ERSATZ-11 for OS/2
PDP-11 EMULATOR
DEMO VERSION 6.0

FOR 30-DAY COMMERCIAL
EVALUATION ONLY

Release date: 01-Nov-2010

DBIT

D Bit • 139 Stafford Road • Monson, MA • 01057 • USA • www.dbit.com

The following are trademarks of Digby's Bitpile, Inc.:

D Bit   E11   Ersatz

The following are trademarks or registered trademarks of Digital Equipment Corporation:

DEC	DECnet	DECtape	DECwriter	DIGITAL
IAS	MASSBUS	PDP	PDT	P/OS
Q-BUS	RSTS	RSX	RT-11	ULTRIX
UNIBUS	VT			

The following are trademarks or registered trademarks of S&H Computer Systems, Inc.:

TSX   TSX-Plus

Other product, service, and company names that appear in this document are used for identification purposes only, and may be trademarks and/or service marks of their respective owners.

Contents

Chapter 1

Introduction

Ersatz-11 is a software replacement for PDP-11 minicomputer systems. When running on typical PC hardware, it substantially outperforms any PDP-11 model ever produced by DEC, outpaces all known aftermarket clone CPUs, and is the fastest PC-to-PDP-11 software emulator available at any price. Yet it is the most inexpensive PDP-11 replacement product on the market.

The achievable performance continues to increase as new IA32-compatible CPUs are released, so future upgrades are possible without requiring that a new PDP-11 CPU board be purchased. The emulated system is configured using simple commands, allowing the flexibility to duplicate almost any existing system easily. The configuration of the emulated system can be changed at any time, even while the system is running. Wherever possible, E11 provides useful defaults, to simplify the configuration process. It will choose between Q-bus and Unibus controller models depending on the emulated CPU type, and will auto-compute "floating" CSR and vector addresses, for devices that use them. It also chooses the default interrupt priority level according to the CPU type, since in many cases the Q-bus version of a peripheral interrupts on level 4 even though the original Unibus version uses level 5. In all cases these defaults can be overridden with SET commands.

Ersatz-11 emulates the entire PDP-11 system in software, including most standard disks, tapes, serial devices, and network interfaces. Facilities are provided to integrate non-standard device emulations into E11 when necessary. E11 is intended to boot and run any PDP-11 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), the Fox 2/30 OS, and XXDP+.

Emulated configuration

- PDP-11/03, PDP-11/04, PDP-11/05, PDP-11/20, PDP-11/23, PDP-11/24, PDP-11/34a, PDP-11/40, PDP-11/44, PDP-11/45, PDP-11/53, PDP-11/60, PDP-11/70, PDP-11/73, PDP-11/74, 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, FIS floating instruction set option, KE11A/B extended arithmetic element (EAE)

- 4 MB main memory

- Many different models of disks, tapes, serial and network devices (see tables below).

1.1   Emulated block device types

dev name	controller type(s)	drive type(s)
DC:	RC11	RS64 fixed-head disks
DD:	DL11	TU58 cartridge tapes
DF:/RF:	RF11, DMS-11X	RS11 fixed-head disks, DDC SSDM 100 RAM disks
DK:/RK:	RK11D	RK02, RK05 front-loading cartridge drives
DL:	RL11[1], RLV11, RLV12	RL01, RL02 top-loading cartridge drives
DM:	RK611[2]	RK06, RK07 top-loading cartridge drives
DO:	(virtual)	Ersatz-11 interface to host file system (for DO.SYS/DOS.TSK/LNX.TSK)
DP:	RP11C	RP02, RP03 pack drives
DS:	RH11, RH70	RS03/RS04 fixed-head disks
DT:	TC11	TU55/TU56 DECtape
DU:	UDA50, KDA50 RQDX3, RQZX1	RA60, RA70–73, RA80–82, RA90/92, RC25, RD31–32, RD50–54, RX33/RX50 MSCP disks
DX:	RX11, RXV11	RX01 8" SS SD floppy
DY:	RX211, RXV21	RX02 8" SS DD (or DS DD) floppy
FX:	Fox 2/30 DBI	DDC 6200 "drum" (actually a fixed-head disk)
HD:	(virtual)	Ersatz-11 hypothetical disk with simplified interface (for HD.SYS)
PD:	RXT11	RX01 8" SS SD floppy (PDT-11/150 microcomputer)

[1] "RL211" was used as a marketing name in systems with RL02 drives, but is the identical controller to the RL11.
[2] "RK711" was used as a marketing name in systems with RK07 drives, but is the identical controller to the RK611.

1.2 Emulated sequential device types

dev name	controller type(s)	drive type(s)
CT:	TA11	TU60 DECassette dual cassette tape
MM:	RH11/RH70 + TM03	TE16, TU45, TU77 Massbus magtape drives
MS:	TS11, TSU05/TSV05, M7454, TQK25	TS04, TS05, TU80 magtape drives, TK25 cartridge tape drive
MT:	TM11	TS03, TU10 magtape drives
MU:	KLESI, TQK50, TQK70	TU81, TK50, TK70 TMSCP tapes
PR:/PP:	PC11	PC04 high speed paper tape reader/punch

1.3 Emulated serial device types

dev name	port type(s)	description
LP:	LP11, LPV11	Line printer interfaces
TT:	DL11, DLV11	Single serial line units
YZ:	DZ11, DZQ11, DZV11	serial PIO multiplexers

1.4 Emulated network device types

dev name	port type(s)	description
NI:	NI1010A, NI2010A	Interlan Unibus/Q-bus Ethernet interfaces
XE:	DELUA, DEUNA	Unibus Ethernet interfaces
XH:	DELQA, DEQNA	Q-bus Ethernet interfaces

1.5   Miscellaneous device types

dev name	port type(s)	description
DO:	(virtual)	PC file interface
KD:	M8644 CDR	countdown registers
KP:	KW11P	programmable clock
VR:	VT30	color graphics display
VT:	VT11	vector graphics processor

1.6   PC hardware support

- Block devices: disk image files, RAM disks

- Tape devices: tape image files, RAM tapes

- Character devices: Emulated VT100 on SVGA (can flip between up to 12 virtual screens)

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. Older 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 from `www.dbit.com`.

The emulator speed depends on the application and the host system. In general E11 on any speed Pentium III, Pentium IV, or Athlon runs on the order of ten to twenty times the speed of a PDP-11/93. E11's disk I/O is much faster than that of real PDP-11s, especially when run under an operating system which provides good disk caching.

The FP11 floating point processor emulation currently requires a math coprocessor (which is built into all current 80x86 CPUs anyway). 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 FIS emulation does not require a math coprocessor.

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. It has been found that passing or failing DEC diagnostics does not bear much relation to actual operation with real-world software and operating systems, because the diagnostics are designed mainly to detect known failure modes of real DEC hardware, and not to verify new implementations.

1.7   Device names

With the exception of PC 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. Specified only with controllers that can support multiple devices. A reasonable default is used if it is omitted, generally the first or only controller of that type. The controller letter is always displayed in output from the SHOW command for devices where it is meaningful, so the name given by SHOW will have the letter even if you didn't specify one. In the demo version of E11, the RH11/RH70 Massbus controllers, TC11 DECtape controllers, and HD: pseudo disks are the only controller types of which there may be more than one. All other types will always use controller A so the default is always the only choice.
>
> 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 consistent naming system than these operating systems do, since the first two letters of a PDP-11 device name *always* depend on the controller type.

In some cases this leads to differences, for example "TTu:" (or "KBu:", which is a synonym) refers specifically to a DL11/DLV11 serial port, while in RSX and RSTS, *all* terminal ports are mapped to one of these names regardless of the port type. But in E11, a serial port located on a DZ11/DZV11 is always "YZcu:". 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:), LPT1:–LPT4: are parallel ports and (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 Adaptec ACB-4000A SCSI/MFM bridge boards. In any case if s is omitted (as well as the preceding underscore) a reasonable default is used.

1.8   Filenames

A few rules apply to PC 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 the E11 executable file is located, 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. Path elements must be separated by "\" characters, except when the file specification is enclosed in single or double quotation marks 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 default extensions are:

.ext	type of file	relevant command
.CMD	command file	@[d:$path$\]$filename$
.DSK	disk image file	MOUNT
.INI	init file	/INITFILE switch
.LOG	log file	LOG
.PAP	paper tape image file	MOUNT PR:/PP:
.PDP	binary memory image	LOAD, DUMP
.TAP	tape image file	MOUNT

1.9 Notes

1.9.1 Interrupts

The emulated interrupt system is somewhat complicated, mainly due to some assumptions in DEC OSes (particularly 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. On the PC, some I/O may complete much faster than on a real PDP-11, and it even appears to be instantaneous with RAMdisks/RAMtapes (and disk/tape image files under MS-DOS), since the I/O is performed synchronously between PDP-11 instructions. Unfortunately this causes trouble with some drivers that depend on being guaranteed the time to execute 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. Under 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 early versions of E11.

The solution to these problems is to use a queueing system, so that the interrupt (and in many 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. Much less trouble has been experienced with disks and tapes, so by default most of them are set to execute all functions in one instruction time. The RSX MSCP initialization sequence is an exception, so the default delays are tuned appropriately. The RK11 handler in DOS/BATCH requires an unusually slow disk controller, so in order to use that you must first issue a command like `SET DELAY RK11D *=8000` to make all RK11D disk commands take 8000 instruction fetches to complete. 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 non-overlapped driver if one is supplied with your OS. Since all your emulated disks will typically be on one physical PC disk with only one head carriage, there's nothing to overlap anyway.

1.9.2 Host systems

D Bit occasionally receives inquiries from users who want Ersatz-11 to be ported to architectures other than the 80x86. There are no plans to port E11 to other processors. Besides the expense and difficulty of moving software between radically different host systems, D Bit is considers its ability to provide adequate customer support to be of primary importance, and this would not be possible if there were too many different versions of E11.

Rather than produce poorly supported versions of E11 for a myriad of host systems which provide more hindrance than help to the task of emulating a PDP-11 system, D Bit has chosen to focus its efforts on the 80x86 architecture under Linux, OS/2, Win32, and DOS. This hardware has the best price:performance ratio of anything currently available, it's what the vast majority of E11 customers are already running anyway, and its programming architecture lends itself well to efficient PDP-11 emulation. The supported operating systems are inexpensive and easy to install (some more than others), and they provide a good set of helpful services to E11 but also allow easy access to hardware so that Ersatz-11 has the control it needs, adequate DOS emulation is available in a variety of other operating systems, so that again, most users already have a system which is capable of running E11.

1.9.3   Copyright and licensing

1.10   Acknowledgments

D Bit would like to thank the many people who provided technical help and debugging input. Bob Supnik, formerly of DEC, and Alan Sieving of 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 (EqualLogic) insights into RSTS and Eduard Vopicka's (Prague University of Economics) and John Shilling's (JSA) help with RSX. The late Chip Charlot (formerly of Mentec), and Dave Carroll of Mentec have provided invaluable technical help and encouragement. Johnny Billquist was very helpful with getting the PDP-11/74 emulation working with RSX, and in debugging the DELUA emulation and the PSW T-bit.

1.11   History

31-Oct-1993; development started.

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.

(Many intermediate V2.0x full versions were released throughout 1997 and 1998.)

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.

V2.1A, 01-Oct-1999; concatenated image files to form one large disk, CDROMx: driver, other minor improvements and bug fixes.

V2.2, 01-Apr-2000; DH11, DM11BB, DR11C, VT11, Interlan NI1010A/NI2010A emulation, FIS instruction set, KE11A/B EAE, mini-assembler. Hardware drivers added or extended for DCI-1300 digital I/O boards, SVGA graphics, ISA RocketPorts, BCI-2004/BCI-2104 bus adapters, multiple physical serial devices. SCSI disk partitioning added. Most disk "write headers" commands perform low-level format. Documentation overhauled.

V3.0, 01-Oct-2000; Linux version released. DMS-11X RAM disk, SET THROTTLE, Data Products printer support, PCI LPT cards, configurable interrupt priorities. Additional SET CPU options to support early CPU models.

V3.1, 01-Jul-2002; Telnet server, TU58 cartridge tapes. Demo version for Linux, expanded limits on Demo/Lite versions.

V4.0, 01-Apr-2004; flat 32-bit version. DJ11, M8644 countdown register emulation, command line editing/recall. DOS version adds drivers for RocketPort "Universal" PCI boards, RealTek RTL8139 Ethernet ports, and autosizing in CDROM: (so DVDs work too). Win32-style "PE" .EXE format DLLs supported.

V4.0A, 01-Jun-2004; bug fixes for new flat version. Experimental Catweasel floppy driver. Win32-style DLL support improved and documented.

V4.0B, 15-Oct-2004; bug fixes. SET MEMORY, SET CPU USERMODE, SET CPU 03.

V4.1, 01-Sep-2005; bug fixes. SET CPU PRIO. DR11W, KW11P, and VT30 emulation. Drivers for LPT port as digital I/O device, and PCI 11W DMA board. THROTTLE: pseudo-driver. SET TELNET: BANNER=*file*. Support for extended CON: screen sizes, SHOW VIDEO. Timer code rewritten for greater precision, with Linux version using /dev/rtc if available.

V5.0, 01-Feb-2006; Win32 native version. CPU speedups. SET CPU 74, SET CPU 60, SET CPUS=*n*. SET TELNET: TERMINAL=*type*. LOG /NOTIMESTAMPS. ASSIGN . . . /CD:*x* /CTS:*x* /DSR:*x* /RI:*x* /SRD:*x* (and SET equivalents). Up to 8 LP11s in full version. Many bug fixes.

V5.1, 01-Aug-2007; disk and tape caching. Raw BIOS hard disks and partitions. Driver for DCI-4100 digital I/O boards, many improvements to digital I/O board emulations and drivers. Bit 3 Q-bus adapters. Newer Q-/Unibus adapters from The Logical Company. SHOW PCI.

V5.2, 01-Aug-2008; mouse cut/paste. SET/SHOW PCLOG. SET CPU NOMMU speedups. ADDPCI. SCAN. Delays after bus adapter cycles. Emulated DMA to I/O page. New mP features for PDP-11/45s: FASTBUS:, IPL:, SET CPU NOSHAREDMEM.

V5.3, 01-Sep-2009; ATTACH BCI: and ASSIGN . . . TELNET: in Win32 version. 64-bit memory addressing in DOS version. DIR, CD, TYPE commands. FAT file system driver in stand-alone version. Ethernet tunneling over UDP in Linux and Win32 versions.

V6.0, 01-Nov-2010; OS/2 native version. TCP/IP stack and drivers for NICs based on PCnet-PCI, Tulip, RTL8139, and Velocity chips in DOS and stand-alone versions. Ethernet booting. DELQA emulation (was DEQNA). TELSINGLE: and TELCLIENT:. LOG /NOBUFFERING and LOG /LAST:*n*. OpenGL graphics driver and new installer for Windows (and OS/2).

Chapter 2

Installation and Configuration

2.1 System requirements

Installation of Ersatz-11 requires the following:

- Intel 80x86-compatible PC with 80386 or later CPU

- At least 6 MB of available system memory (any additional memory may be used for RAM disks/tapes and disk/tape caches).

- 2 MB of available disk space for Ersatz-11 itself, plus space for all disk image files (equivalent to the size of the disk drives they replace).

- IBM OS/2 2.0 or later, or any version of eComStation.

2.2 Installation

Installation is very straightforward. Simply download the installation program, open a command prompt window, and type:

`C:\>install`

The installation program will prompt for a directory into which to install, and will then extract all the E11 files from the distribution archive. The files are as follows:

file	contents
E11.EXE	executable
E11.HLP	"HELP" data file
E11DEMO.PDF	this document, readable with Adobe Acrobat
E11FONTS.FON	fonts for E11terminal sessions
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
XHBOOT.BIN	D Bit implementation of DELQA boot/diag ROM

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.

2.3   Configuration

Ersatz-11 is configured using a text file named "E11.INI," which is normally kept in E11's home directory. This file may be created using any text editor. It contains a series of commands which are read and processed in order every time E11 starts up. Lines which start with a ";" or "!" character are treated as comments, and ignored. Each individual serial line or emulated disk or tape unit is created with a one-line command. Typically there will also be additional commands to define the emulated CPU model, set any non-standard device addresses or device types, and finally the initialization file usually ends with a BOOT command which boots the emulated PDP-11's operating system. The table below summarizes which command is used to add each device type to the system. The syntax of each specific command is described in chapter 8.

device type	command to create
CPU	SET CPU
disk drive	MOUNT
tape transport	MOUNT
TTY line	ASSIGN
line printer	ASSIGN
network port	ASSIGN

Below is a typical E11.INI file. This file is read by Ersatz-11 every time it starts up, and the commands are executed in sequence, as if they were typed at the keyboard.

```
;
; Set PDP-11/44 CPU model
;
set cpu 44
;
; Mount disk and tape units
;
mount du0: rsx11m.dsk
mount du1: ra81.dsk
mount mm0: dump.tap
;
; Add extra DL11 terminal lines and LP11 printer port
;
```

```
assign tt1: con2:
assign tt2: con3:
assign lp0: con4:
;
; Boot MSCP disk unit 0 (and switch keyboard to PDP-11 console)
;
boot du0:
;
; Control returns to the next line when the user presses Shift-Enter
; or the system halts on its own (e.g. SHUTUP.TSK)
;
quit
```

The "boot" command starts the PDP-11 operating system and directs keyboard input to the PDP-11 system console. The user can press Shift-Enter at any time to pop up an E11 command prompt where additional commands may be entered interactively. If there are unread lines remaining in "E11.INI" (i.e. following the "boot" command), they will be read at that time. In this example, the program will exit immediately when Shift-Enter is pressed because it causes the "quit" command to be read.

There are a few more basic options which are entered as switches on the E11 startup command line, rather than being contained in the "E11.INI" initialization file:

/CONSIZE:*WxH*	Set the default size for all CON: screens, including the one implicitly ASSIGNed to TT0: at startup.
/HELP *(syn. /?)*	Display a simple list of switches and their meanings, and exit without starting E11.
/INITFILE:*file*[.INI]	Read the specified initialization command file instead of E11.INI.
/IOPAGE:*nnnn*	Enlarge the I/O page past the usual 8 KB size, with *nnnn* specifying the decimal number of kilobytes. This is mainly useful with customer-written loadable device emulation modules which act like RAM windows, and is equivalent to the "Unibus fence" feature of some PDP-11s that have 22-bit memory addressing.
/MEMORY:*nnnn*	Set the maximum possible emulated memory size of the PDP-11 to *nnnn* (decimal) kilobytes. By default this maximum is 4088 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 8 KB. This switch allows you to conserve memory when emulating a PDP-11 with less memory than E11's default.
/NOINITFILE	Do not process the E11.INI initialization command file.

When configuring the system for the first time, it is best to exactly duplicate the system which is being replaced. There may be a strong temptation to expand the system, now that PDP-11 peripherals are effectively "free" for the asking. But such changes can cause conflicts with existing software, which may contain hard-coded device names or other hidden assumptions about the system configuration. These problems may be avoidable, if system expansion is postponed until after the existing system is brought up as-is.

Note that some operating systems do little or no autosizing and may have problems if the hardware being emulated by E11 differs from the one for which the operating system was generated. 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 what the system expects), or if it is built for Q22 I/O and E11 is emulating a PDP-11/44 with Unibus map registers, or anything like that. Also, the routine in RSX-11M-PLUS 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 this and other "impossible" situations can be easily created in E11.

One thing to watch out for, is that some software has hard-coded assumptions about how fast the hardware operates in relation to the CPU. See section 1.9.1 for a discussion of interrupt timing. This can lead to strange behavior, such as devices that simply hang, or supposedly I/O bound tasks that consume 100% of the CPU. These problems can generally be solved by experimenting on the relevant emulated devices with the `SET DELAY` command. If the system doesn't operate correctly with the default delay counts, but springs to life when they're increased to large numbers, then it's just a simple matter of tuning the numbers to get delays which are long enough that the system works reliably, but still give good performance. Ideally the PDP-11 device drivers should be updated so that they will operate correctly with "infinitely" fast hardware, since this will allow it to get the best possible performance out of *any* fast PDP-11 replacement, but the `SET DELAY` command allows the user to work around these problems without having to touch the PDP-11 software.

Chapter 3

Transferring the PDP-11 Operating System to the PC

In order to run an exact copy of a real PDP-11 system, the contents of the PDP-11's disk(s) must be accessible to Ersatz-11 to be used as emulated disks. In most cases, this means importing a byte-by-byte copy of each entire disk into a large "image" file which is the same size as the entire PDP-11 disk. Note that this is very different from importing the individual *files* from the PDP-11 disk. In some cases the files can be reassembled into a disk image, but if possible it's better to download the disk(s) as a single large image. The fewer transformations the data go through, the fewer opportunities for mistakes that could lead to file damage.

Getting a snapshot of a bootable disk from an existing PDP-11 into a PC file can be tricky, there are many ways to do it and which choice is the best one depends on what software and hardware are available, and what media or protocols the PDP-11 and PC have in common. D Bit can help with some forms of media translation, call or send email for information. 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. So ordering the latest OS version is straightforward, and with the right peripherals the installation kits can be booted directly by E11 and installed on emulated disk(s).

Disk images have been successfully loaded from real PDP-11s using Kermit, or Process Software TCP/IP, or DECnet and Pathworks, or (as a last resort) 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 simple utility 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.

3.1 SCSI disks

The easiest way of all to move any PDP-11 OS to the PC is using a SCSI disk drive. If you were already using a PDP-11 SCSI controller, you should be able to move the disk directly over to a PC SCSI controller, and either read it into an image file (using PUTR.COM) or use it directly (using the SCSI disk support in the full version of E11). Iomega Zip and Jaz drives, and Fujitsu DynaMO 3.5" magneto-optical drives, are inexpensive and work very well with both Ersatz-11 and PUTR.COM, and they provide a cheap, fast, easy way to transfer files or whole disks between real PDP-11s and E11.

Even if the PDP-11 didn't already have a SCSI controller, it might be worth obtaining a used one if a non-trivial

amount of data must be transferred, unless the PDP-11 operating system is too old to support SCSI MSCP devices. Adding support for it to the operating system might require a SYSGEN in some cases, but for RSX, BRUSYS will support a DU: controller regardless of whether your usual monitor is built with the DU: driver. Older versions of RSTS can both backup and restore without ever booting a monitor at all, and the backups are self-booting and include a copy of the backup utility, so restoring them on the target system is easy. New versions of RSTS require that the backups be made under timesharing, but they can still be restored in INIT.SYS.

3.2   SCSI tapes

Magnetic tape can be a convenient way to import PDP-11 data to the PC, as long as a PC tape drive can be found which will read the PDP-11 media. The DEC TZ30 and TK50Z-GA SCSI drives use TK50 tapes, and a variety of companies still make SCSI 9-track tape drives. These may be mounted inside E11 as PDP-11 tape drives and used directly by the emulated PDP-11, or images may be taken using D Bit's DOS SCSI tape utility (available from ftp://ftp.dbit.com/pub/ibmpc/util/st.exe) and the resulting `.TAP` files may be mounted under E11.

3.3   Kermit

Kermit is a protocol for transferring files over serial lines. It provides very good reliability, and requires no special hardware, but it can be slow. At 9600 baud with the default Kermit protocol parameters (no long packets or sliding windows), binary file transfers can take almost an hour per megabyte. This may be acceptable if the system has small disks or the transfer can be performed over a weekend or during some other time when the PDP-11 is not busy with other work. Kermit protocol support is included in most terminal programs, and Kermit software is available at little or no cost for most computers and operating systems, including the following PDP-11 versions:

- KSERVE — D Bit's server-only Kermit for RT-11, available from
  `http://www.dbit.com/pub/pdp11/rt11/kserve.mac`

- K11 — Columbia's official PDP-11 Kermit program, supports all major DEC/Mentec/S&H operating systems, see `http://www.columbia.edu/kermit/pdp11.html`

- KRT — Billy Youdelman's version of K11, enhanced for RT-11 and TSX-Plus, also at
  `http://www.columbia.edu/kermit/pdp11.html`

It's very important to issue SET FILE TYPE BINARY commands to the Kermit programs at both ends before beginning a Kermit transfer of a PDP-11 disk. The other settable Kermit parameters are less critical, as they mainly affect the speed with which the transfer will proceed. Of the above Kermits, only KSERVE is able to download an entire raw disk. The others can download files but not entire raw disks. In some cases it may be possible to get around this limitation by using PDP-11 system utilites (such as RT-11's COPY/FILE/DEV command) to copy an entire raw disk into a file on another, larger disk, and then use Kermit to download that file. Or, if no larger disk is available, it may be possible to break the disk into several pieces (each small enough to fit into the free space on another disk) and copy them to files one at a time, sending each to the PC via Kermit and then deleting it.

3.4   Ethernet

Ethernet can be a very effective way to transfer PDP-11 data to a PC. Unfortunately Ethernet hardware is not nearly as common on PDP-11s as it is on PCs, but used Q-bus/Unibus Ethernet boards are available very cheaply

from used equipment dealers. So if the PDP-11 has software support for an Ethernet board, it may be worth buying one just to transfer the disks.

The main problem with using Ethernet on PDP-11s is the availability of compatible protocols. If a PDP-11 operating system supports Ethernet, it usually uses DECnet, while PCs are more likely to support TCP/IP. However software to support either protocol is available on both PCs and PDP-11s, it's just a matter of getting it.

- Alan Baldwin's excellent free TCP/IP packages for RT-11 and TSX-Plus are available via FTP from `shop-pdp.kent.edu`.

- Megan Gentry's "RTEFTP" RT-11 Ethernet file transfer program is available from `ftp://ftp.std.com/pub/mbg/pdp11/rt11/tools/rteftp/`. It uses its own private protocol and can communicate only with other computers running RTEFTP, however it's fast, very easy to set up, and can transfer raw disks.

- JSA Stackware, a commercial TCP/IP package for RSX, is available from JSA. It provides Telnet and FTP access in both directions.

- Process Software's TCP/IP package for the PDP-11 is also a commercial product, available from Process Software (`www.process.com`).

- DECnet/DOS and Pathworks for DOS are no longer available, but they were able to connect to DECnet hosts. So if you already have one of these packages, it may be useful for transferring disks.

Some of the above programs are not able to transfer entire raw disks over the Ethernet. In this case the same workarounds as used with Kermit transfers may be applied, to copy raw disks to files and then transfer the files.

3.5 ASCII serial dump

Most DEC operating systems provide a utility (typically named DUMP or DMP) which can dump a file or device out to the terminal in octal or hexadecimal. If this program is used to dump out the entire disk, a PC terminal program can be used to capture the output, and it can be translated back to a binary disk image on the PC. This should only be used as a last resort because there's no error checking or correction, and the transfer is very slow because of the inefficient encoding and extra header/trailer/address information that's normally displayed with each block.

If possible the transfer should be done using hexadecimal rather than octal, the transfer will go faster since each word is displayed as only four characters instead of six. On an RSX system, this can be done with the following commands:

```
>INS $DMP
>DMO ddu:/DEV/LOCK=V (if ddu: is the system disk)
>MOU ddu:/FOR (if ddu: is not the system disk)
>DMP TI:=ddu:/BL:0/WD
```

The file produced by capturing the output from the above command can be translated back into a binary disk image using the "HEX2DSK" program, available from `ftp://ftp.dbit.com/pub/ibmpc/util/`. Source code is included so the program can be modified for other dump display formats if needed. RT-11 has a DUMP/TERMINAL command, which is similar to RSX's DMP command but gives a different output format.

Since there is no error detection at all, disks transferred in this manner should be downloaded twice, and the resulting files should be compared, as a test to make sure the dump wasn't corrupted by line errors or buffer overruns.

3.6   System-specific notes

RT-11 is the easiest operating system to transplant. For one thing, an RT-11 Kermit server is available (at ftp://ftp.dbit.com/pub/pdp11/rt11/kserve.mac) which is able to download an entire raw disk image over a serial line. This can take a long time but it just about guarantees that the configuration will be duplicated exactly, just don't forget to type SET FILE TYPE BINARY and REMOTE SET FILE TYPE BINARY to make sure both participating Kermits agree not to try to treat the binary data as text. Failing that, you can use KSERVE, or any of several other file transfer methods, to download the individual files (still in binary mode) into a PC directory. Then use D Bit's "PUTR" program (available at ftp://ftp.dbit.com/pub/putr/) to build a blank RT-11 disk image, copy the files into it (be sure to use PUTR's COPY/B command to copy in binary mode), and make the disk bootable with PUTR's BOOT command.

<div align="center">Note</div>

> 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 master file directory 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.

RSTS images can be assembled from individual files using Paul Koning's freely available "flx" program, see below. All you need to build a bootable pack are the files from SY:[0,1], flx knows how to do the HOOK operation and make the pack bootable.

RSX is more difficult to move since currently there is no DOS software that knows how to build a bootable disk from its component files. If you don't have an easy way to take a raw snapshot of the whole disk (K11.TSK currently can't do it), the best bet is to dump the disk to tape and then restore the tape on the PC, assuming you have suitable tape drives on both. Stand-alone BRU (a.k.a. BRUSYS) is the easiest way to do this. After copying the entire disk to tape, build a bootable BRUSYS tape for E11 (HELP BRU STAND under RSX will tell you how). Then use PUTR.COM to build a blank disk image file of the appropriate type, and boot E11 from the BRUSYS tape, with the empty disk image file mounted. Once BRUSYS has started, you can physically switch tapes, or else pop up an E11 prompt and MOUNT the (first) backup tape if you're using tape image files. Then run FMT (if needed) and BAD on the blank disk, and finally run BRU to restore the tape. If you want to use tape image files instead of using a real tape drive (required on E11 Lite, since it doesn't support SCSI tapes), you can use D Bit's DOS SCSI tape utility (available from ftp://ftp.dbit.com/pub/ibmpc/util/st.exe) to copy between real tapes and E11 tape image files.

3.7   Utilities

PUTR.COM, a companion program to E11 is available from `ftp://ftp.dbit.com/pub/putr/`, and knows how to read and write RT-11 and RSTS/E and DOS/BATCH volumes on a variety of media, as well as how to read RSX 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, and SCSI disks too, which can be useful with any OS. It can also read and write TU58 DECtape II tapes, if the drive is connected to a PC COM port. 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://ftp.dbit.com/pub/flx/` and is written in portable C, so it can be used with any emulator (or with real PDP-11 disk packs on a VAX). A DOS executable is included with the sources.

Chapter 4

Disk Devices

Ersatz-11 emulates a wide variety of disk drive and controller models. These are created using one `MOUNT` command (see page 57) for each drive unit needed. Generally each emulated disk unit must be connected to an emulated disk device (e.g. a large file on the PC's disk) which is at least as large as the disk it emulates. The connection is device-independent, so any emulated disk can be connected to any of the physical disk device types that E11 supports. The controller itself is created implicitly when its first drive is mounted, and can be deleted by dismounting all of its drives. The controller type (RL11, UDA50 etc.) is implied by the device name used in the `MOUNT` command, and the same device name syntax is the same as most DEC operating systems.

E11 has sensible defaults for all disk parameters, so in most cases a `MOUNT` command for each unit is all that is needed. If necessary, the default drive parameters can be overridden using `MOUNT` switches, and the default controller parameters can be overridden using a separate `SET` command, which may be given before or after the drives are mounted. The defaults are as follows:

parameter	default	how to override
controller model	based on `SET CPU QBUS` setting	`SET` *ddc*`:` *model* (e.g. `SET DUA: RQDX3`)
controller CSR/vector	DEC default addresses "floating" addresses are calculated automatically if appropriate	`SET` *ddc*`: CSR=`*xxxxxx* `VECTOR=`*yyy*
interrupt priority	DEC default values, `SET CPU QBUS` setting	`SET` *ddc*`: PRIORITY=`*n*
drive model	based on size of container file	MSCP/TMSCP drives: `/TYPE:`*type* switch (e.g. `/TYPE:RD54`) non-MSCP/TMSCP drives: `/`*type* switch (e.g. `/RK06`)
write protection	disabled	`/WP` or `/RO` switch

After a drive is mounted, a `SHOW` *ddcu*`:` command (e.g. `SHOW DUA0:`) will display the actual drive parameters for that unit, as well as the controller parameters for the controller to which it is attached. The controller CSR, vector, interrupt priority, and type (i.e. controller model) can be changed explicitly at any time with a `SET` command. If these parameters are defaulted, their actual values can also be changed implicitly at any time, by a change in the `SET CPU` setting (which can make a defaulted controller type switch between the Unibus and Q-bus models of that

controller), and by changes in the configuration of other devices that use "floating" CSRs and/or vectors (if this controller is set to be a floating device, as `DUB:` and `DYB:` normally are for example).

4.1   Disk cache

Ersatz-11 provides a built-in disk cache, which can be enabled individually for each emulated disk unit by adding a "/CACHE" or "/CACHE:n" switch to the `MOUNT` command that creates that unit. n is the amount of memory (in bytes unless an optional suffix of `KB`, `MB`, or `GB` is added) to be set aside for the cache. This number includes all variably-sized data structures, so the actual amount of data that can be held in the cache will be lower. The default is 16 MB.

The built-in cache is intended mainly for the DOS version of E11, since the older DOS disk caches don't always scale well to the available RAM and hard disk capacity on modern PCs. On Linux and Windows systems the host operating system already provides good disk caching, so there may be no advantage to using the built-in cache. Some experimentation might be worthwhile though, since using E11's cache in addition to the OS-supplied one may reduce system call overhead enough to be noticeable.

The cache is currently write-through, i.e. it does not use delayed writes. Future versions may introduce write-back caching, with a settable delay.

4.2   PC disk devices

The `MOUNT` command establishes a connection between an emulated PDP-11 disk drive unit, and a PC file or device which will be used to hold the actual disk data. Generally this file or device must be at least as large as the PDP-11 disk unit that is being emulated. E11 supports several different forms of physical media to be used for emulating disks. The connections are made on a per-unit basis so it is possible to mix units attached to different types of physical media within the same emulated PDP-11 disk controller. The supported types of physical media are described below, along with the syntax of the `MOUNT` command for each type. The `/RONLY` (syn. `/WPROTECT`) switch may be specified on any `MOUNT` command, to lock the disk against writes.

4.2.1   Disk image files

Command syntax:

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

Special switches: none

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 `PUTR.COM` or `FLX.EXE` or `RT11.EXE` or a similar utility (see chapter 3). The file is the same size as the total capacity of the PDP-11 disk drive it replaces. The file is located using E11's usual search rules, see section 1.8 for details.

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. 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 (such as the COPFLP.MAC program available from ftp.dbit-.com), 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 guesses whether a file is a block or a sector image based on 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, since the RX50 soft interleave scheme uses all sectors of the disk. So by default they are assumed to be block images. The defaults may be overridden with the "/BLOCK" and "/SECTOR" switches on the MOUNT command.

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 emulations relocate 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, in spite of being the size of a sector image. In this case it may be more natural to use a sector image, but E11 supports either scheme.

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.

<u>Note</u>

The Demo version of E11 limit the combined size of all image files to 256 MB. E11 versions prior to 2.0 did not have this restriction, but they are no longer supported by D Bit.

4.2.2   RAM disk drives

Command syntax:

MOUNT *ddcu*: RAM: [*switches*]

Special switches: /LOAD:[*d*:*path*\]*filename*[.DSK], /SIZE:*blocks*

RAM disks may be created using regular PC memory. This gives extremely fast access to disk data, but has the disadvantage that all data are lost on shutdown. So any needed files should be copied to permanent storage before shutting down E11, and data may be permanently lost in the event of a power failure. The maximum possible size of RAM disks depends on the amount of memory available, so the SHOW MEMORY command may be useful in deciding how much space to allocate. Note that the actual amount of memory needed to create a RAM disk is slightly more than the disk's raw capacity, since E11 needs to maintain internal data structures related to the RAM disk and the emulated PDP-11 controller to which it is attached.

If no MOUNT switches are given, a 1 MB RAM disk is created. This size may be changed using the "/SIZE:*n*" switch, which sets the size of the disk to *n* blocks. Other units besides blocks may be used by appending the unit to the number (with no space in between), for example "/SIZE:20MB" sets the size to 20 megabytes. The RAM disk will initially contain all zeros unless it is loaded from a disk file, which is done with the "/LOAD:[*d*:*path*\]*filename*[.DSK]" switch. The file is located using E11's usual rules, see section 1.8 for details. If this switch is given, the RAM disk will be the same size as the file, unless the size is explicitly set with the "/SIZE" switch. The RAM disk is loaded from the file once when the MOUNT command is issued, then the file is closed. Any data written to the RAM disk do not change the file from which it was loaded. Note that if the RAM disk is not loaded from a file, it will normally need to be initialized by the PDP-11 operating system after creation, in order to have a usable file system.

4.3   Emulated PDP-11 disk devices

This section describes each PDP-11 disk drive type that Ersatz-11 emulates, and defines the device-specific `MOUNT` command switches that apply to each emulated disk type.

name	units	controller	drive/volume switches
`DC:`	0–3	RC11	(none, always RS64)
`DD:`	0–1	DL11	(none, always TU58 DECtape II)
`DF:`	0–7	RF11	(none, based on controller type)
(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`, geometry switches
`DS:`	0–7	RH11/RH70	`/RS03`, `/RS04`
`DT:`	0–7	TC11	(none, always TU55/TU56 DECtape)
`DU:`	0–65535	MSCP	`/TYPE:`*xxxyy*
`DX:`	0–1	RX11	(none, always RX01)
`DY:`	0–1	RX211	`/RX01`, `/RX02`, `/SS`, `/DS`
`FX:`	0	Fox 2/30 DBI	(none, file size is all that matters)
`HD:`	0–15	virtual	(none, file size is all that matters)
`PD:`	0–1	RXT11	(none, always RX01)

The "geometry switches" for DP: refer to `/CYLINDERS:`n, `/HEADS:`n, and `/SECTORS:`n, which may be used to specify non-standard drive dimensions. This allows compatibility with some clone controllers, which allowed extended drive sizes and/or untranslated SMD drive geometry to be used instead of the standard DEC geometry. In some cases the PDP-11 drivers may have been patched to use the non-standard geometry. These switches allow duplicating that hardware configuration so that the patched drivers will run unchanged under E11.

4.3.1   DC: — RC11/RS64

This emulation is included for completeness, it is not expected that anyone has a practical use for it. It is an early drive with a fixed platter and fixed heads yielding very low capacity (128 KB). There are few if any surviving units today, and no current PDP-11 operating system version is known to support it. Since there is only one possible drive type, no switches are required to select it.

`MOUNT DC:` drive switches:	
`/WPROTECT`	enable write protection (syn. `/RONLY`)

`SET DC:` controller parameters:	
`CSR=`*nnnnnn*	set CSR address to *nnnnnn* octal
`PRIORITY=`*n*	set interrupt priority to *n* (4–7)
`PRIORITY=DEFAULT`	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
`VECTOR=`*nnn*	set vector address to *nnn* octal

4.3.2   DD: — DL11/TU58

The TU58 is a small one- or two-transport cartridge tape drive. It is attached to the PDP-11 using a serial port, and communicates using RSP (Radial Serial Protocol), or MRSP (Modified RSP) in later drives. It uses cartridges which are physically similar to DC2000s, but hold only 256 KB each using DEC's proprietary format. DEC introduced the TU58 because they wanted to have a low-end storage peripheral which was cheap enough (well, about $1800, cheap by DEC's standards, and at least it didn't require its own controller card) that they could expect customers all to buy one. Then DEC would be able to get away with shipping diagnostics, microcode, and small software updates on just one medium, instead of having to make everything available on all types of media in use at the time. TU58s were plentiful, but deathly slow and low in capacity. DEC called the drives "DECtape II" in the hope that the name would remind customers of the earlier 555/TU55/TU56 DECtape drives, which were very popular.

TU58 drives are normally attached to a DL11/DLV11 serial line unit, and this is what DEC's OS-supplied drivers expect. However there's no reason why a TU58 can't be attached to any type of serial line, so E11 allows emulated TU58s to be mounted on any of its emulated serial ports. This means that creating a TU58 involves two or three steps: an `ASSIGN` command to attach the `DD:` device to a serial line, and one or two `MOUNT` commands to attach block devices to TU58 units 0 and/or 1.

For example:

```
ASSIGN TT1: DDA:
MOUNT DDA0: UNIT0.DSK
MOUNT DDA1: UNIT1.DSK
```

This would attach the first TU58 controller (`DDA:`) to a DL11 port (`TT1:` appears by default at the CSR and vector addresses expected by DEC's TU58 drivers), and then attach files to each of the two transports on that TU58. The `ASSIGN` and `MOUNT` commands may be given in any order.

Note that the command to boot from this combination is `BOOT TT1:`, *not* `BOOT DDA0:`, since `TT1:` is the device whose registers must be controlled to make the drive boot. The `BOOT` command supports drive 0 only.

MOUNT DD: drive switches:	
`/WPROTECT`	enable write protection (syn. `/RONLY`)

SET DD: controller parameters:	
`NEW`	new firmware (MRSP with bidirectional flow control)
`OLD`	old firmware (RSP, flow control from PDP-11 to TU58 only)

4.3.3   DF: — RF11/RS11, DDC DMS-11X/SSDM 100 (called RF: in RT-11)

The RS11 is another fixed-head disk, and is a PDP-11 version of the RS08 disk for the PDP-8. It was word-addressable and more popular than the RS64, as it was larger (512 KB) and more reliable. Due to their high speed for the time, RS11s were commonly used for swapping, so it may make sense to use a RAM disk to emulate an RS11. E11 allows write protecting the entire unit (with the `/WPROT` or `/RO` switch), but it does not emulate the RS11's switch panel which allowed write-protecting the disk in individual 32 KB segments.

The Digital Development Corporation DMS-11X controller is an extended replacement for the RF11/RS11, which uses a DDC "SSDM 100" RAM drive to hold up to 8 MB of data. It uses slightly different disk addressing from the RF11, and uses the unit select bits as additional track address bits, so only DF0: is useful. Non-zero unit numbers

will not be accessible when emulating a DMS-11X controller. A full 8 MB RAM disk may be created as follows:

```
SET DF: DMS11X
MOUNT DF0: RAM: /SIZE:8MB
```

MOUNT DF: drive switches:	
/WPROTECT	enable write protection (syn. /RONLY)

SET DF: controller parameters:	
CSR=$nnnnnn$	set CSR address to $nnnnnn$ octal
DMS11X	set controller type to DDC DMS-11X
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
RF11	set controller type to RF11
VECTOR=nnn	set vector address to nnn octal

4.3.4   DK: — RK02, RK03, RK05 DECpack (called RK: in RT-11)

These front-loading cartridge drives were very popular in the 1970s thanks to their low cost and small size (a 10.5"
rack-mount box). The disks are similar to the IBM 2315, and many other minicomputer manufacturers used a
similar form factor. The RK02 is a rebadged Diablo model 30 (low density) drive, and holds 1.2 MB. The RK03
is a Diablo model 30 (high density) which holds 2.5 MB. The RK05 is a DEC-built media-compatible replacement
for the RK03 which also holds 2.5 MB. It changes the disk-select method to support a maximum of eight RK05
drives per controller instead of just four RK03s (but can be configured to use the old scheme for compatibility with
RK11C and RK8E controllers). DEC also built the RK05J, which is a later model of the regular RK05, and the
RK05F, which had a fixed platter with twice as many tracks and acts like two RK05 drives. All of these drives
turn at 1500 RPM.

E11 does not distinguish between the RK03 and RK05 because they have identical geometries. The RK02 is no
longer supported by current versions of PDP-11 operating systems. It is unusual in that it stores 256 bytes per
sector, vs. 512 for the RK03 and RK05. Be careful when using odd image file sizes with the DK: emulation — if
you intend a disk to be used as an RK05 but it's not quite 4800 blocks long, E11 will auto-detect it as an RK02,
which probably isn't what you want. Use an /RK05 switch to be sure.

MOUNT DK: drive switches:	
/RK02	set drive type to RK02 (1.2 MB cartridge drive)
/RK03	set drive type to RK03 (2.5 MB cartridge drive)
/RK05	set drive type to RK05 (2.5 MB cartridge drive)
/WPROTECT	enable write protection (syn. /RONLY)

SET DK: controller parameters:	
CSR=$nnnnnn$	set CSR address to $nnnnnn$ octal
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
VECTOR=nnn	set vector address to nnn octal

4.3.5   DL: — RL01, RL02

The RL01 and RL02 are top-loading cartridge drives which hold 5 and 10 MB, respectively. DEC marketed them
successfully as a replacement for the RK05, and they were extremely popular in the 1980s, due to their reliability,

small size (a 10.5" rack-mount box) and the relatively low price of the drives (however the cartridges were very expensive). They turn at 2400 RPM, vs. 1500 RPM for the RK05. New RL01 and RL02 cartridges come with a bad block table written by the manufacturer in the last track, which the PDP-11 operating systems are careful never to overwrite. Some utilities check for this table and will complain if it is not present, so be sure to get the whole disk when making image files. The PUTR utility (available from `ftp.dbit.com`) knows how to write a null bad block track when creating an empty image file (using its `FORMAT ... /RL0x` command).

MOUNT DL: drive switches:	
/RL01	set drive type to RL01 (5 MB cartridge drive)
/RL02	set drive type to RL02 (10 MB cartridge drive)
/WPROTECT	enable write protection (syn. /RONLY)

SET DL: controller parameters:	
CSR=nnnnnn	set CSR address to nnnnnn octal
CSR=FLOATING	set CSR address to be auto-configured
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
DEFAULT	set controller type to default (RL11 for Unibus, RLV12 for Q-bus)
RL11	set controller type to RL11 (Unibus)
RLV11	set controller type to RLV11 (Q18)
RLV12	set controller type to RLV12 (Q22 with BAE register)
VECTOR=nnn	set vector address to nnn octal
VECTOR=FLOATING	set vector address to be auto-configured

4.3.6   DM: — RK06, RK07

The RK06 and RK07 are top-loading twin-platter cartridge drives with capacities of about 13 and 27 MB, respectively. They were mounted on free-standing low-boy cabinets and were sort of a poor man's Massbus drive. Even the geometry and register layouts are similar to Massbus but there are only two platters (one timing surface, three data surfaces), the disks turn at only 2400 RPM, and the control bus is serial rather than parallel. The drives could be dual-ported but this configuration was rare.

MOUNT DM: drive switches:	
/RK06	set drive type to RK06 (13 MB cartridge drive)
/RK07	set drive type to RK07 (27 MB cartridge drive)
/WPROTECT	enable write protection (syn. /RONLY)

SET DM: controller parameters:	
CSR=nnnnnn	set CSR address to nnnnnn octal
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
VECTOR=nnn	set vector address to nnn octal

4.3.7   DP: — RP02, RP03

These are top-loading 12-platter pack drives originally designed to be used with the PDP-10 series, but adapted for the PDP-11. The RP02 holds 20 MB per pack, while the RP03 holds 40 MB. RT-11 can't use a whole RP03 at once due to the RT-11 file structure's limit of 32 MB per volume, so it makes an RP03 look like two 20 MB disks. Although these were fairly early drives and weren't very widely used, the programming model is straightforward and as a result, aftermarket controllers that emulate them using SMD drives are common. Some of these controllers

supported special drive geometries to give extended capacity and/or use the untranslated SMD sector addressing. In order to emulate these configurations, E11 supports the /CYLINDERS:n, /HEADS:n, and /SECTORS:n switches so that the user may specify custom disk dimensions.

MOUNT DP: drive switches:	
/CYLINDERS:n	set non-standard number of cylinders
/HEADS:n	set non-standard number of heads
/RP02	set drive type to RP02 (20 MB 12-platter pack drive)
/RP03	set drive type to RP03 (40 MB 12-platter pack drive)
/SECTORS:n	set non-standard number of sectors
/SERIAL:$nnnn$	set drive serial number (4 decimal digits)
/WPROTECT	enable write protection (syn. /RONLY)

SET DP: controller parameters:	
CSR=$nnnnnn$	set CSR address to $nnnnnn$ octal
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
VECTOR=nnn	set vector address to nnn octal

4.3.8   DS: — RS03, RS04

The RS03 and RS04 are fixed-head Massbus disks that hold 512 KB and 1024 KB, respectively. As with the RS11, these were used for swapping and other cases where speed is critical, so mounting them as RAM disks would work well.

MOUNT DS: drive switches:	
/RS03	set drive type to RS03 (512 KB fixed-head disk)
/RS04	set drive type to RS04 (1024 KB fixed-head disk)
/WPROTECT	enable write protection (syn. /RONLY)

SET DS: controller parameters:	
CSR=$nnnnnn$	set CSR address to $nnnnnn$ octal
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
RH11	set controller type to RH11
RH70	set controller type to RH70
VECTOR=nnn	set vector address to nnn octal

4.3.9   DT: — TU55, TU56 DECtape

Although it's actually 3/4" magnetic tape, DECtape is block-replaceable and uses a fixed number of fixed-size blocks, so really it behaves more like disks do than tapes. It puts 578 512-byte blocks on a 260' tape, and was very popular for off-line storage in the early 1970s when hard disk space was expensive, and floppy disks were not yet widely available. The TU55 is a single-transport drive, while the TU56 has two independent transports, so it takes two MOUNT DTn: commands to define a TU56. That difference is invisible to the PDP-11 so there's no switch to tell them apart.

MOUNT DT: drive switches:	
/WPROTECT	enable write protection (syn. /RONLY)

SET DT: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR6 for Unibus, BIRQ4 for Q-bus)
VECTOR=*nnn*	set vector address to *nnn* octal

4.3.10   DU: — MSCP disks

MSCP (the Mass Storage Control Protocol) is a flexible, device-independent protocol which allows any disk device (up to 2 TB) to connect to any of DEC's later computers. It was used in all of DEC's PDP-11 disk controllers from the mid-1980s on. DEC imposed an artificial limit of four drives per controller, upon which some versions of some PDP-11 operating systems depend. Also, DEC wants unit numbers to be unique within a system (i.e. if there's a unit 0 on one controller there should be no unit 0 on any other controller), and the maximum allowable unit number varies from one device type to another, depending on the method used to assign a unit number to a drive (unit select cap for the "ready" light, DIP switches, etc.). Unlike many of DEC's own drives, E11 implements the full 16-bit unit numbers specified by MSCP, so the amount of available memory is the only limit on unit numbers or the number of drives per controller. Even so, care should be taken to observe the limitations that the PDP-11 operating system imposes on MSCP configuration.

MSCP provides a way for the PDP-11 to find out the model name of each drive, which can be one to three letters followed by two decimal digits. This is largely a cosmetic feature, since MSCP already has ways for the PDP-11 to find out a disk's size and whether it's fixed or removable, so most of the time the name is meaningless. But all the same, the drive type can be set to any appropriate string using the "/TYPE:*name*" switch on the MOUNT command. The default drive type string is "RA81" for image files, or the actual drive type ("RX50" etc.) for floppies.

In some cases the drive type is important, for example some of the programs that format floppy disks on RQDX3 and RQZX1 controllers will refuse to go ahead unless they believe that the drive is an RX33, so you should use "/TYPE:RX33" when mounting a floppy drive which is to be formatted using standard software.

MSCP requires that each volume (except for floppy disks) contain a relocation control table (RCT) at the end of the disk, with a minimum length of one block. This table is used for remapping bad blocks and making the volume appear to be error-free. Since all current PC disk media (except floppy disks) provide some form of invisible bad block remapping of their own, they are already error-free, so there is no real need for this feature. Some aftermarket SCSI controllers provided a one-block RCT anyway, by deducting one block from the reported size of the volume. E11 can be set to do the same thing using the /RCT switch. This is the default for SCSI disks, but it can be overridden using the /NORCT switch.

MOUNT DU: drive switches:	
/NORCT	don't deduct one block for relocation control table
/RCT	deduct one block for relocation control table
/TYPE:*xxxyy*	set drive type to *xxxyy* (1–3 letters, 2 digits)
/WPROTECT	enable write protection (syn. /RONLY)

SET DU: controller parameters:	
CSR=nnnnnn	set CSR address to nnnnnn octal
CSR=FLOATING	set CSR address to be auto-configured
DEFAULT	set controller type to default (UDA50A for Unibus, RQDX3 for Q-bus)
KDA50	set controller type to KDA50
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
RQDX3	set controller type to RQDX3
RQZX1	set controller type to RQZX1
UDA50	set controller type to UDA50
UDA50A	set controller type to UDA50A

4.3.11   DX: — RX01

The RX01 is a single sided, single density floppy disk drive which uses 8" disks with the standard IBM 3740 format, the same as was commonly used in CP/M systems. The RX01 is a dual-drive system. If only one of the two units is MOUNTed, the other one will still appear to be there, since the RX11/RXV11 controller has no way of reporting whether a drive is present or not, but any attempts to do I/O to the missing drive will return error status.

E11 can read and write actual RX01 floppy disks, using an 8" drive attached to a PC floppy controller (with an adapter such as D Bit's FDADAP board) which supports single density.

MOUNT DX: drive switches:	
/WPROTECT	enable write protection (syn. /RONLY)

SET DX: controller parameters:	
CSR=nnnnnn	set CSR address to nnnnnn octal
CSR=FLOATING	set CSR address to be auto-configured
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
VECTOR=nnn	set vector address to nnn octal
VECTOR=FLOATING	set vector address to be auto-configured

4.3.12   DY: — RX02, "RX03"

The RX02 is a double-density follow-on product to the RX01. It holds twice as much data per disk, and also uses DMA instead of programmed I/O to copy between PDP-11 memory and the on-board sector buffer, so there's less CPU overhead. DEC apparently planned a double-sided upgrade, which became commonly know as the "RX03" but was never actually released. However there are connectors for the extra heads on the drive's controller board (most of the "brains" of the RX01 and RX02 systems are actually in the drive), and DEC even documented the register fields that have to do with using double-sided disks, and the RT-11 V4.0 DY.MAC driver includes support (disabled under conditionals) for double-sided disks, but it was removed in later versions.

E11 includes the double-sided support in its emulation. The /SS and /DS switches can be used when mounting the drive to set the number of sides, and SET DYu: SS (or DS) can be issued at any time to change it. This may be necessary when swapping disks in a real floppy drive, because unlike 8" drives, 5.25" drives have no way of distinguishing single- from double-sided disks, so it can't be done automatically.

RSTS/E uses the names DX: and DY: interchangeably to refer to an 8" floppy drive, which may be either an RX01 or RX02. Early versions of E11 did the same thing and used a SET command to set the actual controller type, but

it was decided that it would be better to be consistent and have the device name always reflect the device type. This may mean that the device name needed to emulate an existing drive under E11 is different from the name that had been used to refer to the same drive under RSTS/E.

MOUNT DY: drive switches:	
/DS	double-sided disk
/RX01	single-density disk
/RX02	double-density disk
/SS	single-sided disk
/WPROTECT	enable write protection (syn. /RONLY)

SET DY: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
CSR=FLOATING	set CSR address to be auto-configured
DS[1]	disk in drive is double-sided
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
SS[1]	disk in drive is single-sided
VECTOR=*nnn*	set vector address to *nnn* octal
VECTOR=FLOATING	set vector address to be auto-configured

[1]Really a drive option — unit must be mounted before this option is set

4.3.13   FX: — Fox 2/30 "drum"

The Foxboro Company's Fox 2/30 process control system was based on a PDP-11/15 CPU but used a Foxboro-specific "data bus interface" (DBI) to connect Foxboro's own hardware to the CPU over a private bus. One of these devices was a Digital Development Corporation 6200 or 7311 fixed-head disk (or SSDM 100 RAM disk) which the Foxboro documentation calls a "drum", probably because some of the control electronics are inside the disk shroud, which leads it to be much taller than is usual for a disk. The disk was available in several configurations, with a varying number of tracks (64, 96, 128, 144, and 256 are known possibilities), each containing 242 sectors of 32 bytes each. There's only one disk so it must be FX0:, and the size of the image file decides how many tracks exist. The FX1: device is used to emulate the Foxboro paper tape reader.

The SET CSR value for the FX: device is the base address of the DBI, normally 164000. E11 emulates the registers for FX0: or FX1: or both (depending on whether they've been MOUNTed), but doesn't get in the way of other DBI registers in case the real DBI is connected to E11 through a Unibus adapter. The register at 164000 contains status bits for each of the devices plugged into the DBI. E11 emulates this register as needed, and if a Unibus adapter is present, it combines the emulated status bits with the ones from the real DBI.

MOUNT FX: drive switches:	
/WPROTECT	enable write protection (syn. /RONLY)

SET FX: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR7 for Unibus, BIRQ7 for Q-bus)
VECTOR=*nnn*	set vector address to *nnn* octal

4.3.14 HD: — Hypothetical disk

This is a hypothetical disk device which exists only in Ersatz-11. Its original definition was based on reverse-engineering the `HD_SYS.EXE` device emulation that came with the so-called "Russian" LSI-11/2 emulator which used to be floating around the Internet, so that E11 would be able to boot disk images intended for that emulator. However that emulation was not very useful, since it (apparently) had no provision for interrupts or memory beyond 64 KB or drives bigger than 32 MB. The current emulation has been extended to support 22-bit addressing, up to 16 drives of up to 2 TB each, and it uses interrupts to signal completion so the system doesn't have to stall during I/O. The main reason it's interesting is that it supports variably sized devices like MSCP does, but with a much simpler programming model so that PDP-11 drivers can be written which require far less memory than MSCP drivers. An RT-11 driver (HD.MAC) is included with E11, and one for Fuzzball is available from `ftp.dbit.com`.

MOUNT HD: drive switches:	
/WPROTECT	enable write protection (syn. /RONLY)

SET HD: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
VECTOR=*nnn*	set vector address to *nnn* octal

4.3.15 PD: — PDT-11/150 RX01

The PDT-11/150 was a microcomputer based on the LSI-11 chipset. It had no expansion bus, and used a pair of 8085A processors to simulate the approximate equivalent of a small PDP-11/03 configuration, but it could be built more cheaply and sold as a "smart" terminal system for data entry etc. It used RX01 drives but instead of an RXV11 controller (which would have required a Q-bus) it used a WD1771 floppy controller chip, which one of the 8085As used to simulate an almost-RXV11-like interface which was called the RXT11. It's just different enough from the RXV11 to be incompatible with regular RX01 software, so E11 provides it as a separate emulated controller type.

MOUNT PD: drive switches:	
/WPROTECT	enable write protection (syn. /RONLY)

SET PD: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
VECTOR=*nnn*	set vector address to *nnn* octal

Chapter 5

Tape Devices

The full range of standard PDP-11 magtape drives and controllers can be emulated under Ersatz-11. Tape drives are configured in the same way as disks, with a `MOUNT` command to set up each emulated tape unit. The defaults for controller and drive parameters also work the same way as disks, with the controller model based on the emulated CPU type and the CSR and vector computed according to "floating" address rules if necessary. However the default tape drive models are essentially chosen arbitrarily, since unlike disks, there's no point in trying to guess the drive model based on the size of the PC file or device that's being used to emulate a tape drive, because the amount of data on a tape is variable anyway. In any case, the default drive and controller parameters can be overridden with `MOUNT` switches and `SET` commands, the same way they can with disks. The `/RONLY` (syn. `/WPROTECT` switch may be specified on any `MOUNT` command, to lock the tape against writes.

5.1   PC tape devices

As with disks, E11 uses a device-independent interface to connect any emulated tape drive to any real tape device. This connection is made with a `MOUNT` command, and the current configuration of an emulated tape can be displayed with a `SHOW` command.

5.1.1   Tape image files

Command syntax:

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

Special switches: `/MAXRECORD:`*n*

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 65536 bytes. E11's memory usage may be decreased slightly 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 tapes 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. The file is located using E11's usual rules, see section 1.8 for details. If the file does not exist, it is created as a zero-length file in the current working directory.

5.1.2 RAM tape drives

Command syntax:

MOUNT *ddcu*: RAM: [*switches*]

Special switches: /LOAD:[*d*:*path*\]*filename*[.TAP], /SIZE:*blocks*

RAM tapes may be created using regular PC memory, just like RAM disks. Use SHOW MEMORY to see how much memory is available, E11 uses 1 MB (1,048,576 bytes) by default but a different number of blocks may be specified using the "/SIZE:n" switch. Other units besides blocks may be used by appending the unit to the number (with no space in between), for example "/SIZE:100MB" sets the size to 100 megabytes. As with RAM disks, the actual amount of memory needed to create a RAM tape is slightly more than the tape's raw capacity, due to overhead for internal data structures. The memory allocated is shared between tape data and record length information, so data stored in many short records will require more memory than the same amount of data stored in a few larger records.

The RAM tape will initially be empty unless it is loaded from a tape image file, which is done with the "/LOAD:[*d*:*path*\]*filename*[.TAP]" switch. The file is located using E11's usual rules, see section 1.8 for details. If this switch is given, the RAM tape will be the same size as the file, unless the size is explicitly set with the "/SIZE" switch. The RAM tape is loaded from the file once when the MOUNT command is issued, then the file is closed. Any data written to the RAM tape do not change the file from which it was loaded.

5.2 Emulated PDP-11 tape devices

This section describes the emulated PDP-11 tape drive models, and defines the device-specific MOUNT command switches that apply to each.

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, software can't tell)
MU:	0–65535	TMSCP	/TYPE:*xxxyy*

5.2.1 CT: — TU60 DECcassette

The TU60 was designed to provide cheap off-line storage using ordinary audio cassette tapes. The TU60 is a 51/4" box containing two tape transports and most of the control electronics, connected by a ribbon cable to the TA11

controller, which is an M7892 quad SPC containing a dumb interface.

Under the CAPS-11 cassette programming system, the TA11/TU60 could be used as the system's only mass storage device. It uses programmed I/O rather than DMA, so tape data bytes are transferred one at a time. Unlike other tape devices, the TU60 requires the PDP-11 to know the record length ahead of time when reading a record, and gives an error if the PDP-11 attempts to read more or less than the correct number of bytes. The "`LOG CT`n`:`" command may be used to find out whether a PDP-11 program is trying to read the wrong record size.

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 physical medium. This allows the emulated TU60 general access to tapes that may have been created using some other device (or emulated device), where an extra tape mark at BOT would violate the labeling standards.

MOUNT CT: drive switches:	
`/WPROTECT`	enable write protection (syn. `/RONLY`)

SET CT: controller parameters:	
`CSR=`$nnnnnn$	set CSR address to $nnnnnn$ octal
`PRIORITY=`n	set interrupt priority to n (4–7)
`PRIORITY=DEFAULT`	set interrupt priority to default (BR6 for Unibus, BIRQ4 for Q-bus)
`VECTOR=`nnn	set vector address to nnn octal

5.2.2 MM: — TE16/TU16, TU45, TU77

These tapes all use the TM02 or TM03 Massbus tape formatter to control up to eight slaves. The formatter itself uses one of the eight possible Massbus unit numbers (typically 0), and the resulting two-level unit selection scheme slightly complicates the device naming, since it appears nowhere else. 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 is 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, TM03 formatter 2, slave 5. This same name format may also be used in any other E11 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 MM: drive switches:	
/SERIAL:*nnnn*	set drive serial number (4 decimal digits)
/TE16	set drive type to TE16 (45 IPS NRZI/PE)
/TU45	set drive type to TU45 (75 IPS NRZI/PE)
/TU77	set drive type to TU77 (125 IPS NRZI/PE)
/WPROTECT	enable write protection (syn. /RONLY)

SET MM: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
RH11	set controller type to RH11
RH70	set controller type to RH70
VECTOR=*nnn*	set vector address to *nnn* octal

5.2.3   MS: — TK25, TS04, TS05, TU80

These drives are microcomputer-controlled and have a programming model which is like a simplified version of TMSCP. There are no drive type switches because each controller model supports only one drive type anyway, so the drive type can be controlled implicitly by changing the controller type with a SET command. The TK25 is a cartridge tape, while the others are 9-track magtapes, but they all look similar from the PDP-11 operating system's point of view.

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 issue arises only when there is more than one MS: style tape drive, which is rare.

MOUNT MS: drive switches:	
/WPROTECT	enable write protection (syn. /RONLY)

SET MS: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
DEFAULT	set controller type to default (TU80 for Unibus, TSV05 for Q-bus)
EXTFEAT	enable extended features
NOEXTFEAT	disable extended features
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
TQK25	set controller type to TQK25 (Q22 TK25 controller)
TS11	set controller type to TS11 (Unibus TS04 controller)
TSU05	set controller type to TSU05 (Unibus TS05 controller)
TSV05	set controller type to TSV05 (Q22 TS05 controller)
TU80	set controller type to TU80 (Unibus TU80 controller, M7454 module)
VECTOR=*nnn*	set vector address to *nnn* octal
VECTOR=FLOATING	set vector address to be auto-configured

5.2.4   MT: — TS03, TU10

The TU10 is a vacuum column tape drive which interfaces to the PDP-11 through a classic Unibus interface, made up of a number of flip chip modules on a wire wrap backplane. The TS03 (actually a rebadged Kennedy 9700) is a very small tension-arm drive which uses up just 10.5" of height in a rack, and uses a physically smaller controller made with higher density modules. However they both look the same to software, so there are no drive type switches to distinguish them. The 7-track version of the TU10 is not supported.

MOUNT MT: drive switches:	
/WPROTECT	enable write protection (syn. /RONLY)

SET MT: controller parameters:	
CSR=$nnnnnn$	set CSR address to $nnnnnn$ octal
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
VECTOR=nnn	set vector address to nnn octal

5.2.5   MU: — TMSCP tapes

TMSCP (the Tape Mass Storage Control Protocol) is a device-independent protocol for connecting arbitrary tape drives to any of DEC's later computers. It doesn't particularly reduce the amount of implementation-specific quirks that must be dealt with (tapes never had anywhere near the amount of device-specific details as disks do to begin with), but it provides symmetry with the MSCP protocol for disks, and some systems are able to capitalize on this by sharing code between the disk and tape drivers. DEC generally didn't have more than one tape drive per TMSCP controller, but as with MSCP disks, E11 has no such artificial limits. However this means it may be possible to configure a system which is incompatible with the PDP-11 operating system's TMSCP driver, so it is best to define a separate controller for each drive anyway.

As with 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 TU81.

MOUNT MU: drive switches:	
/TYPE:$xxxyy$	set drive type to $xxxyy$ (1–3 letters, 2 digits)
/WPROTECT	enable write protection (syn. /RONLY)

SET MU: controller parameters:	
CSR=$nnnnnn$	set CSR address to $nnnnnn$ octal
CSR=FLOATING	set CSR address to be auto-configured
DEFAULT	set controller type to default (KLESI for Unibus, TQK50 for Q-bus)
KLESI	set controller type to KLESI (Unibus TU81 controller)
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
TQK50	set controller type to TQK50 (Q22 TK50 controller)
TQK70	set controller type to TQK70 (Q22 TK70 controller)
TUK50	set controller type to TUK50 (Unibus TK50 controller)

Chapter 6

Serial Lines

E11 has flexible support for serial and line printer devices. As with disks and tapes, any supported PC serial device may be used to emulate any emulated PDP-11 character device, and once again E11 chooses sensible defaults for the controller types and CSR/vector addresses based on the emulated CPU type and the configuration of "floating" devices, so in many cases no `SET` commands will be needed.

Each serial line is created with an `ASSIGN` command. `TT0:`, the system console, is connected to `CON1:` (the first emulated VT100 session, displayed on the PC video display) when E11 starts up, but it can be reassigned to any other screen or serial port with an `ASSIGN TT0:` command. If it is assigned to an RS232 serial port, the E11 prompt can no longer be popped up using Shift-Enter, because this key combination does not correspond to any ASCII character that could be sent over a serial line. So by default, a BREAK condition on the console serial port will bring up the "`E11>`" prompt. If this is inconvenient, the `SET BREAK` command can be used to define one ASCII character which will cause the prompt to pop up. For example, putting a `SET BREAK 20` command in the E11.INI initialization file will make the prompt pop up whenever CTRL/P is typed on the console terminal.

Output to any serial or printer device can be captured to a PC file using E11's `LOG` command. For example, `LOG TT0: FOO` will cause all data displayed on the console terminal to be saved in FOO.LOG, until logging is turned off with a `LOG TT0:` command (with no filename).

An `ASSIGN` command will fail if the specified PC port doesn't exist, or if the new `ASSIGN` command would steal `TT0:`'s device for some other port. There must always be something attached to `TT0:` since that's E11's console terminal.

Note that Ersatz-11 does not flag an error if the `ASSIGN` command assigns a `TT:` port to a printer, or assigns an `LP:` port to a screen, even though these are usually 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 serial ports or video screens respond to `XON/XOFF` flow control. One good reason to `ASSIGN` an `LP:` port to a VT100 session is that a `LOG` command will capture the output to a file, and there's no need to actually pop up that session and see it on the screen.

6.1 PC serial devices

name	units	type	switches
`CON`u`:`	1–12	video console	`/SIZE:`W`x`H

6.1.1   Serial options common to all devices

E11 has a set of "serial options" which are common to all serial device types, and are used to set communications parameters. These options may be used as either a switch (with a preceding "/" character) on the `ASSIGN` command when the port is created, or as a separate `SET` command any time thereafter. For example:

```
ASSIGN TT1: COM2: /MODE:1200,N,8,1
SET TT1: MODE=2400,E,7,1
```

(These commands are just examples and do not work on the OS/2 version of Ersatz-11, since it has no support for COM ports.) The first command will create TT1: and attach it to COM2. The line parameters are set to 1200 baud, no parity, 8 data bits, and one stop bit. The second command will change the existing TT1: port to use 2400 baud, even parity, 7 data bits, and one stop bit. The option name may be separated from its parameters by either a ":" or "=" character.

Note that some emulated port types allow the communications parameters to be set by the PDP-11 operating system. E11 allows this by default, but in some cases the user may want to override the PDP-11's parameters, for example to set a port to a higher baud rate than the PDP-11 equivalent supports. In this case the `/LOCK` switch will be useful when creating the port (usually in combination with a `/MODE` switch). The `SET` *ddcu*: `LOCK` and `SET` *ddcu*: `UNLOCK` commands can be used to lock and unlock the port's communications parameters after the port has already been created.

The sections that follow describe each physical PC serial (or serial-like) device that E11 supports, along with the command syntax needed to use them as emulated PDP-11 serial ports. As with disk and tape units, the ports are created on a line-by-line basis so there is no need to use the same kind of physical port for all emulated ports of a given type. Also, serial multiplexers can be sparsely populated. Only one port needs to be created for the whole emulated multiplexer to exist. Missing ports will simply throw transmitted data away and will never receive any input.

6.1.2   Video consoles

Command syntax:

```
SET CONn: SIZE=WxH (if needed)
ASSIGN ddcu: CONn: [switches]
```

Special switches: `/NOREPLY` `/SIZE:`*WxH*

The specified PDP-11 terminal port is connected to one of twelve simulated VT100s 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 on the old 84-key AT keyboard, unless other keys are redefined 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 whenever another Alt/F*n* keypress switches the screen to display one of the hidden sessions.

This is only the default behavior of the function keys, if they are redefined using `DEFINE KEYPRESS` commands then other keys will need to be defined to switch displays, using the "`PRIMARY` *n*" and "`SECONDARY` *n*" keyscript commands.

The `/NOREPLY` switch means that this emulated VT100 should not transmit any automatic replies to control sequences sent by the PDP-11. This can be handy if you're using the "multiple physical ports" feature and want to make sure that only one of the ports answers when you something like "SET TERM/INQUIRE".

The `/SIZE:`*W*`x`*H* switch sets the size of the text screen to *W* columns by *H* lines. The selected window size applies to what would normally be a VT100 in 80-column mode. If the emulated VT100 receives the escape sequence to switch to 132-column mode, E11 will attempt to find a video mode which is the same height but 65% wider. If no such mode is available, E11 will take the closest fit, which may be the same one from before the change. In any case, a session created with `/SIZE:132x24` is not the same as an 80-column VT100 which is initially in 132-column mode. Sending ESC [?3l to this session will not switch it to 80-column mode.

The `SET CON`*n*`: SIZE=`*W*`x`*H* command sets the default screen size for all future `ASSIGN` commands on that unit. It may still be overridden by a `/SIZE` switch on that `ASSIGN`.

`CON`*n*`:` is the only device that may be used with the "`VT:`" VT11 vector graphics display processor, or with the "`VR:`" VT30 raster graphics processor. Graphics support requires that an OpenGL driver be installed in the PC's operating system. The display speed improves tremendously if the driver supports hardware acceleration, rather than doing all rendering entirely in software.

6.2   PC serial pseudo-devices

This section describes pseudo-device layers which may be configured to modify the behavior of E11's actual serial port drivers.

syntax	type	switches
THROTTLE: *pcdev*:	speed limiter for a PC port	/RXDELAY:*n* /TXDELAY:*n*
dev_1 + dev_2 + ... + dev_n	multiple PC ports	

6.2.1   Throttled I/O

Command syntax:

ASSIGN *pdp11dev*: THROTTLE: [/RXDELAY:*r*] [/TXDELAY:*t*] *pcdev*: [/*pcswitches*]

The THROTTLE: pseudo-device adds a throttle to another serial device, setting a limit on the rate at which characters are transmitted and/or received. It is an intermediate layer between an emulated PDP-11 port and an actual PC port, and this is reflected in the command syntax used to configure it. In the ASSIGN command, the THROTTLE: keyword and its switch(es) go between the emulated PDP-11 device name and the physical PC port name.

The ASSIGN command as shown above creates a PDP-11 serial device and connects it to a PC serial device, and times all I/O to ensure that transmitted characters go out at least t milliseconds apart, and that received characters are seen by the PDP-11 hardware at least r milliseconds apart. The /RXDELAY and /TXDELAY switches are both optional. If either is omitted then serial data pass at full speed in the corresponding direction.

6.2.2   Multiple physical ports on one emulated line

Command syntax:

ASSIGN *ddcu*: dev_1 + dev_2 + ... + dev_n

A single emulated PDP-11 serial line may be connected to two or more physical serial ports (as described in section 6.1), by giving multiple physical port names in the ASSIGN command, separated by plus sign (+) characters. Any data transmitted over that PDP-11 port will be copied to all of the physical ports. Input received on any of the physical ports will be mixed together and will go in the PDP-11 port's receiver. Similarly, the emulated modem control signals will be driven in all of the physical ports, however incoming modem status signals will be sampled only on the first port (dev_1).

The physical port names may be any mixture of PC serial devices. In most cases there is no reason to connect two physical ports to the same emulated port so this feature will not be used by most users, but it can be useful for logging purposes or to provide duplicate command terminals so that a system may be controlled from more than one place. The /NOREPLY switch may be useful on CON*n*: sessions, in order to prevent E11's VT100 emulation from replying to a "what-are-you" escape sequence, in cases where the emulated line is shared with a real terminal which will try to reply simultaneously.

Example:

ASSIGN TT1: COM1: + CON2:

This command will connect TT1: (the second emulated DL11 port) to both COM1: (the first PC serial port) and

CON2: (the video session which can be popped up with Alt/F2). Output sent to TT1: will appear on both COM1: and CON2:, and input typed at either port will be received by TT1:.

6.3   Emulated PDP-11 serial devices

This section describes each PDP-11 serial device type emulated by Ersatz-11.

name	units	controller	notes
LPu:	0–3	LP11/LPV11	
TTu:	0–15	DL11/DLV11	KBu: and YLu: are synonyms
YZcu:	0–7	DZ11/DZQ11/DZV11	

6.3.1   LP: — LP11, LPV11 line printer ports

These ports connect a single line printer, using device registers which look similar to the transmitter half of a DL11/DLV11 serial line unit. They normally use a Data Products compatible parallel interface to connect to the printer. Converter boxes are available from Black Box Corporationto allow connecting such a line printer to a PC RS232 serial port.

SET LP: controller parameters:	
CSR=$nnnnnn$	set CSR address to $nnnnnn$ octal
MODE=b,p,d,s	set communications parameters
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR4 for Unibus, BIRQ4 for Q-bus)
VECTOR=nnn	set vector address to nnn octal

6.3.2   TT: — DL11, DLV11 single serial line units

These are the standard single-line serial ports. Every PDP-11 has at least one, which is used for the console terminal. By default Ersatz-11 connects the console terminal to CON1:, which is the same VT100 session that E11 uses for its own command prompt. The DLV11J 4-line serial card actually appears to software as four separate DLV11 ports, so it can be configured in E11 using four ASSIGN commands.

SET TT: controller parameters:	
CSR=$nnnnnn$	set CSR address to $nnnnnn$ octal
CSR=FLOATING	set CSR address to be auto-configured
MODE=b,p,d,s	set communications parameters
PRIORITY=n	set interrupt priority to n (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR4 for Unibus, BIRQ4 for Q-bus)
VECTOR=nnn	set vector address to nnn octal
VECTOR=FLOATING	set vector address to be auto-configured

6.3.3   YZ: — DZ11, DZQ11, DZV11 4/8-line serial multiplexers

This series of boards is essentially a compressed version of the old DJ11 multiplexer. Some confusion about port numbering is possible because the early non-FCC Unibus cabinet kits supported 16 lines even though the DZ11 boards only had 8 lines each, so it takes two boards to drive all 16 lines of the distribution panel. This is outlined below:

model	interface	distribution panel
DZ11A	RS232	16 lines, only 8 work
DZ11B	RS232	none, adds 8 more ports to DZ11A
DZ11C	20 mA	16 lines, only 8 work
DZ11D	20 mA	none, adds 8 more ports to DZ11C
DZ11E	RS232	DZ11A + DZ11B (16 working lines)
DZ11F	20 mA	DZ11C + DZ11D (16 working lines)

The Q-bus versions (DZQ11 and DZV11) support only 4 lines instead of 8. The DZQ11 is a newer dual-height version of the quad-height DZV11.

SET YZ: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
CSR=FLOATING	set CSR address to be auto-configured
DEFAULT	set controller type to default (DZ11 for Unibus, DZQ11 for Q-bus)
DZ11	set controller type to DZ11 (Unibus, 8 lines)
DZQ11	set controller type to DZQ11 (Q-bus, 4 lines)
DZV11	set controller type to DZV11 (Q-bus, 4 lines)
LOCK	lock MODE settings against changes by PDP-11
MODE=*b*,*p*,*d*,*s*	set communications parameters
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR5 for Unibus, BIRQ4 for Q-bus)
UNLOCK	unlock MODE settings
VECTOR=*nnn*	set vector address to *nnn* octal
VECTOR=FLOATING	set vector address to be auto-configured

Chapter 7

Miscellaneous Devices

This chapter describes devices which don't fit into any of the major categories covered in the preceding chapters. This includes graphics displays, paper tape I/O, and the interface to the PC's native file system.

Miscellaneous PDP-11 devices:

name	unit(s)	controller	command to create	type
(none)			LOAD/ROM	ROM devices
DO:	0–3	virtual	MOUNT	PC file interface
KD:	A	M8644 CDR	ASSIGN	countdown registers
KP:	0–1	KW11P	ASSIGN	programmable clock
PP:	0	PC11	MOUNT	paper tape punch
PR:	0	PC11	MOUNT	paper tape reader
VR:	0	VT30	ASSIGN	color graphics display
VT:	0	VT11	ASSIGN	vector graphics processor

7.1   Installable user-written plug-in emulation modules

Command syntax:

INSTALL [$d{:}path\backslash$]$filename$[.DLL]

Special switches: none

This command allows custom user-written (or third-party) plug-in emulation modules to be dynamically installed at run time. This makes it possible for custom hardware to be emulated, without requiring access to the Ersatz-11 source code. Modules are coded in C or 80x86 assembly language, and linked to produce Win32-format dynamic link libraries (".DLL" files). The .DLL loader is part of E11 itself rather than a service of the PC operating system, so the same module format is used in the DOS, Windows, and Linux versions of Ersatz-11, allowing some modules to run unchanged in all environments.

E11 provides a wide selection of callable entry points to be used by the device emulation module. Typically a loaded module will uses these services to register a range (or ranges) of I/O page addresses which it wants to handle. Then, whenever the PDP-11 accesses a device register in that range, the user code is called to emulate the register. Calls are also provided to enqueue and cancel interrupt requests, and also to perform emulated DMA,

allocate/free memory, etc. The programming interface is described in Appendix C.

Loadable modules can also provide a wide range of other services besides just emulating existing custom hardware. Modules have been written which support high-speed inter-task communications (to connect the PDP-11 system to other software running on the same PC under a multi-tasking operating system), extended memory for virtual arrays, and access to a dual-ported memory device for connecting to another processor, among other things.

D Bit can provide programming information, sample code, and consulting services for users who wish to implement a custom device emulation module.

7.2   ROM devices

Command syntax:

LOAD/[EEP]ROM [/BANKED] [d: path\]filename[.PDP] start[: end]

Special switches: `/BANKED`, `/EEPROM`, `/ROM`

E11 can simulate ROM devices, either simple unbanked ones such as the M9312 bootstrap/diagnostic board, or banked ones like the BDV11 board or the on-board ROM in the later quad-height Q-bus CPU modules.

ROM data are loaded from the specified binary file, which is located using E11's usual search rules (see section 1.8). More than one ROM device can be loaded, so for example the individual 128-byte device bootstraps in a M9312 style ROM board (or the on-board equivalents in a PDP-11/24 or PDP-11/44 CPU) can be loaded from separate files with separate commands.

For linear (non-banked) ROMs, a starting address in the I/O page must be specified. The data from the file will appear starting at that address. If an ending address is specified, the file must be at least long enough to reach that address. If no ending address is given, the length of the emulated ROM is the same as the length of the binary file.

A M9312-style bootstrap/diagnostic ROM board can be configured and started with the following commands:

```
E11>LOAD/ROM DIAGBOOT.ROM 17765000
E11>LOAD/ROM RL02BOOT.ROM 17773000
E11>LOAD/ROM MSCPBOOT.ROM 17773200
...
E11>GO 165020
000000 000000 000000 000000
@
```

This is just an example of what can be done. Normally there would be no point in using the M9312 boot ROMs since E11 contains its own bootstraps for all bootable devices that it emulates. Anyway there may be legal problems with copying the ROMs into image files, since they are copyrighted by DEC.

For banked ROMs, the `/BANKED` switch must be used. The data from the file will appear through a 512.-byte window, and the page which is visible is selected by the PDP-11 using the page control register (PCR), which for emulation purposes is considered by E11 to be part of the CPU board (since in most cases, it is).

There are two styles of page control register. `SET CPU KDF11` (or `SET CPU BDV11`) selects the style used in the

KDF11 on-board ROM (or the BDV11 boot/diagnostic board). `SET CPU KDJ11` selects the style used in the on-board ROM of the KDJ11x CPU boards.

The starting address for banked ROM windows is either (17)773000 or (17)765000. The windows must be 512. bytes long, which is E11's default anyway so there's no need to give the end of the address range. The `/EEPROM` switch means that the memory is actually EEPROM (as used on the KDJ11x CPU boards), and must go in the window starting at (17)765000. This is the default, so there's no need to give any address range at all with `LOAD/EEPROM`. Note that while the memory can be written, the data written are not saved to the file from which the EEPROM was loaded. Regular ROM data go in the window starting at (17)773000, so this is the default for `LOAD/ROM`.

ROM devices can be removed at any time using the `UNLOAD` command. The switches and defaults are the same as for `LOAD`, so the starting address of the ROM to remove need not be specified if the `/BANKED` switch is used.

7.3   DO: — PC file access pseudo-device

Command syntax:

`MOUNT DO`n`:`

Special switches: none

DO: refers to the "DOS file device" (the name is left over from the DOS version of Ersatz-11) . This is a pseudo-device which allows privileged PDP-11 programs and/or device drivers to read and write native PC files from within the emulated PDP-11. It is compatible with the supplied DO.SYS device driver for RT-11, and the DOS.TSK program for RSX which is available from `ftp.dbit.com`. The command to create a DOS file device is simply "MOUNT DOn:", since there is no particular physical PC device attached to the physical side of the emulation. In E11 prior to V2.2, one DOS file device was always configured, but now they must be created explicitly. Normally there's no reason to have more than one, so you can recreate the V2.1A behavior by simply adding "MOUNT DO:" to your E11.INI initialization command file.

7.4   KD: — M8644 countdown register board

Command syntax:

`ASSIGN KD`c`:`

Special switches: none

The M8644 countdown register board was a special-purpose device made by DEC Computer Special Systems. It countains four separate channels of 16-bit counters, each of which counts down every millisecond and can be configured to cause an interrupt upon reaching zero. The command to create a countdown register board is simply "ASSIGN KDc:", since there is no particular physical PC device attached to the physical side of the emulation. On the real device, the interrupt vector and priority were settable for each channel, so E11 has separate `SET` commands for each unit number, even though the four units are otherwise considered to be a single device.

SET KD: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR4 for Unibus, BIRQ4 for Q-bus)
VECTOR=*nnn*	set vector address to *nnn* octal

7.5   KP: — KW11P programmable clock

Command syntax:

ASSIGN KP*n*:

Special switches: none

The KW11P (M7228) is a programmable clock, on a quad Unibus SPC. It has a 16-bit up/down counter which can be programmed for one-shot or periodic use. It has on-board crystal-controlled 10 kHz and 100 kHz clocks, and can also be triggered by the AC line frequency, which requires a connection from the power supply to pin CE1 of the KW11P's slot, or by low-to-high transitions on an external TTL input, which is connected to spade lugs on the M7228 board.

The command to create a KW11P is simply "ASSIGN KP*n*:", since there is no particular physical PC device attached to the physical side of the emulation. The 10 kHz and 100 kHz clocks are derived from the PC's internal timer, as is the AC line input, which ticks at the frequency set by the SET HERTZ command (60 Hz by default).

SET KP: controller parameters:	
CSR=*nnnnnn*	set CSR address to *nnnnnn* octal
PRIORITY=*n*	set interrupt priority to *n* (4–7)
PRIORITY=DEFAULT	set interrupt priority to default (BR6 for Unibus, BIRQ6 for Q-bus)
VECTOR=*nnn*	set vector address to *nnn* octal

7.6   PP: — PC04 paper tape punch

Command syntax:

MOUNT PP: [*d*:*path*\]*filename*[.PAP]

Special switches: none

The PC04 is a high speed paper tape reader/punch, interfaced to the PDP-11 through a PC11 interface. This command attaches the specified PC file to the output half of the device, so that data bytes sent to the punch will be written to the file instead.

In E11 prior to V3.0, one paper tape reader/punch was always configured, but now they must be created explicitly. You can recreate the V2.2 behavior by adding "MOUNT PR: NUL" to your E11.INI initialization command file, and issue new MOUNT command(s) later to change the file(s) used.

See below for SET commands. The PC11 is a combined reader/punch controller so one set of settings configures both the reader and punch halves.

7.7   PR: — PC04 paper tape reader

Command syntax:

`MOUNT PR:` $[d{:}path{\backslash}]filename[$`.PAP`$]$

Special switches: none

This command attaches the specified PC file to the input half of the PC11/PC04, so that input bytes will come from the file instead. The `BOOT PR:` command supports binary executable tape images written in the Absolute Loader format (e.g. by the `LINK /LDA` command in RT-11). The `SET PR: REWIND` command will reset the tape to begin reading at the beginning of the file on the next access.

In E11 prior to V3.0, one paper tape reader/punch was always configured, but now they must be created explicitly. You can recreate the V2.2 behavior by adding "MOUNT PR: NUL" to your E11.INI initialization command file, and issue new MOUNT command(s) later to change the file(s) used.

`SET PR:` controller parameters:	
`CSR=`*nnnnnn*	set CSR address to *nnnnnn* octal
`REWIND`	rewind tape to beginning
`PRIORITY=`*n*	set interrupt priority to *n* (4–7)
`PRIORITY=DEFAULT`	set interrupt priority to default (BR4 for Unibus, BIRQ4 for Q-bus)
`VECTOR=`*nnn*	set vector address to *nnn* octal

7.8   VR: — VT30 color graphics display processor

Command syntax:

`ASSIGN VR: CON`*n*`:`

Special switches: none

The VT30 "Colour Mimic Diagram Display System" is a simple display-list-based raster graphics device, built by DEC CSS in Reading, UK. It is a Unibus device, mounted in a 5.25" high external box. It displays 8x8 characters from a software character generator on a 64x36 grid, yielding 512x288 pixels, displayed on an attached PAL monitor.

E11 requires an EGA, VGA, or SVGA with at least 256 KB of memory to emulate the VT30. The VT30 screen can be popped up with an Alt-function key just like a VT100 session.

`SET VR:` controller parameters:	
`COLOR`	color monitor (default)
`CSR=`*nnnnnn*	set CSR address to *nnnnnn* octal
`GRAYSCALE`	gray-scale monitor (for future expansion, currently always white)
`MONOCHROME`	monochrome monitor (black or white only)
`PRIORITY=`*n*	set interrupt priority to *n* (4–7)
`PRIORITY=DEFAULT`	set interrupt priority to default (BR4 for Unibus, BIRQ4 for Q-bus)
`VECTOR=`*nnn*	set vector address to *nnn* octal

7.9   VT: — VT11 vector graphics display processor

Command syntax:

`ASSIGN VT: CON`*n*`:`

Special switches: none

The VT11 vector graphics display processor consists of three hex-height boards in a 4-slot backplane. It maintains the picture on a VR14 or VR17 monitor by constantly scanning a display list held in PDP-11 memory, and refreshing each vector many times per second. It can be used with a DEC model 375 light pen, and an LK40 ASCII keyboard.

The VT11's output is displayed in a window. The VT11 is assigned to one of the twelve possible virtual console sessions, so it can be popped up with an Alt-function key just like an emulated VT100 session, except of course that it runs the display in graphics mode.

The mouse is used to emulate the DEC model 375 light pen. The crosshairs indicate the mouse position continuously (since a mouse can't be lifted away from the screen like a real light pen), so the left mouse button is used to indicate whether a light pen "hit" should be allowed to register. The PC keyboard takes the place of the LK40 keyboard on a GT40 system, so all keyboard input typed to a VT11 session is sent to TT0:.

`SET VT:` controller parameters:	
`CSR=`*nnnnnn*	set CSR address to *nnnnnn* octal
`PRIORITY=`*n*	set interrupt priority to *n* (4–7)
`PRIORITY=DEFAULT`	set interrupt priority to default (BR4 for Unibus, BIRQ4 for Q-bus)
`VECTOR=`*nnn*	set vector address to *nnn* octal

Chapter 8

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. The SET BREAK command can define a regular ASCII character that pops up a prompt if these methods are all inconvenient. Some versions of the Windows DOS box intercept Alt-SysReq, but Shift-Enter still works. 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.

Ersatz-11 supports command line editing and recall at the "`E11>`" prompt, using the VT52/VT100 arrow keys and some control characters.

key	action
`CTRL/A`	go to beginning of line
`CTRL/D`	delete character to right of cursor
`CTRL/E`	go to end of line
`CTRL/H`	go to beginning of line
`CTRL/J`	delete word to left of cursor
`CTRL/K`	delete to end of line
`CTRL/L`	clear screen and re-display line
`CTRL/R`	re-type line
`CTRL/U`	delete to beginning of line
`CTRL/W`	delete word to left of cursor
`CTRL/Z`	exit prompt and connect to TT0:
`DEL`	delete character to left of cursor
left arrow	move left one character
right arrow	move right one character
up arrow	move up a line in history buffer
down arrow	move down a line in history buffer

@*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 (see section 1.8). Command files may be nested up to four deep. Lines read from the file are not echoed.

ASSEMBLE [/*switches*] [*addr*]

Starts the mini-assembler at the specified address, in the specified address space (syntax and defaults are the same as for the LIST command). E11 will prompt with the address, and you can type in lines of PDP-11 assembly code, which E11 will translate to machine language and deposit into PDP-11 memory at that address. E11 keeps prompting for more lines (at updated addresses) until the user enters a blank line, or types CTRL/C. The syntax is a subset of the standard MACRO-11 syntax. Simple expressions are allowed in operands but there is no symbol table, macro processor, or conditional assembly. Only the .BYTE, .WORD, .BLKB, .BLKW, .ASCII, and .ASCIZ pseudo-operations are supported.

Example:

```
E11>a 1000
001000  mov     #1022,r5        ;point at string
001004  tstb    @#177564        ;is port ready?
001010  bpl     1004            ;spin until it is
001012  movb    (r5)+,@#177566  ;send next character
001016  bne     1004            ;loop unless it was a NUL
001020  halt                    ;stop in that case
001022  .asciz  /Hello there/<15><12> ;test string
001040  ^C
E11>g 1000

Hello there

%HALT
R0/000000 R1/000000 R2/000000 R3/000000  CM=K PM=K PRIO=0
R4/000000 R5/001040 SP/000000 PC/001022  N=0 Z=1 V=0 C=0
001022  add     (r5)+,(r0)
E11>
```

ASSIGN *pdp11dev*: *pcdev*: [*switches*]

Assigns a physical PC device to emulate the specified PDP-11 serial or network device. See chapter 6 for information about serial devices, and chapter **??** for information about network devices.

pdp11dev is the name of the PDP-11 device name being created. It conforms to the syntax defined in section 1.7. The first two letters define the device type, and are generally the same as the PDP-11 operating system's name for that device.

pcdev is the name of the PC device which will emulate this PDP-11 device. Within each class of peripherals (serial, network), any PC device may be used to emulate any similar PDP-11 device.

The optional *switches* consist of keywords which begin with a forward slash character ("/"). Some switches take a value, which is preceded by a ":" or "=" character. The acceptable switches vary depending on the device names used, see the section describing the device for more information.

BOOT *ddu*: [/*switches*]

Boots the system from the specified disk, magnetic tape, DECtape, network device, or paper tape. The disk/tape must have been mounted with the `MOUNT` command, unless it is a real device attached through a bus adapter. The optional switch is an OS name. Useful switches 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 Note that the OS switches are meaningful only on block devices. If you prefer to set the time and date manually then the switch is not needed. RSX doesn't have a way to pass the time and date to a fresh monitor, so you'll have to use "F12" with a TIM command, or else use the TOY.TSK program (available on ftp.dbit.com) to read the TOY clock, which can be enabled on any emulated CPU type by putting `ASR` as the last item on the SET CPU command line. Recent RSX-11M-PLUS versions have a built in `TIM /SYN` command to do this, which works if the CPU type is PDP-11/93 or PDP-11/94.

There is also a `/HALT` switch, which tells E11 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 expression and displays the result in octal, decimal, hex, ASCII and radix-50. The operators are `* / + -`, unary `+ - ^C` (where "^C" means logical complement), and `( )`, with the usual precedence. `&` and `!` are the bitwise AND and OR operators, and have the same precedence as `*` and `+`, respectively. 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`". 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`".

E11 can accept expressions in most of the commands or switches which take numeric arguments. The default radix depends on the particular command or switch.

`C:` (etc.)

Changes the default PC disk drive to the one specified.

`CD` [*d:*][*path*]

Changes the current directory on the PC, or displays the current directory if none is given.

`DEFINE KEYPRESS` *keyname* = *statement*
`DEFINE KEYRELEASE` *keyname* = *statement*
`DEFINE MOUSEPRESS` *buttonname* = *statement*
`DEFINE MOUSERELEASE` *buttonname* = *statement*

Defines the action taken when the specified key or mouse button 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`). The mouse uses the same keyscript language, defined using the `DEFINE MOUSEPRESS/MOUSERELEASE` commands. 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 (MOUSEPRESS) and SHOW KEYRELEASE (MOUSERELEASE) commands. By default most keys do nothing on release and have no KEYRELEASE definition, except for the Alt, Ctrl, and Shift keys, and the left mouse button. Using the "E11.INI" initialization file, the user may easily redefine some or all of the keyboard and/or mouse as required. The keyscript language is described in Appendix A, and the list of key names that E11 accepts is in section A.3.

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. Flag names and meanings are described in section A.4.

DEASSIGN *ddu*:

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

DEPOSIT [*/switches*] *addr val₁ val₂ ...*

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.

DIRECTORY [*d:*][*path*][*wildcard*]

Displays a list of PC files that match the specified wildcard. The listing pauses at the end of each screen. Typing "Q" (or CTRL/C) ends the listing, carriage return advances by one line, and any other character advances to the next screen.

DISMOUNT *ddcu*:

Dismounts the specified mass storage device (see MOUNT).

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

With no switches, dumps PDP-11 memory to the specified PC 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 in the 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/EEPROM". 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.

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 or MAP 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 $[/switches]\ [d\!:\!path\backslash]filename[.\mathtt{PDP}]\ [s_1\!:\!e_1\ s_2\!:\!e_2\ \ldots\ s_n[\!:\![e_n]]\ ]$

With no switches, loads the specified PC 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*: [[*d*: *path*\]*filename*[.`LOG`]] [*switches*]

Logs the activity of the specified PDP-11 device to a file. The default filename extension is "`.LOG`". For character devices, all output sent to that device is logged. For all others, commands sent to the device are logged to the file, with a timestamp on each one. If no filename is specified, any existing log file for that device is closed. The available switches are as follows:

> `/APPEND` appends to an existing log file, rather than replacing it.
>
> `/LAST:`*n* limits the log file to only the last *n* lines of log data (or possibly fewer, if that many lines won't fit in the internal buffer).
>
> `/NOBUFFERING` disables multi-line buffering so that the internal log buffer is flushed to the file after each line-feed character. This causes increased disk activity (especially when combined with `/LAST`) but can be useful in diagnosing a crash.
>
> `/NOTIMESTAMPS` omits the timestamps on each command. This can be useful when using a file-compare utility on log files from two runs to look for differences. This switch has no effect on character devices, since they never log timestamps.

`LOG` *ddcu*: [[*d*: *path*\]*filename*[.`LOG`]] [*switches*]
(where *dd* is `LP` or `TT`.)

Logs all output to the specified character device in the specified file.

`LOG` *ddcu*: [[*d*: *path*\]*filename*[.`LOG`]] [*switches*]
(where *dd* is `CT`, `DC`, `DF`, `DK`, `DL`, `DM`, `DP`, `DS`, `DT`, `DU`, `DX`, `DY`, `II`, `KD`, `KP`, `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, DIP11A, M8644, KW11P, RXT11, TM03, TS11, TM11, or TMSCP controller, or the host file device or `HD_SYS.EXE` pseudo-controller, to the specified file. 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.

`LOG` `XE`*n*: [[*d*: *path*\]*filename*[.`LOG`]] [*switches*]
`LOG` `XH`*n*: [[*d*: *path*\]*filename*[.`LOG`]] [*switches*]
`LOG` `NI`*n*: [[*d*: *path*\]*filename*[.`LOG`]] [*switches*]

Controls logging of Ethernet events. If a filename is specified then the log file is opened. These devices support additional switches specifying which 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

MAP   57

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".

MAP [/*switches*] *addr*

Displays the physical address corresponding to the specified virtual address. The switches to specify the virtual address space are the same as the ones used with the EXAMINE command.

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.

pdp11dev is the name of the PDP-11 device name being created. It conforms to the syntax defined in section 1.7. The first two letters define the controller type, and are generally the same as the PDP-11 operating system's name for that device.

pcdev is the name of the PC device or filename to be used to emulate this particular PDP-11 drive. The possible names are described in chapters 4 and 5.

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. See chapters 4 and 5 for more information.

PRIMARY *n*

Switches the session number (in the range 1–12) being displayed on the primary (or only) video monitor. This command is equivalent to pressing Alt-F*n* on the keyboard, except that it may be issued from an initialization file or serial console without requiring that key combination to be typed manually on the PC keyboard.

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. Or the entire processor status word may be set at once with "`PS=`*xxxxxx*".

`SECONDARY` *n*

Switches the session number (in the range 1–12) being displayed on the secondary video monitor (if any). This command is equivalent to pressing Ctrl-F*n* on the keyboard, except that it may be issued from an initialization file or serial console without requiring that key combination to be typed manually on the PC keyboard.

`SET BREAK {`*nnn* `| NONE}`

Sets the octal value of an ASCII character that can be used in place of a serial BREAK signal to pop up an E11 prompt. For example, "`SET BREAK 020`" will cause any CTRL/P character typed on TT0: to bring up an E11 prompt. The default value is "NONE", meaning that all ASCII characters are passed through and only a serial BREAK, or a keyscript PROMPT command, will bring up the prompt. This command is intended for use in cases where the console (TT0:) has been ASSIGNed to a real serial port, connected to a terminal (or terminal program) that has difficulty generating BREAK signals.

`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 due to the oscillator frequency being slightly off, 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. 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 under some cases. `SHOW CPU` displays the current settings of all options.

E11 does not really 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, 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.

`SET CPU` options:

number	Set all CPU options to match PDP-11/*number* model. Recognized values are 03, 04, 05 (syn. 10), 15 (syn. 20), 23, 24, 34, 35 (syn. 40), 44, 45 (syn. 50 or 55), 53, 60, 70, 73, 74, 83, 84, 93, 94.
ASH31	J11 CPU bug with 31-bit shifts using ASH/ASHC.
ASR	KDJ11E additional status register (TOY clock etc.).
CB	Cache bypass bit in Unibus map registers.
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.
EAE	KE11A/B Extended Arithmetic Element.
EIS	Extended (integer) Instruction Set.
EIS16	EIS instructions with odd destination set condition codes based on 16-bit result.
FIS	Floating Instruction Set (FIS option for PDP-11/35,40 and LSI-11).
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.
LDUB	PDP-11/60 LDUB instruction (load microbreak register), as no-op.
LKS7	Bit 7 (monitor) of LKS clock status register is mechanized.
LOCKASRB	ASRB instruction uses interlocked memory access.
MARK	MARK instruction.
MAS	PDP-11/60 MAS instruction (maintenance alignment shift), as no-op.
MBR	PDP-11/70 microprogram break register (at (17)777770).
MED	PDP-11/60 MED instruction (maintenance examine/deposit), as no-op.
MFPT[=*n*]	MFPT instruction (returns *n* in R0).
MNS	PDP-11/60 MNS instruction (maintenance normalization shift), as no-op.
MPP	PDP-11/60 MPP instruction (maintenance partial product), as no-op.
MSEA	Memory system error address register (at (17)777740/2).
MSER	Memory system error register (at (17)777744).
MMTRAPS	PDP-11/45,55,70-style memory management traps, 3-bit ACF.
MMU	Memory management unit, MFPD/MFPI/MTPD/MTPI instructions.
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).
PDRA	Working "A" bit (bit 7) in MMU PDRs. PDP-11/45,50,55 and PDP-11/70 only.

PDRW	Working "W" bit (bit 6) in MMU PDRs. All PDP-11 MMUs have this bit, but if the PDP-11 operating system doesn't need it, disabling it may yield a small improvement in emulation speed.
PIRQ	PDP-11/45-style 7-level software interrupts.
PRIO	Multiple interrupt priority levels (as opposed to the PDP-11/03's single level).
PSWIO	PSW accessible from I/O space (at (17)777776).
QBUS	Q-bus exists (otherwise Unibus). Affects default models and default interrupt priorities of many emulated peripherals.
RTIRTT	RTI instruction works like RTT (early CPUs).
RTT	RTT instruction.
SIZE	PDP-11/70 system size registers (at (17)777760/2).
SOBSXT	SOB, SXT instructions.
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.
SWABV	SWAB instruction preserves V bit.
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.
USERMODE	User mode (or anything besides kernel mode).
UWIN	Unibus appears in address window at 17000000.
XOR	XOR instruction.

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 DELQA, DELUA, DL11, HOSTFILE (the host 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.

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). 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 native PC file I/O taking more than 16.67 ms (or 20 ms in 50 Hz clock mode) to complete.

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

Sets the frequency of the KW11L line clock (startup default is `60`).

`SET IDLE [NO]RELEASE`

Sets the behavior when the emulated CPU is idle, i.e. halted or executing a WAIT instruction. RELEASE means to ask the host OS to release the current time slice. This way E11 won't hog 100% of the host CPU's time for no reason. However, in OS/2's DOS box this is taken as a sign that E11 will be idle for some time so its priority gets reduced as a result, so this option should not be used in OS/2. NORELEASE means E11 should just keep looping until there's something for it to do.

`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. The swap is made upstream of the keyscript interpreter, so `CAPSLOCK` and `LCTRL` execute each other's `KEYPRESS` and `KEYRELEASE` scripts when the keys are swapped.

`SET MEMORY` *nnnn*

Sets the size of emulated PDP-11 memory. *nnnn* is the number of kilobytes of memory, anything from 8 up to 4088 (or the limit specified by the `/MEMORY:`*n* switch when E11 was started).

`SET PCLOG {ON | OFF}`

Enables or disables logging of the PC (program counter) register's value before each PDP-11 instruction fetch, at a slight speed penalty. Used for debugging PDP-11 code.

`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 SWITCH` *n*

Sets the value of the emulated switch register to *n*, as an expression with the same syntax as used by the `CALCULATE` command.

`SET THROTTLE [ON] [OFF] [DELAY=`*x*`] [INTERVAL=`*y*`]`

Enables or disables a simple throttle which can slow down the average emulated CPU speed, for compatibility with PDP-11 software which contains timing loops which are tuned for a particular PDP-11 CPU model.

Throttling is implemented by inserting a fixed delay at regular intervals in the instruction simulation. DELAY=x sets the length of this delay to x microseconds. INTERVAL=y sets the number of PDP-11 instructions that will be executed between delays. So there will be a delay of x microseconds, every y instruction fetches.

The ON keyword turns throttling on. Throttling is also turned on implicitly when the DELAY=x and/or INTERVAL=y parameter(s) is/are given.

The OFF keyword turns throttling off, and allows PDP-11 instruction emulation to proceed at the maximum possible speed. This is the default setting.

SET *pcdevcu*: ...

Sets parameters for the specified PC device, with the optional controller letter and/or line number given using the usual device name syntax, see section 1.7. SET commands for real PC devices must be issued before the first ASSIGN or MOUNT command which uses that PC device. The PC device names are the same as those used on the right hand side of an ASSIGN or MOUNT command.

The rest of the command line consists of keywords appropriate to that device type. The valid SET parameters for each PC device type are listed in the section that describes that device. Multiple keywords may be specified on the same SET command, they will be processed in left-to-right order.

SET *ddcu*: ...

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

BOOT=*option*	Selects whether incoming *MOP* boot frames will be honored. Specifies either the BOOT command parameters, or DISABLED (default) to disable network-initiated booting.
CSR=*nnnnnn*	Sets the base CSR address to *nnnnnn* octal.
CSR=DEFAULT	Sets the base CSR address back to the default address for this device (useful if it had been changed with an earlier SET command). SHOW commands will display "(D)" after an address which was chosen using DEFAULT.
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, unless

they are locked with the `LOCK` option. 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.

PRIORITY=n Sets the interrupt priority to n (4–7).

PRIORITY=DEFAULT   Sets the interrupt priority to DEC's default value for this controller type. In many cases, the original Unibus version of a controller uses priority 5, but the Q-bus equivalent uses priority 4, since early LSI-11 CPUs didn't have a multi-level priority system. When set to DEFAULT, the priority will normally be automatically reduced to 4 when emulating a Q-bus CPU. This is true even on devices which were available from DEC only for Unibus, since there may have been aftermarket Q-bus versions which used priority 4.

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

RH11   Sets this Massbus adapter (specified by controller letter, unit number is unimportant) 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=DEFAULT   Sets the base vector address back to the default address for this device (useful if it had been changed with an earlier `SET` command). `SHOW` commands will display "(D)" after an address which was chosen using `DEFAULT`.

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 BREAK

Shows the octal ASCII value of the character that can be used in place of a serial BREAK signal to bring up the E11 prompt. The default is NONE, meaning that only a genuine serial BREAK signal, or PROMPT keyscript command (normally generated with Shift/Enter), will bring up the prompt, and all ASCII characters are passed through.

SHOW CLOCK

Shows the current assumed frequency (in Hz) of the PC motherboard's 14.318 MHz clock, used to generate the KW11L/KW11P 50/60 Hz clock.

SHOW CPU

Shows emulated CPU type, number of emulated CPUs if greater than one, and emulated core size, along with breakdown of features, as well as the CPU type of the host processor.

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 IDLE

Shows the SET IDLE setting, either `RELEASE` (release the host CPU when idle) or `NORELEASE` (keep polling until there is something to do).

SHOW KEYPRESS *keyname*
SHOW KEYRELEASE *keyname*

Shows the keyscript currently bound to the keypress or keyrelease event for the specified key, if one is defined. See section A.3 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`. See section A.4 for a list of flag names.

SHOW MEMORY

Shows the amount of PC memory used by Ersatz-11, how much of that memory is emulated PDP-11 memory, and how much PC memory is free. Free memory must be available in order to use the `ASSIGN`, `DEFINE KEYPRESS`, `LOG`, or `MOUNT` commands.

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

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

SHOW MOUSEPRESS *keyname*
SHOW MOUSERELEASE *keyname*

Shows the keyscript currently bound to the press or release event for the specified mouse button, if one is defined. See section A.3 for key names.

SHOW PCLOG

Shows the log of PC (program counter) values from the last 64 PDP-11 instruction fetches, if enabled with `SET PCLOG ON`. Used for debugging PDP-11 code.

SHOW THREADS

Shows a list of all threads currently running in Ersatz-11, and a brief description of each one. This command is mainly intended as a support aid when contacting D Bit for help.

`SHOW THROTTLE`

Shows the current status of the emulated CPU speed throttle. See the `SET THROTTLE` command for more information.

`SHOW VERSIONS`

Shows the version numbers of Ersatz-11, the host 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, interrupt priority, 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 or interrupt priority means that parameter is set to "DEFAULT" and the displayed type or priority is the default based on the "SET CPU [NO]QBUS" setting. Similarly, "(F)" following a CSR or vector value means that parameter is set to "FLOATING" and the value given is the one currently in use based on the system configuration.

The default controller types, floating CSR or vector addresses, and default interrupt priorities are assigned specific values only for controllers that have actually been created with an `ASSIGN` or `MOUNT` command. If the controller does not currently exist then a defaulted controller type or interrupt priority will simply be listed as "DEFAULT" and floating addresses will be listed as "FLOATING".

The following PC devices support SHOW: `CON`*n*`:`.

Examples:

Show a simple PDP-11 device:
```
E11>show tt0:
TT0: CON1:
 CSR=777560(D) VECTOR=060(D) PRIORITY=4
```

Show an auto-configuring unmounted PDP-11 device:
```
E11>show dl0:
DLA0: NONEXISTENT
 DEFAULT CSR=17774400 VECTOR=160 PRIORITY=DEFAULT
```

Mount the device and show it under a Unibus emulation (PDP-11/70):
```
E11>mount dl0: \e11\rt11
E11>set cpu 70
E11>show dl0:
DLA0: RL02 READ/WRITE FILE "C:\E11\RT11.DSK"
 RL11(D) CSR=17774400(D) VECTOR=160(D) PRIORITY=5(D)
```

Switch to a Q-bus emulation (PDP-11/93) and show changes:
```
E11>set cpu 93
E11>show dl0:
DLA0: RL02 READ/WRITE FILE "C:\E11\RT11.DSK"
```

```
RLV12(D) CSR=17774400(D) VECTOR=160(D) PRIORITY=4(D)
```

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.

TYPE [*d:*][*path*]*filename*

Types the specified PC text file on the screen. The display pauses at the end of each screen. Typing "Q" (or CTRL/C) ends the listing, carriage return advances by one line, and any other character advances to the next screen.

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.

Appendix A

Keyboard Script Language

The script language used by E11's `DEFINE KEYPRESS` and `DEFINE KEYRELEASE` commands (and `DEFINE MOUSEPRESS` and `DEFINE MOUSERELEASE`) is powerful, yet very easy to use. Unlike some systems where keys can have only characters or strings bound to them, and mouse buttons can be reordered but not changed, E11 allows the user to attach a small script to each key so that more complicated operations can be defined. In particular, there's nothing special about the shift keys (Ctrl, Alt, Shift), they can be redefined to be data keys and vice versa.

When defining a new keyscript, it may be helpful to use a `SHOW` command to display the existing default keyscript for that key, or a similar one, and use it as the basis for writing the new keyscript. To make the keyboard send a character or string to the system, simply enclose that character or 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, and Shift keys).

`NONREPEATING` and `NOREPEATS` commands are used in many of the default keyscripts to emulate the autorepeat behavior of the VT100 keyboard precisely. `NONREPEATING` tells E11 that this keyscript should not be autorepeated even if the key is held down (like the VT100 Esc, Tab, and Return keys), 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, popping up the E11 command prompt, and switching the video display to show another session.

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.

A.1 Default keyboard layout

The keypad layout generated by E11's predefined keyscripts 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. Similarly, the CTRL characters on the main keyboard are in the same places as on a VT100, including the non-alphabetic ones (e.g. CTRL/SPACE generates NUL, CTRL/~generates RS, etc.). The "backspace" key generates DEL because that's what DEC operating systems normally use. Use CTRL/H to get a backspace character. Line feed is CTRL/J, and on 104-key keyboards it's

also the "context menu" key next to the right-hand Ctrl key.

The keypad digits and "." key work as marked (you must be in Num Lock mode to get this on 84-key keyboard, it 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. CVT *Avant Prime* and *Avant Stellar*, and Northgate *Omnikey 102* keyboards (which were all apparently designed by the same person) 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, so F8 must be used instead. 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 VT100 keypad hyphen, comma, period, and ENTER keys are also available as the F6, F8, F9, and F10 keys (see below). On an IBM AT 84-key keyboard (which has the F-keys stacked vertically) this gives the same layout as the right-hand edge of a VT100 keypad.

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.

CON1:	CON2:	CON3:	CON4:

CON5:	CON6: _	CON7:	CON8: ,

CON9: .	CON10: Enter	CON11:	CON12: Date

Default F-keys on 101- and 104-key keyboards

NumLock PF1	PF2	PF3	PF4
7	8	9	
4	5	6	−
1	2	3	
0		.	Enter

Default keypad layout on 101- and 104-key keyboards

Esc PF1	NumLock PF2	ScrLock PF3	SysReq PF4
7	8 ↑	9	−
4 ←	5	6 →	,
1	2 ↓	3	
0		.	Enter

Default keypad layout on IBM AT 84-key keyboards

CON1:	CON2:
CON3:	CON4:
CON5:	CON6: _
CON7:	CON8: ,
CON9: .	CON10: Enter

F-keys on IBM AT keyboards

A.2   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*)), all 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. See section A.4 for a list of flag names.

COPY

Copies the currently selected text (see `SELECT` etc.) into the paste buffer, where it can be be pasted with `PASTE`.

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. `GETTIME` should be executed once before any sequence of commands that sends the various date/time fields. This is done explicitly, instead of having each individual command get the time itself before sending its field, in order to ensure that all the fields are coordinated and don't contain mismatched data if the clock rolls over while the date/time string is being composed. 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 section A.4 for a list of flag names), 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 actually be multiple statements separated by ":" or "\" characters) following the THEN is executed, and after that execution then skips to after the ENDIF. Otherwise the expression following the ELSEIF (if any) is similarly tested, followed by any successive ELSEIFs if the first fails. Finally the ELSE clause (if any) is executed if no (ELSE)IF was true. This is pretty much the same as FORTRAN-77 or any language with block-structured IFs.

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. This has the effect of converting it to lower or upper case as appropriate, and converting it to the correct control character if CTRL is true, assuming it's a US-ASCII alphabetic character.

```
MILLISECOND3
```

Sends the 3-digit millisecond (000–999) as read by GETTIME.

```
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.

`PASTE`

Pastes the text from the paste buffer (as copied with `COPY`), as if it were typed at the keyboard.

`PRESS` *keyname*

Executes the "keypress" script for the specified key, i.e. the script defined by `DEFINE KEYPRESS` for that key.

`PRIMARY` *n*

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

`PROMPT`

Pops up an E11 command prompt. The primary video display is switched to `TT0:`'s screen if it is not already there.

`RELEASE` *keyname*

Executes the "keyrelease" script for the specified key, i.e. the script defined by `DEFINE KEYRELEASE` for that 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). This is useful only on systems with two video displays, and performs no operation on systems with only one.

SELECT

Begin selecting text using the mouse. Text on the screen is shown in reverse video starting at the current mouse location, and extending to wherever the mouse moves, until a `COPY` command is executed (typically by the `MOUSERELEASE` keyscript for the same button which executed the `SELECT`) which copies the highlighted text into the paste buffer. Selection is aborted (and the text being selected is un-highlighted) if the PDP-11 sends more data to the screen while selection is in progress.

SELECTWORD

Select the word at the current mouse location.

SELECTLINE

Select the entire line (including a carriage return character at the end) at the current mouse location.

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

Sets a read/write flag. See section A.4 for a list of flag names. 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 disabled users. Note that the current key (the one whose keyscript contains this "`SET ... FOR` *n*" command) 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
```

TICK

Sends the timer tick number as read by `GETTIME`. Normally this is a 2-digit number, 00–49 for a 50 Hz clock or 00–59 for a 60 Hz clock, but the actual number of digits is selected to fit the maximum possible timer tick number, which is one less than the `SET HERTZ` setting.

TOGGLE *flag*

Toggles a read/write flag. See section A.4 for a list of flag names.

`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`.

A.3   Key names

This section defines the key names used for `DEFINE KEYPRESS`, `DEFINE KEYRELEASE` commands, and `PRESS`, `RELEASE` statements. Keycap 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 considered an error to bind keyscripts to them even when only an 84-key AT keyboard is present, but keyscripts for keys that are missing will never be executed.

Key	Description
`'`	[" / '] key
`*`	`*` key on keypad, or [PrScr / *] key if present
`,`	[< / ,] key
`-`	[_ / -] key
`.`	[> / .] key
`/`	[? / /] key
`0`–`9`	Numeric keys (top row of main keyboard)
`;`	[: / ;] key
`=`	[+ / =] key
`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` are 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, so it would not be useful to try to redefine this key as a shift key
PGDN	PgDn (EKB only)
PGUP	PgUp (EKB only)
POWER	Power (newer keyboards)
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
SLEEP	Sleep (newer keyboards)
SPACE	Space bar
SYSREQ	Sys Req (84-key keyboard only)
TAB	Tab
UARROW	Up arrow key (EKB only)
WAKE	Wake (newer keyboards)
[	`{` `[`
\	`|` `\`
]	`}` `]`
`	`~` `` ` ``

In addition to the above, the following keywords define keys that don't exist on most keyboards, for completeness in case they are useful on special-purpose keyboards:

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 (syn. for POWER)
KEY5F	Sends scan code 5F hex (syn. for SLEEP)

Mouse button names may be used only in `DEFINE MOUSEPRESS` and `DEFINE MOUSERELEASE` commands:

LEFT	Left mouse button
MIDDLE	Middle mouse button (if any)
RIGHT	Right mouse button
X1	Extra button on some mice
X2	Extra button on some mice

The double- and triple-click events are used in `DEFINE MOUSEPRESS` only:

DOUBLELEFT	Double-click on left mouse button
DOUBLEMIDDLE	Double-click on middle mouse button (if any)
DOUBLERIGHT	Double-click on right mouse button
DOUBLEX1	Double-click on X1
DOUBLEX2	Double-click on X2
TRIPLELEFT	Triple-click on left mouse button
TRIPLEMIDDLE	Triple-click on middle mouse button (if any)
TRIPLERIGHT	Triple-click on right mouse button
TRIPLEX1	Triple-click on X1
TRIPLEX2	Triple-click on X2

A.4 Flags

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

A.4.1 Read/write flags

Read/write flags can be tested or modified by keyscripts. Their values may be used in `IF` expressions or `DEFINE LED` commands, or they may be changed using `SET`, `CLEAR`, and `TOGGLE` statements. The descriptions below are only defaults used by E11's initial keyboard definitions, the user is free to redefine them at will.

CAPS	Caps Lock state
EDITBACKUP	EDT/KED editor direction, maintained by default keyscripts for keypad 4 and 5 (true=BACKUP, false=ADVANCE)

FLAG*n*	User-defined flags (*n*=1–4), reserved for user keyscripts
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

A.4.2   Read-only flags

Read-only flags can be tested by keyscripts but not modified. This means they may be used in `IF` expressions or
`DEFINE LED` commands, but not in `SET`, `CLEAR` or `TOGGLE` commands. The values are maintained and updated by
E11 itself.

ALT	OR of `LALT` and `RALT`
APPKEYPAD	Applications keypad mode (ESC =)
CHARSETA	Character set A (UK) is currently selected
CHARSETB	Character set B (US) is currently selected
CHARSET0	Character set 0 (graphics) is currently selected
CHARSET1	Character 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)

Appendix B

Debugging Features

Ersatz-11 provides a full assortment of debugging commands. These can be useful for PDP-11 program development, and can also be invaluable for troubleshooting configuration problems, or for providing detailed information to D Bit when reporting a problem in Ersatz-11 itself.

Real PDP-11s all provide some form of programmer's console. This can be either a traditional binary "switches and lights" front panel, an octal keypad and LED display, a simple ASCII console emulator program running out of ROM, or the "Micro-ODT" program implemented in microcode in systems based on the LSI-11, DCF11, and DCJ11 chip sets. Regardless of the implementation, the programmer's console provides some way to examine and deposit memory, inspect and modify the general registers and processor status word, and control program execution.

E11 provides all of this functionality, as well as many more features, using the interactive "`E11>`" command prompt, which can be popped up at any time by pressing Shift-Enter. Most of the commands related to program debugging can be abbreviated to one letter, to save typing, and the user can further streamline the debugging process by binding often-used commands to a single keystroke using `DEFINE KEYPRESS` commands.

Debugging commands:	
`ASSEMBLE`	Assemble PDP-11 code into memory
`C=` (etc.)	Set condition flag value
`CALCULATE`	Evaluate an expression (abbreviation = "`&`")
`CM=` (etc.)	Set current/previous processor mode
`DEPOSIT`	Deposit into memory or I/O page registers
`DUMP`	Dump memory to a file
`EXAMINE`	Examine memory or I/O page registers
`FPREGISTER`	Display/change floating point registers
`GO`	Start program execution
`HALT`	Halt program execution
`INITIALIZE`	Initialize CPU (like RESET instruction)
`LIST`	List disassembled PDP-11 code
`LOAD`	Load a binary file into memory
`LOG`	Log disk/tape/Ethernet controller commands to a file
`MAP`	Translate virtual address to physical address
`PRIO=`	Set processor priority
`PROCEED`	Proceed with execution, with an optional breakpoint
`PS=`	Set processor status word
`R0=` (etc.)	Set general register value
`REGISTER`	Display general registers and PSW fields
`SET DISPLAY`	Set up hardware display register
`SET PCLOG`	Enable/disable PC logging on each fetch
`SET SWITCH`	Set up switch register (real or virtual)
`SHOW CSR`	Display name of any I/O page register
`SHOW DISPLAY`	Inspect value of display register
`SHOW MEMORY`	Show E11 memory usage
`SHOW MMU`	Display MMU maps
`SHOW VERSIONS`	Show version numbers of E11 and host OS
`STEP`	Execute one (or several) single program step

B.1   Displaying and modifying memory

Memory may be displayed or modified one word at a time using the `EXAMINE` and `DEPOSIT` commands. Both can accept switches that define the virtual address space to use. If no switches are specified, the default is to use the same space as the previous `EXAMINE` or `DEPOSIT` command. Any switch used on either command sets the default for both commands.

These commands can also operate on multiple memory words. Normally when only one address is specified, the `EXAMINE` command will display just one word, but if a starting and ending address is specified, that entire range, no matter how large, is displayed on the terminal. If an `EXAMINE` command is given with no address whatever, it displays eight words starting at the word following the last `EXAMINE` command. Multiple words can be entered into memory by simply entering more than one number following the address in a `DEPOSIT` command. The numbers are entered into consecutive words of memory starting at the specified address. For convenience, these commands may be abbreviated to `E` and `D`.

B.2   Assembly and disassembly

The `ASSEMBLE` and `LIST` commands display or modify memory in very much the same way as the `EXAMINE` and `DEPOSIT` commands, except that they accept and display assembly language source instead of octal data. Both commands accept optional switches to set the virtual address space to use, and if no switches are specified the default is to use the same virtual address space from the previous `ASSEMBLE` or `LIST` command.

The `ASSEMBLE` command starts the assembler at the specified address, or if no address is specified the default is to continue assembly at the next address following the last line assembled by the previous `ASSEMBLE` command. E11 prompts for each line, giving the address where it will be entered into memory. Entering a blank line (or typing CTRL/C) returns to the E11 command prompt.

The `LIST` command lists eight lines of code starting at the specified address, or if no address is specified then eight lines of code are disassembled following the last line displayed by the previous `LIST` command or `REGISTER` display. Two addresses can be given, to produce a disassembly of all code in that range of addresses.

B.3   Registers

The PDP-11 registers may be displayed at any time using a `REGISTER` command (`R` for short). This displays all of the registers and PSW flags at once. Individual registers may be displayed using commands like "`& R3`". Any value that is displayed by the `REGISTER` command may be set using the same keywords as is used in the display. For example, if the register dump includes "`C=1`" and "`PM=U`" to indicate that the carry flag is set and the previous mode is "user", it is possible to clear the carry flag by typing "`C=0`", or change the previous mode to "supervisor" by typing "`PM=S`". And of course the general registers may be changed with commands such as "`R3=100`", or even "`SP=SP+4`".

The FP11 floating point processor state may be inspected using the `FPREGISTER` command. However it is less flexible than the regular `REGISTER` command, all register values are displayed and set using octal numbers.

The `SET PCLOG ON` command can be used to log the PC value before each instruction fetch, at a slight speed penalty. The last 64 values may be shown at any time using `SHOW PCLOG`. This can be very useful for tracing the events leading up to a trap or halt.

B.4   Breakpoints and single-stepping

The `PROCEED` command takes an optional numeric argument, which is a PC value. PDP-11 instruction execution will halt whenever that address is reached. This is done using address comparison, rather than by depositing anything into memory (regular PDP-11 debuggers do breakpoints by inserting a `BPT` instruction into memory), which means it works even if the memory contents are overwritten before the breakpoint is reached.

`STEP` allows proceeding one instruction at a time. If a numeric argument is specified, it gives the number of steps to perform, with a default of one step. This command can be abbreviated to `S` so that it may be typed quickly. For even greater convenience, a key may be redefined to enter this command with a single keystroke. For example:

```
def key kpplus = 'step'+chr$(13)
```

This will cause a step to be executed every time the keypad `+` key is pressed.

The `GO` command starts execution at full speed, with no breakpoints or single stepping. An optional starting address may be given. If it is omitted, execution continues at the current PC value. Unlike some versions of ODT, E11 does not issue an implied INIT pulse with either the `GO` or `PROCEED` command. If necessary the system may be initialized, including all emulated devices and bus adapters, using a separate `INITIALIZE` command.

B.5   Memory mapping

E11 can display the current status of the memory management unit at any time using the `SHOW MMU` command. There are two optional arguments, the mode (`KERNEL`, `SUPERVISOR`, or `USER`) and the space (`INSTRUCTION` or `DATA`). Both may be abbreviated to one letter, and the defaults are `KERNEL INSTRUCTION`. The display for each page gives the starting virtual address, starting physical address, block limits, access, and the A (accessed) and W (written) dirty bits. Also, the current values of MMR0 through MMR3 are displayed (MMR3 is omitted if the current CPU emulation doesn't include MMR3).

The `MAP` command may be used to compute a physical address, given a virtual address. It takes the same switches as the `EXAMINE` and `DEPOSIT` commands, to give the mode and space parameters for the virtual address. It displays the resulting 22-bit physical address, as if it had been processed by the MMU during a PDP-11 instruction. However, like the `EXAMINE` and `DEPOSIT` commands, the MMU's status is not affected by this command, so the `A` bit will not be modified in this page's PDR, and no MMU abort will occur if the page is marked "no access" or the address is outside of the range of valid blocks in that page.

The `SHOW CSR` command displays a one-line description of an emulated CSR on the I/O page, given its address. It works only on devices which are currently configured.

B.6   Device logging

Although the controller `LOG` commands are provided mainly as an aid to reporting Ersatz-11 problems to D Bit, they can be invaluable to users who are debugging device drivers or boot blocks. When logging is enabled for a given disk, tape, or network controller, every command issued to that controller by the PDP-11 is recorded in the log file with a time stamp. Other parameters such as buffer addresses and lengths, unit numbers, disk addresses, and tape record lengths, are recorded too, and for some devices the command's completion status is saved as well. For Ethernet devices, switches on the `LOG` command line give independent control over logging of host commands, transmitted packets, and received packets.

B.7   Loading and dumping memory

The `LOAD` and `DUMP` commands may be used to transfer between PDP-11 memory and files on the host system. An optional series of address ranges may be specified to do scatter/gather transfers. If no addresses are specified, the transfer starts at the beginning of memory and goes until the end of the files (for `LOAD`) or the end of memory (for `DUMP`).

B.8   Switch and display registers

The `SET SWITCH` and `SHOW DISPLAY` commands give access to emulated switch and display registers, which work the same way as those on the binary front panels on early PDP-11 models. The switch/display register is located at address `17777570` as long as a `SET CPU SR` command, or some other `SET CPU` *nn* command which includes `SR`, has been issued. This provides a crude one-word I/O register which can be accessed by a program running in kernel mode, for things like displaying checkpoint information during debugging.

Simple hardware can be attached to E11's switch register and display register emulations using `SET` commands. A display register which connects to an LPT port is available from D Bit as a bare PC board.

Appendix C

Installable Plug-ins

Ersatz-11 has a built-in linking loader, invoked by the `INSTALL` command, which allows "plug-in" modules to be integrated with E11 at run time. This makes it possible to implement custom devices and/or instructions using 80x86 assembly language, Watcom C, or Digital Mars C, without requiring a custom version of Ersatz-11 or access to its source code.

A plug-in module is in the 32-bit "portable executable" .DLL format, which is the standard executable file format for Windows NT and Windows 95 and later, and is supported by standard linkers. Ersatz-11 provides no-op stubs for a number of Win32 calls, so that the regular Windows library start-up module supplied with Watcom C or Digital Mars C will run unmodified. The supplied stubs are only enough to start up the C code without errors, but they don't provide Windows system services. In particular, anything which uses `malloc()` will not work (use `GetMemory()` instead).

There is no difference in the file format or command syntax needed to load an instruction emulation plug-in, as compared to a device emulation plug-in. A plug-in's purpose is defined by the calls it makes. Also, there is no need to split multiple device or instruction emulations into separate modules, they can be combined into one.

C.1   Calling conventions

As is traditional with Win32 DLLs, Ersatz-11 plug-in modules must use `__stdcall` calling conventions when calling API entry points, and these conventions are also used when Ersatz-11 is calling user code. These are as follows:

- Caller pushes arguments right to left, before making NEAR call

- Callee pops arguments upon return, using `RET` n instruction

- EBX, EBP, ESI, EDI and all segment registers are preserved by callee

- DF is 0 on entry and must be preserved by callee

- Symbol names are case-sensitive but are otherwise used as-is (no underscore prefix or `@`nn suffix is added, despite linker-induced illusions to the contrary)

- Return value (if any) is passed in EAX (zero- or sign-extended as needed)

C.2   Entry conditions

After a plug-in module is loaded and linked, a call is made to its entry address, with the following parameters:

`int __stdcall DllMain(unsigned long hinstDLL,unsigned long fdwReason,void *lpvReserved)`

This is compatible with the standard Win32 DLL entry point (and this is the name of the user-written routine which the C library startup code will call after its own initialization, with the same parameters).

`hinstDLL` is a unique number assigned when the module is loaded. It has no purpose in E11, but is part of the Win32 library startup so it is included for compatibility.

`fdwReason` is the reason for the call, either 1 (`DLL_PROCESS_ATTACH`) if the library is being loaded, or 0 (`DLL_PROCESS_DETACH`) if it is being unloaded.

`lpvReserved` points at the command line when loading, and contains NULL when unloading. The command line is whatever follows the filename in the `INSTALL` command, if anything, in DOS form (i.e. a length byte followed by a CR-terminated string). This can be useful for passing CSR addresses (etc.) or other configuration information to the installed module. For example, "INSTALL FOO.DLL CSR=164000" will result in `lpvReserved` pointing at the following (in E11's own scratch memory, which will be re-used after the initialization routine returns):

```
        .BYTE   11.              ;length of string
        .ASCII  / CSR=164000/    ;the string itself
        .BYTE   15               ;always ends with carriage return
```

`DllMain()` returns 1 (`TRUE`) for success, or 0 (`FALSE`) for failure, in which case E11 will issue an error message.

C.3   Exit conditions

Since the loading of a plug-in module, and whatever calls it makes, are completely unrelated in E11, E11 has no idea what resources have been allocated by a particular module so it doesn't automatically free them when that module exits (e.g. due to a QUIT command). Therefore, each module must keep track of its resources and free them when they are no longer needed. The library shutdown call to `DllMain()` is a good time to release any unneeded resources.

C.4   Building plug-in modules

All C source files should include a `#include ''e11.h''` line at the beginning. The `E11.H` header file and `E11.LIB` import library are available from D Bit.

To build a DLL with Watcom C (available from www.openwatcom.org, runs under MS-DOS or Windows or OS/2), compile and link as follows (e.g. to build FOO.DLL from FOO.C):

```
wcc386 -bt=nt -bd -s foo.c
wlink @foo
```

This requires a `FOO.LNK` linker script as follows:

```
system nt_dll
name foo.dll
file foo
library e11.lib
```

To build a DLL with Digital Mars C (available from www.digitalmars.com, runs under Windows), compile and link as follows (e.g. to build FOO.DLL from FOO.C):

```
dmc -mn -WD foo.c e11.lib kernel32.lib
```

C.5   Entry points

This section defines all of the callable Ersatz-11 entry points. They are grouped roughly by function. Most of these calls may be safely made only from main program level, i.e. *not* from an interrupt service routine, signal handler, or non-mainline thread. Exceptions are noted in the entry point descriptions below.

An include file and import library for Watcom C or Digital Mars C are available from D Bit. Defined data types are as follows:

`E11HANDLE` is a 32-bit token used by E11 internally.
`word` is an unsigned 16-bit value.
`dword` is an unsigned 32-bit value.

Pointers are normally 32-bit flat addresses. DLLs are considered to be trusted, so E11 does not check parameter values and crashes are possible if bad numbers are passed.

C.5.1   Ersatz-11 environment

This section describes calls which obtain information about the environment where the plug-in module is running.

unsigned long E11Version(void)

Returns the Ersatz-11 version number, as follows:

```
MAJOR*256.+MINOR
```

MAJOR is the major version number as an integer, and MINOR is the minor version number in hundredths (0–99).

void *GetItem(const char *itemname)

Gets a pointer to a constant data structure or string, which corresponds to the supplied .ASCIZ item name. If the item is undefined, then NULL is returned. This allows checking for arbitrary E11 features which may be present in particular versions, without needing specific knowledge of ranges of Ersatz-11 versions.

C.5.2   I/O device emulation

The most likely purpose for a plug-in module is to emulate a peripheral device. In order to do this, it must be able to intercept reads and writes to its I/O page registers, and must be able to interrupt the emulated PDP-11 and/or perform DMA to and from PDP-11 memory. The next few calls accomplish these functions.

```
E11HANDLE GetCSRBlock(dword first,dword count,word (*dati)(),void (*dato)(),
        void (*init)())
```

Allocates a block of control/status register locations on the I/O page, to be emulated by this module. The return value is zero on failure, otherwise it's a handle which may later be passed to `RetCSRBlock()` to release the block of addresses. `first` is the absolute address of the first CSR to allocate, and is represented as a 22-bit even number, even if the current CPU emulation is set for a smaller address space. `count` is the count of registers, i.e. the number of words of I/O space to allocate starting at that address. `dati`, `dato`, and `init` are pointers to functions which E11 will call according to what the PDP-11 code does:

```
word __stdcall dati(dword addr)
```

Called whenever the PDP-11, or console `EXAMINE` command etc., reads a byte or word on the I/O page which is within the range passed to `GetCSRBlock()`. This is equivalent to a Unibus "DATI" bus cycle. PDP-11 busses do not distinguish between byte and word read cycles, a word is always read and if it's a byte instruction then the CPU chooses the proper byte out of that word. `addr` is the absolute 22-bit address of the I/O page location to be read, and the function's return value is the 16-bit word to be passed back to the PDP-11.

```
void __stdcall dato(dword addr,dword value,dword datob)
```

As above, but called when a byte or word is being written to an I/O page register in the range covered by the `GetCSRBlock()` call. This is equivalent to a Unibus "DATO" or "DATOB" bus cycle. `addr` is the absolute 22-bit I/O page address as above. `value` is the 16-bit word to be written, or if it's a byte write, it's two side-by-side copies of the 8-bit byte being written. `datob` distinguishes between word and byte writes, 0 means word (DATO), 1 means byte (DATOB).

```
void __stdcall init(dword addr)
```

Called on every `RESET` instruction, or on console `INITIALIZE` commands etc. This is equivalent to a Unibus "INIT" pulse. The device emulation code should perform whatever initialization would be appropriate for a bus reset. `addr` is the base 22-bit address that was passed to `GetCSRBlock()`, and is supplied in case the same handler routine is used for more than one device, so that it can tell them apart.

```
void RetCSRBlock(E11HANDLE handle)
```

Returns a block of I/O page CSR registers, which had previously been allocated using `GetCSRBlock()`. This is normally done as part of the cleanup in the module's `DllMain()` exit handling. Future accesses to the I/O page addresses covered by the `GetCSRBlock()` call will cause emulated bus timeouts.

`E11HANDLE GetIntQel(dword level,dword rank,dword vec)`

Allocates an interrupt queue element. A module which needs to deliver emulated interrupts to the PDP-11 must allocate one interrupt queue element for every possible interrupt that can be outstanding at one time. This normally means one queue element for each interrupt vector that the device uses. This function returns zero on failure, otherwise the return value is the queue element's handle, which is a number used to identify the queue element in other calls to E11.

`level` is the interrupt priority level, which must be in the range 4–7. This corresponds to the Unibus BR and BG lines (or their Q-bus equivalents), and is the same as the priority that would be set using a priority jumper plug on a typical Unibus peripheral card.

`rank` is the daisy chain rank within a given priority level, and indicates the device's distance from the CPU on this priority level. Since the emulator has no physical daisy chain, it's generally not possible to know what other devices this one should go between, so the daisy chain rank is just a rough guess. The value 0 is reserved for the PIRQ feature that's built into some CPU models, and E11 generally uses 1 for the 50/60 Hz clock and the console SLU ports, since in most systems these are physically located on the CPU card, or immediately next to the CPU. So most devices will use values 2 and up. If two devices interrupt at the same priority level and with the same daisy chain rank, E11's policy is to acknowledge them in the order that they occurred.

Finally, `vec` is the vector address through which the interrupt will be taken once it is acknowledged. It must be a multiple of four. The vector need not be valid except when an interrupt has actually been enqueued using `EnqueueInt()`, so if the `EnqueueInt()` completion routine will set the vector at bus grant time, a junk value such as -1 may be used here.

`void RetIntQel(E11HANDLE handle)`

Returns an interrupt queue element that had been obtained earlier using `GetIntQel()`. This call is normally made as part of the `DllMain()` exit processing. If an interrupt is enqueued using this queue element, it is dequeued before the element is freed.

`void EnqueueInt(E11HANDLE handle,int (*crtn)(dword),dword parm)`

Enqueues an interrupt request on a previously allocated interrupt queue element. This is equivalent to a Unibus device asserting its "BR" line and waiting for service. The interrupt is only enqueued, and has not yet begun processing on return, because that can only happen when the emulated PDP-11 is between instructions. Note that like most E11 entry points, this one may be called only from main program level, not from a hardware interrupt service routine or signal handler. The `EnqueueFork()` call provides access to E11's internal "fork" queue, which allows scheduling a call in the near future to a user subroutine at main program level, from anywhere. This call may be used in an interrupt service routine or signal handler to schedule a call to a separate user-supplied routine which makes the `EnqueueInt()` call.

Simple devices can simply pass NULL as the `crtn` parameter, and any value for the `parm` parameter. In this case the interrupt will quietly occur on its own whenever the PDP-11 lowers its priority and begins interrupt service, and the vector address that was set in the `GetIntQel()` call will be used, unless it is canceled by a `DequeueInt()` call before then.

Devices which need to be informed when an interrupt is actually being serviced, and/or need to wait until the last moment to choose an interrupt vector, should pass a function pointer as the `crtn` parameter. This is a completion routine, which is called like this:

```
    int __stdcall crtn(dword parm)
```

This routine is called just as the emulated PDP-11 is acknowledging the interrupt and preparing to fetch the interrupt service routine's address from the vector. `parm` is the same value that was passed to `EnqueueInt()`. It may be any 32-bit value and is there so that the same completion routine may be used for more than one device. The routine should do whatever processing its emulated device would do at bus grant time. Its return value is the actual vector address to interrupt through (which need not be the same as the value passed to `GetIntQel()`), or -1 if the interrupt should not be taken after all (a Unibus "passive release").

Very few devices need this routine. An example would be certain models of the 50/60 Hz line clock, which clear the "monitor" flag in bit 7 of the CSR whenever the interrupt request is granted. If a NULL pointer is passed instead of a completion routine address, E11 will just interrupt as usual using the vector address that was already set.

void DequeueInt(E11HANDLE handle)

Dequeues an interrupt request which was scheduled using the `EnqueueInt()` call. This is generally done on bus resets, or any time the emulated device's status is changed so that a previously enqueued interrupt will not happen after all. `handle` is the handle of the interrupt queue element to dequeue, as returned from `GetIntQel()`. It is perfectly safe to call this routine if no interrupt was actually enqueued, in this case it will simply return without doing anything. However the `handle` parameter must be valid, since unpredictable behavior will result if it is not.

long AbsTransfer(dword func,dword pdp11addr,void *x86addr,dword len)

long MapTransfer(dword func,dword pdp11addr,void *x86addr,dword len)

Performs DMA between 80x86 memory and emulated PDP-11 memory. `func` gives the direction of the transfer, either `TOPDP` to transfer from the host to the PDP-11, `TOHOST` to transfer from the PDP-11 to the host, or `COMPARE` to compare the two. `pdp11addr` gives the PDP-11 buffer address, either an 18-bit Unibus address (for `MapTransfer()`, which goes through the Unibus map if it is present and enabled), or an absolute 22-bit address (for `AbsTransfer()`, which uses absolute addressing like Q22-bus or the RH70). `x86addr` is the 80x86 buffer address. `len` is the length of the transfer. The return value is zero on success, or the number of bytes *not* transferred (if stopped due to a bus timeout), or the two's complement of this number (if stopped due to a compare mismatch).

C.5.3   PDP-11 instruction emulation

A plug-in module may be used to add custom instructions to the PDP-11 architecture, or provide modified versions of existing ones. This is similar to what was possible using the hardware user microcode options on the LSI-11 and PDP-11/60 machines, but it is much more powerful since the code doing the emulation has access to the entire host computer and operating system, rather than just a PDP-11 micro-machine.

E11HANDLE GetOpcode(dword first,dword count,void (*exec)(),void (*init)())

Allocates a block of PDP-11 opcodes to be emulated by this module, instead of by Ersatz-11 itself. The return value is zero on failure, otherwise it's a unique handle which may later be passed to `RetOpcode()` when the instruction handler is unloaded. If the range of opcodes conflicts with existing PDP-11 instructions, or other loaded plug-ins, whoever requested each opcode last will be the one called when it gets executed by the PDP-11.

first is the first opcode value to be intercepted and passed to this handler. **count** is the number of opcodes to be intercepted, starting with that value. **exec** and **init** are subroutines which E11 will call when appropriate, as follows:

```
void __stdcall exec(dword opcode)
```

This routine is called by E11 any time an opcode is executed which is in the range from **first** to **first+count**-1. **opcode** is the actual opcode used, and may be used for the handler's internal dispatch table, **switch()** statement, etc. This routine does whatever processing is required to execute the instruction, and uses other calls (defined below) to read and write PDP-11 registers and memory space.

```
void __stdcall init(void)
```

This routine is called any time the PDP-11 is initialized, through a **RESET** instruction, **INITIALIZE** command etc.

void RetOpcode(E11HANDLE handle)

Returns a block of PDP-11 opcodes, previously allocated using **GetOpcode()**. This call is a normal part of **DllMain()** exit processing.

word GetReg(dword regnum)

Reads the value of one of the eight PDP-11 registers, in the current register set. **regnum** gives the register value, 0–5 for R0–R5, 6 for the SP, and 7 for the PC. The return value is the 16-bit register value.

void SetReg(dword regnum,dword value)

Sets the value of a PDP-11 register. **regnum** is the register number (0–7, the same as for **GetReg()**), and **value** has the value to store there in its low 16 bits.

dword GetPSW(void)

Reads the zero-extended value of the processor status word.

void SetPSW(dword value)

Sets the value of the processor status word to **value**.

long Peek(dword space,dword pdp11addr)

Reads a word from a PDP-11 virtual address space. **space** selects which space to use, and is one of the following:

0	KD	kernel data space
1	KI	kernel instruction space
2	SD	supervisor data space
3	SI	supervisor instruction space
6	UD	user data space
7	UI	user instruction space
8	D	current data space
9	I	current instruction space
10	PD	previous data space
11	PI	previous instruction space

pdp11addr is the 16-bit virtual address to read within that space. On success, the return value is the data word, in the range 0–177777 octal. If the read fails due to an odd address, MMU error, or bus timeout, -1 is returned. Note that the return value is a signed 32-bit number, so that the -1 error indication can be distinguished from a valid data value of 177777 octal.

```
long PeekB(dword space,dword pdp11addr)
```

Byte version of Peek(), reads a byte from a PDP-11 virtual address space. Return value is the 8-bit byte sign-extended to a 32-bit number, or else -1 the in case of an error.

```
int Poke(dword space,dword pdp11addr,dword data)
```

Writes a word into a PDP-11 virtual address space. space selects which space (see table above), pdp11addr is the 16-bit virtual address in that space, and the low 16 bits of data are the word to write to that address. The return value is 0 on success, or -1 on failure.

```
int PokeB(dword space,dword pdp11addr,dword data)
```

Writes a byte into a PDP-11 virtual address space. The parameters and return value are the same as for Poke(), except that only the low 8 bits of the data parameter are used.

```
long PeekCD(dword pdp11addr)
```

```
long PeekBCD(dword pdp11addr)
```

```
int PokeCD(dword pdp11addr,dword data)
```

```
int PokeBCD(dword pdp11addr,dword data)
```

Special versions of Peek(), PeekB(), Poke(), and PokeB(), which access the current data space. Execution is slightly faster than the equivalent Peek(D,...) (etc.) calls.

```
long PeekPhys(dword pdp11addr)
```

```
long PeekBPhys(dword pdp11addr)
```

```
int PokePhys(dword pdp11addr,word data)
```

```
int PokeBPhys(dword pdp11addr,word data)
```

Special versions of `Peek()`, `PeekB()`, `Poke()`, and `PokeB()`, which bypass the MMU to access 22-bit physical address space.

C.5.4   PC memory management

```
void *GetMemory(dword size)
```

Allocates a block of PC memory. `size` is the size of the requested region in bytes. If successful, returns a pointer to the block of memory. Returns zero (`NULL`) on failure.

```
void RetMemory(void *addr)
```

Frees a block of PC memory, previously allocated by `GetMemory()`.

C.5.5   Fork queue

There are often cases where an interrupt service routine or signal handler, which is entered asynchronously, needs to switch to main program level. This may be because it needs to call code which is non-reentrant, such as the majority of E11 entry points, or it needs to perform processing which would take too long to be done at interrupt level. E11 does this by enqueueing requests for callbacks, and then processing them at "fork level", which is what the main program switches to whenever E11 is between PDP-11 instructions and there are items in the "fork queue". Callbacks are made serially (one at a time) until the fork queue is empty. Many PDP-11 operating systems have a similar concept, so this may be familiar.

```
E11HANDLE GetForkQel(void)
```

Allocates a queue element which can be used to enqueue requests for callbacks at fork level. One queue element is required for every possible request that can be outstanding at one time. The return value is a non-zero handle to the queue element on success, or zero on failure.

```
void RetForkQel(E11HANDLE handle)
```

Frees a fork queue element that was obtained earlier from `GetForkQel()`. If a request is currently enqueued, it is dequeued before the queue element is flushed. This routine would normally be called during the module's

`DllMain()` exit processing.

void EnqueueFork(E11HANDLE handle,void (*crtn)(dword),dword parm)

Enqueues a request for a callback at fork level. `handle` is the queue element handle returned from `GetForkQel()`, `crtn` is a pointer to the completion routine to call at fork level, and `parm` is an arbitrary 32-bit parameter which will be passed to the completion routine. `EnqueueFork()` is unusual in that it may be safely called from an interrupt service routine or signal handler.

The completion routine is called in the very near future, before the next PDP-11 instruction is fetched, as follows:

```
void __stdcall crtn(dword parm)
```

`parm` is the same value that was passed to the `EnqueueFork()` call, and is for user-defined purposes, for example it may be a pointer to per-instance data so that the same fork completion routine may be used for several different devices. When the completion routine is called, E11 is running at main program level, and all entry points are safe to call.

void DequeueFork(E11HANDLE handle)

Dequeues the previously queued fork request, if any, which was made using the specified `handle`. If no request had been made, this call simply returns without doing anything.

C.5.6 Thread management

E11HANDLE SpawnThread(const char *name)

Creates a child thread. `name` is the address of an .ASCIZ string giving the name to be associated with the thread. The return value is either -1 on error (no child thread has been created), 0 if this is the child thread (in which case all registers have been destroyed except for `EBP`), or the positive thread handle of the child thread if this is the parent.

void KillThread(E11HANDLE thread)

Kills a child thread. `thread` is the thread's process handle as returned by `SpawnThread`. Threads should be killed by the mainline rather than making an OS/2 DosExit() call of their own.

void StopThread(E11HANDLE thread)

Stops a child thread, by sending it a SIGSTOP signal and then waiting until a SIGCHLD signal is received. This is necessary because on multi-CPU systems, SIGSTOP doesn't take effect immediately if the child thread is currently executing on another CPU. This call guarantees that the child thread is frozen before it returns. `thread` is the thread handle of the child thread to stop.

```
void ContThread(E11HANDLE thread)
```

Continues execution of a child thread that was stopped by `StopThread()`. `thread` is the thread handle of the child thread to restart.

Appendix D

PDP-11/74 and PDP-11/45 Multiprocessing (EXPERIMENTAL/UNSUPPORTED)

In the late 1970s, DEC built prototype PDP-11 systems which had several modified PDP-11/70 CPUs ("PDP-11/74s") attached to a shared memory to form one large system, and extended RSX-11M to support these machines (this project led to RSX-11M-PLUS). Also, the earlier PDP-11/45 had a limited ability to share its memory with another processor. Ersatz-11 attempts to emulate these multiprocessor machines, however this feature is completely unsupported. D Bit is happy to accept comments and problem reports about E11's multiprocessing features but makes no commitment to act on them. These features are provided for experimental uses only and are not guaranteed to work reliably, or at all.

The vestigial PDP-11/74 code that's still present in RSX-11M-PLUS was written for DEC's internal testing only, and is absolutely, completely unsupported by Mentec, HP, Compaq, or DEC. Please do not contact them about anything having to do with PDP-11 multiprocessing. The RSX multiprocessing code and documentation are there only because there was never any reason to rip them out, but multiprocessing was never officially supported.

A PDP-11/74 multiprocessor system consists of the following:

- 2–4 PDP-11/74 CPUs

- one or more MKA11 shared memory boxes

- DIP11A IIST (interprocessor interrupt and sanity timer) device

- console terminal for each CPU

- peripherals as usual

Below is a simple multiprocessor `E11.INI` file.

```
;
; Get decent response even w/o enough host CPUs
;
set idle release
```

```
;
; Set PDP-11/74 CPU model
;
set cpu 74
;
; Set number of emulated CPUs (changes prompt to CPA>)
;
set cpus=2
;
; Set up console for CPB
;
set tt1: bus=cpb
assign tt1: con2:
;
; Set up IIST ports (using default BUS settings of II0 = CPA, II1 = CPB, ...)
;
assign ii0:
assign ii1:
;
; Mount disk and tape units
;
mount du0: rsx11mp.dsk
mount mu0: dump.tap
boot du0:
```

D.1   CPUs and console terminals

The SET CPU 74 and SET CPUS=n commands are used to set up the processors themselves. You must use both commands, in either order. A SET CPU 74 command alone will just give you a single PDP-11/74 CPU. The SET CPUS=n command (n is from 1 to 4) changes the number of CPUs in the system. When you add CPUs, the new ones come up halted.

Unlike CPA (the main CPU that E11 starts with), the new CPUs do not get a console terminal set up for them by default, so you must create one for each CPU after CPA. Use a SET TTn: BUS=x command to tell E11 that the DL11 will be on CPx's bus, and then use an ASSIGN TTn: command to attach it to a terminal device. As long as it's the lowest-numbered TT: device on that CPU's bus, it will use the default console terminal addresses of CSR=17777560 and VECTOR=60, so there will be no need to set those.

Whenever E11 is emulating more than one CPU, the "E11>" prompt changes to the name of whichever CPU is operated upon by the execution/debugging commands. Initially this is "CPA>", but you can change it by typing the name of the CPU you want (e.g. CPB etc.). Any command which works on the CPU itself is affected by this. That includes EXAMINE/DEPOSIT/LIST, REGISTER, GO, HALT, INITIALIZE, STEP, etc. ASSIGN/MOUNT/SET commands (which add, remove, or configure peripherals) are not affected by the debugger focus, and may be issued at any time. BOOT commands always apply to the CPU that owns the device being booted (according to its SET ... BUS=x setting).

D.2   Busses

Each PDP-11/74 processor has its own Unibus. E11 needs to know which bus each device is on, so most devices have a SET *ddcu*: BUS=*x* command, which sets that controller to be on CP*x*'s bus. This command must be issued before the command that creates the device. RSX has a corresponding parameter for each device, called the "URM" or Unibus Run Mask. It's expressed variously as an octal number (which represents a binary bitmap, with bit 0 = CPA, bit 1 = CPB, etc.) or as the name of the CPU (e.g. CPA, CPB, etc.). E11 will accept either notation. The fact that RSX always uses "Unibus" in its nomenclature is simply because the PDP-11/74 is a Unibus machine. As far as E11 is concerned, they could just as easily be Q-busses.

Most devices are on bus A by default. The exceptions to this rule are the KW11L line clock and the DIP11A IIST device. There's one KW11L per CPU and it's always on its host CPU's bus, and DIP11A IISTs are useful only in a multiprocessing system so they default to II0: = CPA, II1: = CPB, ... since that's how most users will use them. To avoid causing confusion to users who aren't using the multiprocessor feature, the bus is not shown by SHOW commands unless a SET CPUS=2 (or higher) command is in effect, or the bus is other than A. The default CSR/vector values are assigned on a per-bus basis, so the first device of a given type on a given bus always gets the first default CSR/vector values, even if it's not the first device of that type in the entire system. In particular, the DIP11A IIST devices typically all use the same CSR/vector address, but each one is on a different bus. You can always override a device's CSR/vector values with a SET command.

D.3   II: — DIP11A interprocessor interrupt and sanity timer

Command syntax:

ASSIGN II*n*:

Special switches: none

The DIP11A IIST device is the device used by CPUs in a multiprocessor system to interrupt and/or boot other CPUs, and also to inform the rest of the system when one of the CPUs has crashed. It uses a serial bus with an M8717 interface card in each processor. Since the system uses one common memory for all CPUs, there's no need for the IIST to be very complicated, it just makes other CPUs interrupt and it's up to the operating system to set up its internal tables to show what the interrupts mean. RSX does this by setting bits in a CPU's $MPTAB slot. The bit values are defined in [11,10]MPTBL.MAC.

The RSX $IIWAK routine (in [11,10]IISRV.MAC) uses a software polling loop to wait until the target CPU(s) have accepted their interrupt(s), and returns an error mask if they don't all respond in time. A limit of 64 K loop iterations was generous on real hardware, where all CPUs are running continuously at the same speed (and the IIST is a BR6 device so its interrupts are usually serviced immediately), but it can cause problems running under emulation on a multitasking host operating system where threads can pause for many milliseconds at a time, any time there are at least N+1 active threads on a PC with N CPUs.

E11's workaround is to start counting PDP-11 instructions executed after a CPU sends interrupt(s) to other CPU(s). Once the interrupting CPU has executed N instructions, it pauses its own execution (if necessary) until all the CPUs it's interrupting have also executed N instructions. On a quiet SMP PC with one real 80x86 CPU for each emulated PDP-11 CPU and no active tasks besides E11, this pause will normally be short or nonexistent. But this method means that $IIWAK will always succeed, even on a heavily loaded single-CPU host PC. The default value of *n* has worked in testing, but if necessary it may be changed with the SET DELAY DIP11A 0=*n* command.

SET II: controller parameters:	
`BUS=`x	set home bus segment to CPx
`CSR=`$nnnnnn$	set CSR address to $nnnnnn$ octal
`PRIORITY=`n	set interrupt priority to n (4–7)
`PRIORITY=DEFAULT`	set interrupt priority to default (BR6 for Unibus, BIRQ6 for Q-bus)
`VECTOR=`nnn	set vector address to nnn octal

D.4   Configuring RSX-11M-PLUS for multiprocessing

These instructions are for RSX-11M-PLUS V4.6. Other versions are likely to have differences, which you'll have to figure out. As mentioned at the beginning of this chapter, this feature is unofficial and is not supported in any way by Mentec or any of DEC's incarnations. Not only is it unsupported, but there has never been much hardware to test it on, so the configuration tends to be fragile, and you need to make careful choices and you may need to solve some problems with the automatically-generated build scripts. If at all possible, you should probably try to install completely from scratch (from a Mentec distribution tape), rather than trying to merge multiprocessor support into an existing system. The fewer parameters you change at once, the better.

The first thing you need to do is to edit [200,200]SGNPREFIX.CMD, comment out `.SETF $M$PRO` (by putting ".;" before it) and uncomment `.SETT $M$PRO`, before doing your `@SYSGEN`. When you've done this, suddenly SYSGEN will know all about multiprocessor systems and will ask you many more questions when you run it. The important answers are:

- Number of CPUs (obviously)

- IIST — the default CSR/vector addresses are OK, but see below

- MK11 (MKA11) memory boxes — you must define at least one, even though E11 doesn't specifically emulate them (E11's memory is non-relocatable and always on line)

- Don't specify "XX:" as your crash device, or you'll get an undefined global

- Don't include XDT

You can define other peripherals as usual. It will include a lot of RH70 Massbus adapters without asking you, which is fine. It's up to you whether you put devices on them or use some other kind of disks/tapes. Be careful when including TMSCP tapes ("MU:"), since RSX's default vector for a TMSCP controller is 260, which is the same as the IIST so you should set one of them to something else. MU: is easier to move since its vector is set in software so you won't need to add a `SET MU:` command to your E11.INI file. Similarly, the default CSR and vector for the TA11/TU60 DECcassette ("CT:") are both the same as the IIST, and so is the priority so this is probably not a coincidence (the TU60 wasn't very popular and was mainly used on low-end systems so it may have been a sensible device from which to borrow the CSR/vector). So you'll have to move one or the other if you need CT:.

The rest of the build/save/bootstrap process is the same as always, unless you have errors to work out. Once you have a bootable disk, just boot it the same as on a single-CPU system. Then use the CON ONLINE CPB (etc.) command to boot the other CPU(s) and bring them on line — they aren't included in a CON ONLINE ALL command. It normally takes a little over four seconds to bring a new CPU up, because HRC's boot code includes a 4-second delay between booting the CPU and interrupting it to see if it has come up successfully. The new CPU's console terminal is brought online implicitly when the CPU is started, so there's no need for CON ONLINE YLB (etc.) commands to wake up the consoles.

Ersatz-11's multi-processing support is fragile, and it is easy to crash E11 by doing the wrong things, or by doing the right things in the wrong order. Most emulated devices try to be multiprocessing-compatible, but only a few of

the physical device drivers have multiprocessing support. They are: disk image files, tape image files, RAM disks, RAM tapes, CON: terminal sessions, Linux character devices, and Win32 COM: devices. Anything else is likely to malfunction when attached to a processor other than CPA.

Currently it is important that a processor be created (with `SET CPUS=`*n*) *before* any devices are attached to it with `MOUNT` or `ASSIGN` commands. The `SET ... BUS=`*x* command that sets the bus must also come before the `MOUNT` or `ASSIGN`. This applies even to TU58 controllers, which are not actually attached to a bus but must handle events on the same CPU as the DL11 to which they are connected. If this requirement is not satisfied, data structures may be corrupted.

There are other known bugs: currently it appears to be impossible to bring more than three CPUs online under E11 — CON gives an "offline" error on whichever one you do last, no matter which one it is (not just CPD). This is presumably because of a bug in E11. Also, sometimes a CPU will halt for no apparent reason. D Bit makes no commitment to fix these or any other multiprocessing-specific bugs, but would be very interested in hearing any input on what might be going wrong.

D.5   Multiple PDP-11s with non-shared memory

There are a few scenarios where it may be useful to have multiple PDP-11 CPUs which are completely separate. This is generally no different from running separate instances of E11 under a multi-tasking host OS, except on DOS where there can be only one instance. Creating multiple separate PDP-11s is done the same way as creating a mP PDP-11/74 system, except that the SET CPU NOSHAREDMEM command must be used to give each CPU private memory, and since there are no IISTs each processor must be booted individually.

Here's a multiprocessor `E11.INI` file for separate CPUs:

```
;
; Set PDP-11/45 CPU model (all must be the same) with private memory
;
set cpu 45 nosharedmem
;
; Set number of emulated CPUs (changes prompt to CPA>)
;
set cpus=2
;
; Set up console TTY for CPB
;
set tt1: bus=b
assign tt1: con2:
;
; Mount disks (move DKB: controller to CPB)
;
set dkb: bus=b
mount dka0: rk05a.dsk
mount dkb0: rk05b.dsk
;
; Boot CPUs (do CPA last since it will take the CTY away from E11.INI)
;
boot dkb0:
boot dka0:
```

D.6   FASTBUS: — Fastbus memory on detachable Unibus segment

The PDP-11/45, PDP-11/50, and PDP-11/55 are sometimes described as having dual Unibusses. What this actually means is that any internal MS11-A/B/C semiconductor memory which is connected to the CPU's private "Fastbus" is presented as a Unibus slave for NPR purposes through a separate pair of Unibus in/out slots ("Unibus B"), while core memory (or any kind of standard Unibus memory) appears on the regular I/O bus ("Unibus A") with all of the system's other Unibus peripherals, including the three quad SPC slots in the CPU backplane itself.

Normally this is all connected together as one Unibus using an M9200 jumper in the A/B positions of slots 26 and 27 of the processor backplane, but it need not be. The internal semiconductor memory may be separated, in which case it is not available for NPRs from peripherals on Unibus A.

This separate Unibus B can be connected to a separate DMA-only bus (using an M9300 to grant NPRs in place of a processor), for use e.g. on the "Unibus B" side of an RH11 Massbus adapter. Or, the Unibus B connections can be used to make the internal memory act as a slave device to an entirely separate processor, such as Unibus A of another PDP-11/45. In this way a multi-processor system can be built where each processor makes a portion (the Fastbus part) of its memory available to another processor or processors. The memory controller is dual-ported between its PDP-11/45 host and Unibus B. The catch is that a system with split Unibusses is unable to use DMA to access its own internal memory, so presumably it will also have some Unibus memory for use with its mass storage.

A Fastbus memory is created with the following command:

`ATTACH FASTBUS`p`:` $start$`:`end

This creates a block of memory on processor `CP`p (which is the only processor that can't access the memory by DMA), covering the specified absolute address range. This range must be outside of regular memory, so the I/O page will probably have to be enlarged (using the `/IOPAGE:`$nnnn$ switch on the E11 startup command line so that the desired address range begins at or above the line between plain memory and I/O space.

The memory is attached to other processors by issuing the same `ATTACH` commands (with the identical address range, or with the address range omitted) while the other processor has the debugger focus (because of the `CPA`, `CPB` etc. commands). Unlike real hardware, E11 allows the same memory to be visible to more than two processors. If the very first `ATTACH` command gives one processor in the `FASTBUS`p`:` device name but is issued while another processor has the debugger focus, then the memory is connected to both processors and no further commands are necessary.

D.7   IPL: — Internal inter-processor link

E11 includes a virtual inter-processor link device which can connect two emulated DR11C (DRV11) or DR11W (DRV11WA) ports to each other, for very high-speed communication between processors of a multi-processor system. The typical usage is as follows:

```
assign xa0: ipla0:
set xa1: bus=b
assign xa1: ipla1:
```

In this example, XA0: is on processor CPA by default, and XA1: is explicitly put on CPB. Each `IPL`c`:` controller letter represents a link, and units 0 and 1 are the two endpoints of that link.

Appendix E

Dates and Times

Ersatz-11 has several features that facilitate passing date and time information back and forth between the host system and the PDP-11 operating system. The PC clock can be read from the PDP-11, and E11 can simulate a software bootstrap of RT-11 and RSTS/E so that these systems will pick up the current time as if they were started with a warm boot from a monitor that was already running.

E.1 Booting

RT-11 and RSTS/E both have methods for discovering whether they were software booted from another running RT-11 or RSTS/E monitor, and if so they can "inherit" the time and date from that monitor. E11's `BOOT` command can simulate this situation. If you use it with a `/RT11` or `/RSTS` switch, it obtains the current PC date and time and passes them on to the PDP-11 operating system.

E11's keyscript language has commands which allow composing a time and/or date string in almost any format, so that it may be entered with a single keystroke. The default keyscript for the F12 key sends the current time and date in a format accepted by recent versions of RSX, so it may be pressed in response to the date/time prompt at system startup. See Appendix A for information on keyscript commands, and type `SHOW KEY F12` at the E11 command prompt to see how the existing binding works.

Newer versions of DEC operating systems have support for the KDJ11E TOY clock, which E11 emulates using the host operating system's clock. For example, the RSX "`TIM /SYN`" command will read the RSX time from the PC, and "`TIM /SETTOY`" will set the PC clock from the RSX time. These commands work only if RSX has detected a KDJ11E-based CPU (i.e. PDP-11/93 or PDP-11/94), however E11 can add the TOY clock to any CPU emulation by adding `ASR` to the end of the `SET CPU` command (since the TOY clock is accessed through the Additional Status Register). A `TOY.TSK` program is available from `ftp.dbit.com` which has `/SYN` and `/SETTOY` switches which work just like the TIM command, but with no restriction on the CPU type, and it doesn't have the year 2000 problem that `TIM /SYN` did in earlier versions of RSX.

E.2 PC clock

PCs don't have a 50/60 Hz line frequency clock, so E11 simulates the KW11L style clock by reprogramming the PC's crystal-controlled interval timer for 50 or 60 Hz (the `SET HZ` command selects which frequency, the default is

60 Hz). These speeds can not be produced precisely by the PC's interval timer (it uses rather odd numbers since its master clock is the 14.31818 MHz ISA bus clock fed into a divide-by-12 counter), so E11 programs it as close as possible to the right value, and then uses fractional math to decide when to insert "leap ticks" so that the average rate will be exactly 50 or 60 Hz. In practice, this is not perfectly accurate because many (or most) PCs don't have a very accurate 14.31818 MHz clock to begin with. So their clocks tend to gain or lose time regardless of what applications software or operating system is running.

E11's solution to this is the SET CLOCK command, which allows adjustment of the actual master clock frequency value (which can be specified in 1 Hz increments) which E11 uses to calculate the interval clock divisor and schedule leap ticks. There's no need to actually measure the master clock frequency on the PC's motherboard, simply experimenting with values can greatly improve the accuracy and of course letting the system run for a while and then checking its clock will allow you to figure out the exact drift.

E.3   Year 2000 issues

Ersatz-11 has no year 2000 issues of its own. It does very little manipulation of dates, and when dates are handled internally it uses 16 bits to hold the year, which is enough to last until the year 65,535 AD. However problems do occur when translating dates to and from the formats required by the various PDP-11 operating systems, the PC operating system, and the KDJ11E TOY clock. Also the YEAR2 keyscript command sends only 2-digit years, but it's there for the specific purpose of working with non-Y2K-compliant PDP-11 software. If the PDP-11 software can handle 4-digit dates, use YEAR4 instead, which is what's used in the default keyscript for F12 anyway.

E.3.1   KDJ11E TOY clock

The Dallas Semiconductor DS1215 clock chip used in the KDJ11E battery-backed time-of-year ("TOY") clock contains only two digits for the year. Therefore it inherently suffers from the year 2000 bug, and so must any accurate emulation of it, including the one in E11, otherwise it would be incompatible with PDP-11 software designed for the real thing. However this is not as bad as it might seem. Since the TOY clock is used only to hold the *current* time, the only ambiguity that the PDP-11 software has to resolve is what century it is *now*. This is as opposed to dates held in databases, file system timestamps, etc., which can represent past and future dates and so must not be ambiguous. The latest versions of most PDP-11 operating systems have been updated to use a 100-year window when interpreting the year read from the TOY clock. For example 80–99 can be taken to mean 1980–1999, and 00–79 would mean 2000–2079. Future PDP-11 operating system releases can move this window (at this point 00–99 might as well mean 2000–2099), so this scheme can be extended indefinitely. Recent versions of RSX-11M-PLUS cleverly use the TOY clock's day-of-the-week field to figure out what century this must be, so there's no need for a 100-year window.

E.3.2   Dates in RT-11 and TSX-Plus

Older versions of RT-11 used only 5 bits to store the year, as the number of years since 1972. This format stopped working on 01-Jan-2004, and the older RT-11 versions (V5.6 and earlier) have year 2000 problems too so they had already stopped working correctly anyway. Newer versions of RT-11 adapt two previously unused bits in the date word to extend the year offset to 7 bits, which will last until 31-Dec-2099. RT-11 V5.7 is the first version that fully implements this change in all utilities as well as the monitor. TSX-Plus V6.50 also supports the 7-bit year field, so when installed on top of RT-11 V5.7 it too will last until 31-Dec-2099.

E.3.3   Dates in RSX

RSX's internal date format uses a 16-bit word to hold the number of years since 1900. This format will last for many millennia, and it's unusual in that it allows going back all the way to 1900 so that dates in the low 2000s really are ambiguous when expressed in only two digits. Current versions of RSX accept four-digit years in user commands and process them correctly, but older versions are limited to two-digit years which are all assumed to be in the 1900s. Also, the Files-11 ODS-1 disk structure used by RSX and IAS uses only 2-digit years, stored as ASCII digits rather than binary values. Newer versions of RSX extend this format for several more centuries by allowing the "10s" digit to take on the value of characters which follow "9" in the ASCII code.

E.3.4   Dates in RSTS/E

RSTS/E uses the same date format as the old DOS/BATCH system. This consists of the number of years since 1970, times 1000 (decimal), plus the day within the year (1 = January 1), stored as a 16-bit word. In older versions of RSTS/E this word was signed, so its maximum usable value was 32365., or 31-Dec-2002. Again, year 2000 problems with INIT.SYS's time/date parser prevented getting even that far. Newer versions of RSTS/E have changed to using an unsigned date word (negative values were never allowed anyhow), so now it lasts until 31-Dec-2035. Unfortunately RSTS/E's extension to the date format is not applicable to the few remaining DOS/BATCH systems, since DOS/BATCH uses the sign bit of the date word in directory entries to flag a file as contiguous, so it's not available for expansion of the date word. Multiplying the year offset by 1000 makes for very sparse usage of the available date word values, so there is room for expansion of the date format (using day-within-year values from 367 to 999) if that ever becomes necessary, but updates to PDP-11 software will be required.

E.3.5   Dates in Fuzzball

Although the Fuzzball operating system emulates the RT-11 system calls, it actually uses its own 14-bit date format internally instead of RT-11's format. This is encoded as the number of days since 01-Jan-1972, so it runs out in 2016. Fuzzball has year 2000 problems, but replacement modules are available from `ftp.dbit.com` to make it work until 2016. The date format could be extended to 16 bits, but that would require relocating the flags that currently go in the high 2 bits of the date word, and the `.GDAT` system call would require an incompatible change to support that.

E.3.6   Dates in Unix

Unix (and Unix-compatible) systems normally store dates as a signed number containing the number of seconds since midnight UTC on 01-Jan-1970. The type that this value is stored in ("`time_t`") is normally a 32-bit signed integer (in some cases this is true even on Unix systems that run on 64-bit processors), which means that this format will run out in early 2038. This limit applies both to PDP-11 Unix systems being run under emulation, as well as 80x86 Unix or Linux systems being used as the host environment for Ersatz-11.