch is attempted starting at that address, regardless of what mode the computer is executing in, and regardless of whether the contents of that location have changed since the breakpoint was set. This can be handy for tracing code that hasn't been loaded yet. Note that hard breakpoints and single stepping with the `STEP` command interfere with the operation of the PDP-11 T bit, so don't combine them with a debugger (or CPU traps diagnostic program) running on the PDP-11 or you'll get strange behavior.

`QUIT`

Exit the simulator, closing all image and log files and resetting all devices that were in use.

`REGISTER` [*r val*]
reg=*val*
flag=*val*

If "*r*" and "*val*" are given, sets register "*r*" (0–7) in the current register set to contain "*val*." Otherwise displays the values of all eight registers, the condition codes, the current and previous processor modes, and the current interrupt priority level. Registers and condition code flags may also be set by typing the register name, an equals sign, and the new value at the command prompt. Any expression that works with `CALCULATE` is valid in this case, so for example one may type "`PC=PC-2`." The CPU priority may be set in the same way using "`PRIO=` *val*", where *val* is from 0 to 7. Also the current mode and previous mode may be set with "`CM=`*x*" and "`PM=`*x*", where *x* is `K`, `S`, or `U` for kernel, supervisor, or user mode.

`SET CLOCK 14318`*nnn*

Informs E11 of the actual frequency (in Hertz, as a decimal number) of the PC motherboard's 14.318 MHz system clock used to derive the 50/60 Hz clock, among other things. This frequency is supposed to be 14318180 Hz (which is what E11 assumes by default), but if your PDP-11 OS's clock gains or loses time at this setting you can use SET CLOCK to make slight changes, and E11 will adjust its math accordingly. The correct value can be determined experimentally, or measured using test equipment (this would require a very accurate frequency counter to be useful though). Setting values wildly different from 14318180 may produce unexpected results.

`SET CPU` *item* [*item* ...]

This command changes the emulated CPU type, either by changing to a new model all at once, or on a feature-by-feature basis. Each keyword enables a particular feature, or disables it if preceded by "`NO`". Any number of keywords may be specified in one line, and they are applied left to right so for example "`SET CPU 44 NOFPP`" will create a PDP-11/44 and then delete its floating point processor. This gives you the ability to roll your own CPU, which need not correspond to any actual existing PDP-11 model. Changing the CPU's type while it is running will work but is likely to crash the PDP-11 operating system. `SHOW CPU` shows the current settings of all options.

E11 does not emulate cache memory, since that would greatly slow down emulation rather than speeding it up. Maintenance features such as "write wrong parity" are not emulated either, since again they would needlessly add huge overhead and anyway since the data paths being tested by these modes are all different on a PC, so PDP-11 diagnostic software would not gain any useful information by exercising them. So for these cases E11's emulation is limited to creating the appropriate registers in the I/O page so software can read and write them without losing data or receiving unexpected bus timeout errors. Note that if RSTS/E sees a parity CSR or KTJ11B maintenance CSRs it attempts to exercise them, giving a fatal error if they do not work; to avoid this problem, the CPU configuration given by `SET CPU 94` has `NOKTJ11B` and `NOPARCSR` by default as a workaround. These CSRs may still be enabled for software that needs them with `SET CPU 94 KTJ11B PARCSR`, however both RT-11 and RSX11M+ will work with the default setting.

The real mode versions of E11 are further limited by the constraints of DOS memory. `MMU22` enables the 22-bit MMU emulation, but the emulated PDP-11 is limited to available DOS memory, rather than the 3840 KB or 4088 KB limit of the full version of E11. The real mode versions also don't have space for an opcode dispatch table big enough to fully decode instruction operands. This means that the `DESTFIRST` and `JMPPLUS2` options can't be supported without significantly slowing down all `JMP`, `JSR`, and double operand instructions, so these options do nothing in the real mode versions of E11.

`SET CPU` options:

number	Set all CPU options to match PDP-11/*number* model. Recognized values are 23, 24, 34, 35 (syn. 40), 44, 45 (syn. 50 or 55), 53, 70, 73, 83, 84, 93, 94.
`ASH31`	J11 CPU bug with 31-bit shifts using ASH/ASHC.
`ASR`	KDJ11E additional status register (TOY clock etc.).
`CCR`	Cache control register (at (17)777746).
`CD`	Cache disable bit in PDRs.
`CDR`[`=`*n*]	KDJ11x configuration/display register (at (17)777524), *n* is 8-bit DIP switch value.
`CHR`	Cache hit register (at (17)777752).
`CHRNZ`	If CHR is present, reads as non-zero.
`CMDR`	PDP-11/44 cache memory data register (at (17)777754).
`CPUERR`	CPU error register.
`CSM`	`CSM` instruction (requires `SUPMODE` to work).
`DESTFIRST`	Evaluate destination operand first in dual operand instructions with mode 0 source. Effect is to use incremented/decremented value of register as source with mode 2-5 destination using same register, or PC+2 for mode 07 source and mode 67 or 77 destination.
`DSPACE`	Split I/D space.
`DUALREGSET`	Dual register set.
`EIS`	Extended (integer) Instruction Set.
`EIS16`	EIS instructions with odd destination set condition codes based on 16-bit result.
`FPA`	Floating point accelerator, sets KDJ11x MR bit indicating FPJ11 present.
`FPBACKOUT`	J-11 SR1 behavior, autoinc/dec is always undone on aborted FPP instruction.
`FPP`	FP11 floating-point instruction set.
`HALT4`	`HALT` in user mode traps to 4 instead of 10.
`JMP4`	`JMP R`*n* or `JSR R`*n* traps to 4 instead of 10.

`JMPPLUS2`	`JMP (R)+` and `JSR X,(R)+` jump to incremented value of `R` (R+2).
`KTJ11B`	KTJ11B Unibus adapter maintenance registers.
`MFPT`[=*n*]	`MFPT` instruction (returns *n* in R0).
`LKS7`	Bit 7 (monitor) of LKS clock status register is mechanized.
`MBR`	PDP-11/70 microprogram break register (at (17)777770).
`MSEA`	Memory system error address register (at (17)777740/2).
`MSER`	Memory system error register (at (17)777744).
`MMTRAPS`	11/45,55,70-style memory management traps, 3-bit ACF.
`MMU`	Memory management unit.
`MMU22`	22-bit MMU (must use `UMAP` too if emulating Unibus CPU).
`MR`[=*n*]	Maintenance register (at (17)777750). If n<16., KDJ11x-style maintenance register which reads *n* as model code in bits 7:4. If n=44, PDP-11/44 style MR, and if n=70, PDP-11/70 style MR.
`MXPS`	`MFPS`, `MTPS` instructions.
`ODD`	Odd address trapping.
`PARCSR`	Parity/ECC memory CSR address (at (17)772100).
`PCR`[=*x*]	KDF11/BDV11 page control register and read/write register if x="`KDF11`", or KDJ11 CSR/page control register if x="`KDJ11`" (at (17)777520/2).
`PIRQ`	11/45-style 7-level software interrupts.
`PSWIO`	PSW accessible from I/O space (at (17)777776).
`QBUS`	Q-bus exists (otherwise Unibus).
`SIZE`	11/70 system size registers (at (17)777760/2).
`SPL`	`SPL` instruction.
`SR`	Switch register/display register (at (17)777570).
`SR1`	MMU status register 1.
`STACKLIM`	PDP-11/70 stack limit register (at (17)777774).
`SUPMODE`	Supervisor mode.
`SYSID`[=*n*]	PDP-11/70 system ID register (at (17)777764), returns *n* when read.
`TSTSET`	J-11 `TSTSET`, `WRTLCK` instructions.
`UMAP`	Unibus map (maps 18-bit I/O bus to 22-bit memory).
`UNDOAUTO`	Undo mode 2/3 autoincrements on bus error etc.

`SET DELAY` *device* c_1:n_1 c_2:n_2 ...
`SET DELAY` *device* `*`:n

Sets the number of instructions that the specified command opcodes appear to take to complete on the indicated device. The *device* may be `DELUA`, `DL11`, `DOSFILE` (the DOS file access pseudo-device), `KW11L`, `LP11`, `PC11`, `RC11`, `RF11`, `RK11D`, `RK611`, `RL11`, `RP11C`, `RS03` or `RS04` (synonyms), `RX11` or `RX211` (synonyms), `TA11`, `TC11`, `TM03`, `TM11`, or `TS11`. There may be an arbitrary number of parameters of the form "*c*:*n*" or "*c*=*n*", where *c* is the octal opcode for the command (or "`*`" for all commands for this device) and *n* is the decimal number of PDP-11 instructions to delay before signaling completion of the command.

The reason that device commands, such as "*read sector*" on an RX02, or "*transmit character*" on a DL11, delay signaling completion (by raising a "*ready*" flag and/or triggering an interrupt) instead of completing right away (which would seem natural in an emulator) is that some OS software contains assumptions that at least a certain number of instructions are guaranteed to be executed before a device is able to interrupt, even when interrupts from that device are enabled. The default interrupt delays are set for the "worst case", so that each one is long enough to avoid any known (or suspected) problems with DEC OS software. The `SET DELAY` command may be useful in cases where your OS needs a longer delay than the default, or in cases where your OS's treatment of a device is "clean" and you can gain a noticeable I/O speed increase by setting all the delays for that device to 1, or in cases where you're debugging OS software and want to test against variety of interrupt rates. If this means nothing to you, you can safely forget about it (all you lose is a little speed), this command is intended mainly for advanced users.

Note that some devices don't have numbered command opcodes per se, but the `SET DELAY` command syntax requires one anyway for consistency, and pseudo opcode numbers are assigned if necessary. On DL11 SLUs, opcode "0" refers to the delay between reading a character from the receiver buffer, and getting the interrupt for the next character (only if it's the second or later character of a function key sequence on an emulated VT100, all other keyboard interrupts correspond to actual asynchronous events); and opcode "1" refers to the delay between writing a character to the transmitter buffer, and getting the completion interrupt (for emulated VT100s; COM and LPT ports use real completion interrupts). Similarly, PC11 opcode "0" refers to how long it takes to read a character from paper tape, and opcode "1" refers to how long it takes to write one. An LP11 has only one opcode, which is "0" and corresponds to the same thing as opcode "1" of a DL11. An RK611 has only opcodes 0–17, but the `SET DELAY` command defines an opcode "20" which refers to the delay between the interrupt that acknowledges reception of a head movement command (which is itself delayed), and the "attention" interrupt which signals completion of the head movement. The RK11D emulation has a similar dummy opcode "10" which means the same thing, and the TA11 emulation has a dummy opcode "10" which defines the delay between character interrupts within a block. A KW11L has no opcodes, so opcode "0" sets the delay between simulated interrupts which are used to catch up if clock interrupts are missed due to DOS I/O taking more than 16.67 ms (or 20 ms in 50 Hz clock mode) to complete.

`SET DISPLAY NONE`
`SET DISPLAY PORT` *n*
`SET DISPLAY LPT`*n*`:`

If `PORT` is specified, specifies the 80x86 I/O address (as an expression with the same syntax as used by the `CALCULATE` command) of a word port which when written as a word, sets the 16-bit display register. Building the trivial hardware to support this is left as an exercise to the reader.

If a PC LPT port name is given, it specifies a port which has a multiplexed LED board plugged into it, and E11 will refresh each half on alternate 60Hz (50Hz) clock ticks; there's a little flicker but it works and requires no chips or power supply, just build your board so that the `D0`–`D7` lines (pins 2–9 of the DB25) drive the anodes of the both the `D0`–`7` and `D8`–`15` LEDs (through the same set of eight 100 ohm resistors since only one set of LEDs will have their cathodes grounded at a time). Then add two NPN switching transistors (2N3904 etc.), one for each byte, with the emitters grounded (pin 25), each collector connected to the cathodes of all 8 LEDs for the appropriate byte, and the bases connected through 1K current limiting resistors to $\overline{\texttt{STROBE}}$ (pin 1) for the `D0`–`D7` side, or $\overline{\texttt{INIT}}$ (pin 16) for the `D8`–`D15` side. A bare PC board is available from the author at cost ($14.68 plus shipping for the current batch, 12/94).

If `NONE` is specified, then the current DR value is available only from the `SHOW DISPLAY` command (the default condition).

`SET DISPLAY` {`DR | BDR | R0 | PC`}

Sets what register is displayed on a hardware LED display register (either the parallel port kind described above or the kind that plugs into a bus slot and is addressed by a word `OUT` instruction). By default the DR is displayed (i.e. the last word written to (17)777570), but the BDR (boot/diagnostic display register, i.e. the last byte written to (17)777524), or R0 or the PC may be selected instead, since the null jobs in some operating systems display a pattern in R0 (and the PC in some cases) during a `WAIT` instruction. The pattern may be used to get a rough idea of system load, and the R0/`WAIT` method is a standard way to display a number on the PDP-11/70, which has no display register address. For completeness, registers R1–R5 or SP may be selected too.

SET HERTZ {50 | 60}
SET HZ {50 | 60}

Sets the frequency of the KW11L line clock (startup default is 60); Ersatz-11 reprograms the PC timer chip for this rate to simulate line time clock interrupts, and then maintains a count in software so that it knows when to trigger BIOS 18.2 Hz interrupts; there is some jitter due to the BIOS interval being rounded down to the previous KW11L interrupt, but there is no cumulative error so the DOS clock is still correct when you exit out of E11.

SET KEYBOARD [NO]SWAP

SWAP sets the keyboard handler to exchange the functions of the Caps Lock and left Ctrl keys for people who don't like the IBM Enhanced Keyboard. NOSWAP sets the handler back so that the keys work as marked.

SET [NO]SCOPE

Sets whether the console terminal is a scope or a hardcopy terminal, for the purpose of handling rubout characters typed at the "E11>" prompt. Mainly useful if the console is redirected to a COM port with a DECwriter (etc.) plugged into it. Also determines whether typing ^L at the command prompt will attempt to clear the screen.

SET SCROLL {HARD | SOFT}

Selects the mechanism used for video scrolling. HARD scrolling offers superior performance (it works by programming the video board(s) to change the start address of the screen each time a full-screen scroll is needed), but may expose problems in video virtualization in GUI DOS boxes, or cause problems with TSRs that use the display. SOFT scrolling does things the slow obvious way, by copying the whole screen a line up on each line feed, and ought to work with anything. HARD scrolling is the default; you should try putting SET SCROLL SOFT in your E11.INI file if your display gets scrambled a few lines into each attempted E11 session.

SET SWITCH *n*
SET SWITCH PORT *n*

If PORT is specified, specifies the 80x86 I/O address (as an expression with the same syntax as used by the CALCULATE command) of a word port which when read as a word, gives the current 16-bit switch register value. Otherwise (PORT not specified), sets the value of the emulated SR to *n* (again as a CALCULATE-style expression).

`SET` *pcdevcu*`:` ...

Sets parameters for the specified PC device (with the optional controller letter and line number given using the usual device name syntax). Possible parameters are as follows:

name	*device*	*options*
`COM`*u*`:`	COM port	`IRQ`*n* `PORT=`*nnnn* `SHARE`
`LPT`*u*`:`	LPT port	`BIOS IRQ`*n* `NOBIOS NOIRQ PORT=`*nnnn* `SHARE`
`NE2000`*c*`:`	NE2000 NIC	`PORT=`*nnnn* `IRQ`*n* `SHARE AUTO PCI ISA`
`SBMIDI:`	MIDI port	`IRQ`*n* `PORT=`*nnnn* `SHARE`

The options are defined as follows:

`BIOS` For LPT ports, tells E11 to use the BIOS to send data to the printer. Mainly useful with network software that captures printer output. PORT and IRQ settings are not meaningful with BIOS printers, and may be set to any value.

`IRQ`*n* Sets the IRQ number (0–15) used by the board. For COM ports, the default IRQ is IRQ4 if the I/O port address is 300 (hex) or more, or IRQ3 for 2FF (hex) or less. These defaults have been traditional since IBM XTs were the standard, and most multi I/O boards will be set up this way. However it's not uncommon with newer boards for COM3 to use IRQ5 and COM4 to use IRQ2 (or IRQ9 really, which is effectively the same on an AT) by default. For LPT ports, the default IRQ is IRQ7.

`NOBIOS` For LPT ports, tells E11 to use its own built-in LPT drivers rather than the PC BIOS to send data to printers. This is the default setting.

`NOIRQ` Sets LPT ports to use polled I/O based on a timer rather than actual interrupt-driven I/O. This option is useful for LPT ports only, ASSIGN commands will fail if it is used with any other port type.

`PORT` Sets the base I/O port address (in 80x86 I/O space) used by the board. Numbers are hex by default as long as they start with a digit 0–9, otherwise any expression may be used, the same as with `CALCULATE`.

`SHARE` Specified along with an `IRQ`*n* parameter, SHARE means that the device has special IRQ sharing hardware (as documented in the *IBM AT Technical Reference Manual*) so that more than one device may use the same IRQ at once; this hardware is relatively rare, so if you don't know whether you have it, you probably don't. Note that without this hardware it's not possible for more than one device to be actively using the same IRQ at the same time; so for example if you have a mouse attached to COM1 using IRQ4 and it has been initialized by a mouse driver, E11 will not be able to use COM3 if it also uses IRQ4 (an error message will be given if you try to issue an ASSIGN command for COM3).

`SET` *ddcu*: ...

Sets parameters for the specified PDP-11 device. Possible parameters are as follows:

name	*controller*	*options*
`CT:`	TA11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`DC:`	RC11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`DF:`	RF11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`DK:`	RK11D	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`DL:`	RL11	`CSR=`*nnnnnn* `VECTOR=`*nnn* *ctrltype* (`DEFAULT`, `RL11`, `RLV11`, `RLV12`)
`DM:`	RK611	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`DP:`	RP11C	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`DS:`	RH11/RH70	`CSR=`*nnnnnn* `RH11` `RH70` `VECTOR=`*nnn*
`DT:`	TC11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`DU:`	MSCP	`CSR=`*nnnnnn* *ctrltype* (`DEFAULT`, `KDA50`, `RQDX3`, `RQZX1`, `UDA50`, `UDA50A`)
`DX:`	RX11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`DY:`	RX211	`CSR=`*nnnnnn* `DS` `SS` `VECTOR=`*nnn*
`HD:`	virtual	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`LP:`	LP11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
		`MODE=`*bps*`,`*par*`,`*dbits*`,`*sbits*
`MM:`	RH11/RH70,TM03	`CSR=`*nnnnnn* `RH11` `RH70` `VECTOR=`*nnn*
`MS:`	TS11 etc.	`CSR=`*nnnnnn* `VECTOR=`*nnn* `DEFAULT`, `TQK25`, `TS11`, `TSU05`, `TSV05`, `TU80`
`MT:`	TM11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`MU:`	TMSCP	`CSR=`*nnnnnn* *ctrltype* (`DEFAULT`, `KLESI`, `TQK50`, `TQK70`, `TUK50`)
`PD:`	RXT11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
`PR:`	PC11	`REWIND`
`RH:`	RH11/RH70	`CSR=`*nnnnnn* `RH11` `RH70` `VECTOR=`*nnn*
`TT:`	DL11	`CSR=`*nnnnnn* `VECTOR=`*nnn*
		`MODE=`*bps*`,`*par*`,`*dbits*`,`*sbits*
`XE:`	DELUA	`CSR=`*nnnnnn* `VECTOR=`*nnn* `BOOTSTRAP=`*ddcu*`:`[`/`*os*]

The options are defined as follows:

`BOOTSTRAP=`*option* Selects whether incoming *MOP* boot frames will be honored, specifies either the `BOOT` command parameters, or `DISABLE` (default) to disable network-initiated booting. Remote booting is not yet supported, so this command has no visible effect.

`CSR=`*nnnnnn* Sets the base CSR address to *nnnnnn* octal.

`CSR=FLOATING` Sets the base CSR address according to the PDP-11 floating CSR rules, for devices that support this scheme. CSR addresses for all floating devices are recomputed every time the device configuration is changed. `SHOW` commands will display "(F)" after an address that was chosen using `FLOATING`.

ctrltype Set controller type. This is mainly used to distinguish between Unibus and Q-bus versions of controllers, which differ in number of address bits used and whether they go through the Unibus map, and in some cases in other ways as well. `DEFAULT` means to choose a default controller type based on whether `SET CPU QBUS` is in effect. `SHOW` commands will display "(D)" after a controller type which was chosen using `DEFAULT`.

`DS/SS` Sets the disk to be single-sided or double-sided; real RX211s autodetect this but 3.5" and 5.25" disks don't have a separate index hole for DS disks, so the number of sides must be set explicitly.

`MODE` Sets the mode parameters for the specified port using a syntax similar to the DOS MODE command. The four parameters "*bps*,*par*,*dbits*,*sbits*" set the decimal number of bits per second (which unlike DOS MODE, may not be abbreviated, so "96" really means 96 baud not 9600), a single letter indicating the parity ("E" for even, "O" for odd, "N" for none, "M" for mark, "S" for space), a decimal number of data bits (5–8), and a decimal number of stop bits (1–2). This is mainly useful for DL11 type ports where the communications parameters are not set by software, for mux ports any parameters set with `SET MODE` will be overridden by the values written to the mux by the driver in the PDP-11 OS. Note that the set of valid baud rate values depends on the hardware emulating that port, if the value specified is not available E11 will use the closest rate the device supports.

`REWIND` Rewind the paper tape, so that subsequent input will start over at the beginning of the `MOUNT`ed file.

`RH11` Sets this Massbus adapter (specified by controller letter, unit number is meaningless) to be an RH11; 18-bit addressing, goes through Unibus map if one is configured with `SET CPU UMAP`. This is the default for all Massbus disks and tapes.

`RH70` Sets this Massbus adapter to be an RH70; 22-bit absolute addressing, RHBAE/RHCS3 registers exist.

`VECTOR=`*nnn* Sets the base vector address to *nnn* octal.

`VECTOR=FLOATING` Sets the base vector address according to the PDP-11 floating vector rules, for devices that support this scheme. Vector addresses for all floating devices are recomputed every time the device configuration is changed. `SHOW` commands will display "(F)" after an address that was chosen using `FLOATING`.

`SHOW BDR`

Shows the current value of the boot/diagnostic display register (last value written to (17)777524).

`SHOW CLOCK`

Shows the current assumed frequency, in Hz, of the PC motherboard's 14.318 MHz clock, used to generate the KW11L 50/60 Hz clock.

`SHOW CPU`

Shows emulated CPU type, along with breakdown of features, as well as the CPU type of the host processor (and whether the Pentium `FDIV` bug is present).

`SHOW CSR` *addr*

Shows the name of the device register at the specified octal I/O page address.

`SHOW DELAY` *device*

Shows the currently active list of interrupt delay counts for the specified device, starting with the delay for opcode number 0. See `SET DELAY` for details.

`SHOW DISPLAY`

Shows the current value of the display register (last value written to (17)777570).

`SHOW HERTZ`
`SHOW HZ`

Shows the current frequency, in Hz, of the emulated KW11L line clock. The default is 60.

`SHOW KEYPRESS` *keyname*
`SHOW KEYRELEASE` *keyname*

Shows the script currently bound to the keypress or keyrelease event for the specified key, if one is defined. See `DEFINE KEYPRESS` for key names.

`SHOW LED` *ledname*

Shows the name of the flag (from the keyboard script language) whose value is being tracked by the specified LED, or "`NONE`" if the LED has been disabled. LED names are `CAPS`, `NUM`, and `SCROLL`.

`SHOW MEMORY`

Shows the amount of DOS memory used by Ersatz-11, how much of that memory is emulated PDP-11 memory, and how much DOS memory is free. Free memory must be available in order to use the `LOG` commands, or to create Ethernet devices, or to create DL11 or LP11 devices attached to video screens accessible using function keys.

`SHOW MMU` [{`KERNEL | SUPERVISOR | USER`} [`INSTRUCTION | DATA`]]

Shows the current mapping for the specified space. Defaults are `KERNEL` and `INSTRUCTION` space.

`SHOW VERSIONS`

Shows the version numbers of Ersatz-11, the operating system, and any other relevant interfaces.

`SHOW` *ddcu*:

Shows the configuration of the specified PDP-11 or PC device. The actual data shown are dependent on the device type, in general the display contains the information that can be `SET` for that device (e.g. controller type, or CSR/vector addresses), and for PDP-11 devices the name of the PC file or device attached with `ASSIGN` or `MOUNT` is shown. Not all devices support `SHOW`.

In the output from a `SHOW` *ddcu*: command, "(D)" following a controller type means that the controller type is set to "DEFAULT" and the displayed type is the default based on the "SET CPU [NO]QBUS" setting. Similarly, "(F)" following a CSR or vector value means that that parameter is set to "FLOATING" and the value given is the one currently in use based on the system configuration.

`STEP` [*count*]

Executes the specified number (default=1) of single instruction steps and displays the updated registers after each. Note that if real time clock interrupts are enabled and the CPU priority is below 6, `STEP` will immediately enter the clock interrupt service routine instead of executing the instruction at the current PC. An easy workaround is to disable clock interrupts with "`D 17777546 0`" before using `STEP`, and then reenable them with "`D 17777546 100`" before continuing regular execution.

`UNLOAD` [*/switches*] [*address*]

Unloads a ROM or EEPROM page previously loaded with "`LOAD`". Either the "`/ROM`" or "`/EEPROM`" switch is required, "`/BANKED`" may be given to invoke the default starting addresses of (17)773000 for "`/ROM`" and (17)765000 for "`/EEPROM`", otherwise the starting address of the ROM must be given.

HISTORY

V0.8 BETA, 29-Mar-1994; initial release.

V0.9 BETA, 05-Jul-1994; many bug fixes (mainly trap handling, MMU emulation, `DIV` instruction, and VT100 reverse video), added RX211 emulation, multiple DL11s, and 50 Hz KW11L mode.

V1.0 BETA, 14-Nov-1994; more bug fixes, added FP11A, RK611/RK06–07, LP11, D-space, and supervisor mode emulation. Also `CALCULATE`, `HELP`, `INITIALIZE`, `LOG`, `SET/SHOW CPU`, `SET DR LPT`*n*`:`, `SET SCROLL`, `SHOW MMU` commands, VT100 graphics/underline, changed to `.EXE` file (ran out of space in unified code/data segment in .COM file).

V1.1 BETA, 22-Mar-1995; still more bug fixes (IAS finally works), DELUA Ethernet emulation, disk `LOG`ging, indirect command files, workaround for Pentium `FDIV` bug, help text moved to file, Russian `HD:` device (and RT-11 driver), PC11, display general registers on parallel port LED board.

V2.0 DEMO, 20-Jul-1997; many bug fixes as usual, limited 22-bit MMU with and without Unibus map. MMU SR1 mechanized, TOY clock, CPU emulation extended to include 11/24, 11/44, 11/45, 11/70, 11/94. Added RXT11/RX01, RK02/RK05, RS03/RS04, TU56, TU10, TU60, TE16/TU45/TU77 device emulation. Definable keyboard. Loadable ROM/EEPROM. Many new floppy types, which may now be used with any disk controller type.

V2.1 Full version, 01-Apr-1999; runs in protected mode with full 22-bit MMU. MSCP, TMSCP, RMxx/RPxx, TS11, DZ11/DZV11, DHU11/DHV11, DEQNA device emulation. Support for Q/Unibus bridges. Boca, Chase PCI-FAST, Digi, RocketPort/PCI, SBMIDI serial drivers. NE2000 Ethernet driver. Physical port drivers for SCSI disk/tape drives, RAM disks/tapes. FLOATING address calculation, DEFAULT controller types. PDP-11/23, 53, 73, 83, 84, 93 CPU types added. Demo/Lite versions have a subset of these features.

NOTES

INTERRUPTS

The interrupt system is somewhat complicated, mainly due to some assumptions in DEC OSes (mainly RSX and RT-11 SJ) about how many instructions are guaranteed to be executed after writing a command to a device CSR, before the device will complete the operation and interrupt. Since MS-DOS doesn't support asynchronous I/O (unless you go to extremes which wouldn't have made sense in a CPU-bound program like an instruction set simulator), it's natural to have most emulated device I/O appear to the PDP-11 to be instantaneous (not really, the PC takes time between emulated PDP-11 instructions to do the transfer), with the completion interrupt occurring before the instruction following the one that started the transfer. Unfortunately this causes trouble with some drivers that assume that they are guaranteed to complete a certain number of instructions before the completion interrupt occurs. This is not actually a bug if it works on all real PDP-11 models, but it leads to incorrect operation if the emulated hardware appears to be fast enough to complete an operation before the expected minimum number of instructions is executed. In my testing, RSX appeared to issue `WAIT` instructions for TTY output which was assumed not to have completed yet a few dozen instructions after writing a character to a DL11 (thus hanging the system), and similarly the RT-11 SJ (but not FB/XM) keyboard interrupt service routine runs with interrupts enabled on the assumption that another keyboard interrupt couldn't possibly happen before the current ISR finishes (when this does happen the ISR recurses and the characters are put in the buffer in reverse order, which was happening with VT100 keypad keys in E11). The solution to these problems was to put in a queueing system, so that the interrupt (and in most cases the transfer itself) doesn't occur until a pre-set number of instruction fetches after the instruction that started the transfer. The default delays are intended to be adequate for most users; however when troubleshooting with custom operating systems, this is a good place to experiment if E11 appears to work with your application using certain emulated devices, but not others. So far no such trouble has been experienced with disks or tapes, so they are all set to execute all functions in one instruction time by default. However for reasons given above, the character-at-a-time devices have larger default delay counts. RK05/06/07 seek completion attentions may be delayed still further beyond acknowledging the seek command so as not to confuse overlapped seek drivers; however you'll get faster results using a nonoverlapped driver if one is supplied with your OS. Since all your emulated disks will typically be on one physical DOS disk with only one head carriage, there's nothing to overlap anyway.

KEYBOARD

The default keypad layout may take a little getting used to but it's intended to be familiar if your fingers are already comfortable using KED or EDT on a real VT100; just don't look at the keypad, the keys are where you expect in spite of having the wrong labels. The digits and "." key work as marked (must be in Num Lock mode to get this on 84-key keyboard, doesn't matter on 101- or 104-key keyboards). The keys around the top and right edges of the keypad are *not* as marked, but correspond to the PF1–PF4, hyphen, comma, and ENTER keys of the VT100. The comma key is missing unless you have an 84-key AT keyboard; Northgate *Omnikey 102* keyboards have an "=" key where the VT100 comma belongs but unfortunately there is no way for software to distinguish it from the =/+ key on the main keyboard so E11 can't use it as a comma, you'll have to just use F8. To get the normal function of the Num Lock key (and Esc, Scroll Lock, and Sys Req on an 84-key AT keyboard), press Alt, Ctrl, or Shift at the same time (it doesn't matter which). The keypad hyphen, comma, ENTER, and period keys are also available as the F6, F8, F10, and F9 keys. These assignments make some sense on an 84-key AT keyboard, so it might help to picture that. Backspace is CTRL/H, line feed is CTRL/J, the other tricky control codes are the same as on a real VT100 (e.g., NUL is CTRL/SPACE). If your keyboard has an F12 key, pressing it will send the current date and time in the format "*hh:mm:ss dd-mmm-yyyy*," unless you `DEFINE` it to do otherwise. This is intended to be useful when starting an RSX or IAS system, since they have no way to inherit the date and time from a boot program (although the current versions of RSX have a "`TIM /SYN`" command that can read the TOY clock in E11's 11/93 and 11/94 emulations).

DISK IMAGES

Getting a snapshot of a bootable disk from an existing PDP-11 into a DOS file is up to the user, D Bit will not be involved in pirating software so please don't ask for images. It appears that DEC, Mentec, and S&H are all now willing to sell PDP-11 OS licenses to emulator users, there doesn't seem to be an issue about the lack of a CPU serial number. Anyway disk images have been successfully loaded from real PDP-11s using Kermit, or Process Software TCP/IP, or DECnet and Pathworks, or an OS-supplied DUMP command on the PDP-11 with the output captured with a PC terminal program and then massaged back into binary with a small C program. Also, `PUTR` (available from `FTP.DBIT.COM`, see below) can build bootable RT-11 image files using a floppy disk distribution kit, without the need for booting a real PDP-11. The full version of Ersatz-11 can boot and install directly from a DEC/Mentec floppy or tape distribution kit, given the appropriate PC hardware.

Note that some operating systems do little or no autosizing and may have problems if the system being emulated by E11 differs from the one for which the OS was built. In particular you may run into trouble if your OS depends on any static memory allocation (if E11 is emulating a different amount of memory than the OS expects), or if the OS is built for Q22 I/O and E11 is emulating an 11/44 with UMRs, or anything like that. Also, the routine in RSX11M+ that counts the number of registers in an RH70 depends on PDP-11/70 autoincrement semantics, and will get the wrong answer if you set the CPU type to PDP-11/44. This normally causes no problems since real PDP-11/44s can't have RH70s but they can in E11.

`PUTR.COM`, a companion program to E11, is available by anonymous FTP from `FTP.DBIT.COM`, in the `pub/putr` directory, and knows how to read and write RT-11 and OS/8 format volumes on a variety of media, as well as how to read RSTS/E volumes. It can write blank container files with the serial numbers and (empty) bad block data filled in correctly, and format many types of DEC floppies, which can be useful with any OS. Assembly language source is included.

Paul Koning (former RSTS/E developer) has written a very complete program named "`flx`" for manipulating files in RSTS disk images. Among other things it can build a bootable disk given the files from [0,1]. It's available from `FTP.DBIT.COM` in `pub/flx` and is written in portable C, so it can be used with any emulator (or with real disk packs on a VAX). A DOS executable is included with the sources.

NOTE

> The RT-11 `DL:` and `DM:` device handlers expect to find a bad block replacement table in block 1 of a disk. If something else is there (like the pack label in Files-11 and RDS 1.1 and later, or the MFD in RDS 0.0), they will replace blocks at random and you'll get a corrupted disk image. So either modify your Kermit (etc.) to use the appropriate `.SPFUN` instead of `.READ`, or don't use RT-11 programs to read non-RT-11 disks.

HOST SYSTEMS

"Why didn't you write a version to run on {*machine*} under {*OS*}?" The program is written entirely in 80x86 assembly language (over 100,000 lines); porting the devices, debugger, interrupt system etc. to another architecture would be straightforward but the instruction set processor depends heavily on similarities between the 80x86 and the PDP-11 (most notably byte order and the condition flags; correcting the byte order and/or deriving condition flags "by hand" on another machine would be very inefficient). The FP11 simulation likewise relies heavily on the 80x87 data formats. Also the VT100 emulator gets pretty intimate with the video hardware. So while a port to another architecture would be possible, it already runs fine on the cheapest, most common hardware around under an OS that's cheap, easy to install, and requires very little memory, so porting doesn't seem worthwhile at this point.

Meanwhile, if what you want is a UNIX-based PDP-11 emulator in C, at least four have already been released, by Bob Supnik of DEC (J-11, in `pub/mbg/simulators` on `FTP.STD.COM`), Eric Edwards of RIT (11/34, `pub/csh/mag/pdp.tar.Z` on `FTP.CSH.RIT.EDU`), der Mouse of McGill (FP-less J-11? `pub/people/mouse/pdp11` on `FTP.CIM.MCGILL.CA`), and Begemot Willi (Harti Brandt and Joerg Micheel) of GMD (J-11, formerly in `pub/nthp/11` on `FTP.FOKUS.GMD.DE`, archived in `pub/PDP-11-sims` on `MINNIE.CS.ADFA.OZ.AU`, which also has archives of all of the above emulators). You'll need a faster machine to get the same results, and there are fewer emulated devices, but you do get portable source code.

SOURCE CODE

There are no plans to make source code to E11 publicly available.

COPYRIGHT NOTICE

ACKNOWLEDGMENTS

I would like to thank the many people who provided technical help and debugging input. Bob Supnik at DEC and Alan Sieving at QED provided valuable details of poorly documented PDP-11 instruction set semantics. Many people have helped debug Ersatz-11 with their configurations; Frank Borger's (U. Chicago) work with RT-11SJ and IAS has been particularly impressive, as have Paul Koning's (Xedia) insights into RSTS and Eduard Vopicka's (Prague University of Economics) help with RSX. Chip Charlot and Dave Carroll of Mentec have provided invaluable technical help and encouragement.

FEEDBACK

Continued feedback on the program would be greatly appreciated, whether it's good or bad. Please include as much information as possible when reporting bugs, `LOG` files for peripheral devices can be valuable troubleshooting aids. Suggestions for improvements are always welcome, as well as reports of how E11 behaves under operating systems that have not been tried previously.

AS ALWAYS, MAKE A BACKUP OF ANYTHING YOU'D REGRET LOSING BEFORE RUNNING THIS PROGRAM.

Digby's Bitpile, Inc. DBA D Bit
11 Bank Street
Troy, NY 12180
USA
+1 (518) 271-6824
+1 (518) 272-3853 FAX
`e11@dbit.com`
`http://www.dbit.com`