Unix PDP-11 Emulator
(As11 & Em11)
User's Guide

Duncan White
Jan-Simon Pendry
Diomidis Spinellis

ABSTRACT

As11 and *em11* form an emulated PDP-11* environment which can be used on UNIX† systems to design and develop simple PDP-11 assembly language programs. The emulated PDP-11 includes 16K bytes of store, a screen, a keyboard, a line printer, and two random access disks.

1. Filename Conventions

Just as Modula-2 uses standard suffixes such as *.def* and *.mod* to identify files as belonging to Modula-2, so the PDP-11 system uses the suffixes *.a11* for an assembly language file, and *.e11* for an emulator input file.

2. The Assembler

As11 is a free-format assembler, accepting all the standard PDP-11 mnemonics and operand types. It is invoked by:

as11 file

The action of the assembler is to translate the single *.a11* file named on the command line [you may omit the .a11 suffix] into the corresponding *.e11* file.

Error messages and *warnings* during assembly are reported on the standard error stream. These are intended to be self-explanatory.

The assembler continues after a warning, but aborts after a fatal error.

* PDP-11 is a registered trademark of DEC
† Unix is a registered trademark of AT+T Bell Labs

2.1. An Example Assembly Language Program

To make the following discussion clearer, here is a simple example of a PDP-11 Assembly Language program.

```
;               An Example PDP-11 Assembly Language Program

; A useful ASCII char, newline
nl              =               12

;               Make space for the stack
                .org            500
stack:
;               then declare the startpoint:
                .org            1000
start:
;               initialise the stack ptr
                mov             #stack,sp

                mov             #greeting, -(sp)
                jsr             pc, scr_mesg
                add             #2, sp
                halt

greeting:
                .byte           nl, nl, "hello there everyone"
                .byte           / isn't it a lovely day ? /, nl, nl
                .byte           0
                .even
```

For the moment, let us not worry about the scr_mesg routine. Accept that it simply displays a null terminated message whose starting address is passed on the stack.

2.2. The Format of Assembly Language Programs

Most of the lines in the above program contain a single PDP-11 instruction. Some lines, however, declare labels, or perform assembler directives [known as *pseudo-ops*].

Any line may be terminated by a comment, introduced by a semi-colon which acts until the end of the current line. A line, if so desired, can contain nothing except a comment.

Between the various constituents of a line, you may place any number of **tabs** and **blanks** which act as separators.

The assembler is not sensitive to upper and lower case.

2.3. Basic Concepts

2.3.1. Symbols

A *symbol* is the assembler equivalent of a Modula-2 *constant.* That is, it is a name which is used to represent a particular numeric value, increasing the readability of a program.

It is an error to *redefine* a symbol.

The assembler accepts indefinite-length symbols, which are sequences of alphanumeric and underscore characters, where the first character is not numeric.

There are two ways of declaring a symbol:

1). A symbol assignment of the form:

```
symbol      =           expr
```

2). A label declaration of the form:

```
symbol :
```

This assigns the current value of the Assembler's *location counter* to the symbol.

In the example program, there are the lines:

```
start:
                mov     #stack,sp
```

The first of these lines declares the label *start* to represent the starting address of the program.

All symbols can be used in the rest of the Assembly Language program, as the destination of branches, and in expressions. All the symbols are also available to you in the Emulator.

2.3.2. A Recommendation

We recommend that labels, if given, stand **alone** on the line : it makes editing simpler.

That is, we suggest that the above two lines are **not** laid out as:

```
start:          mov     #stack,sp
```

2.3.3. Registers

The following symbols:

```
r0 r1 r2 r3 r4 r5 r6 r7
sp pc
```

are predefined to represent the PDP-11 registers.

The user may not *define* symbols to represent registers.

2.3.4. Assembler Location Counter

All assemblers need to maintain the position in memory at which to generate code. This is normally known as the location counter. The PDP-11 assembler uses the special symbol "." [dot] to represent the location counter. You can use "." freely in expressions.

2.3.5. Numeric constants

All numeric constants are interpreted as 16-bit two's complement integers, although byte-sized operands will be silently truncated from 16-bit values to 8-bit ones.

Numeric constants may be written in four bases, or as character constants :

1). By default, a number is octal. If preceded by an "0", it is decimal. If preceded by "0x" it is hexadecimal. If preceded by "0b" it is binary.

2). The value of a single character constant, such as *'A'* , is the ASCII code of the given character.

3). The value of a double character constant, such as *'AB'* , is a 16-bit word, computed by placing the ASCII code of the first character in the high byte, and the ASCII code of the second character in the low byte.

2.3.6. String constants

String constants are allowed in the **.byte** pseudo-op, and consist of a sequence of characters, quoted with single quotes, double quotes or slashes. Such a string represents a sequence of 8-bit integers.

2.3.7. Expressions

An expression is a sequence of **symbols** , **constants** , **operators** and **brackets** representing a value. It is evaluated at *assembly time*, not at *run time.*

All arithmetic is performed using 16-bit two's complement.

There are seven levels of precedence for the operators, listed here from highest precedence to the lowest:

Operators	Meaning
unary - ~	negate [2s complement] 1s complement
* / %	multiplication division modulus
+ binary -	addition subtraction
< >	left shift right shift
^	bitwise xor
&	bitwise and
|	bitwise or

All operators of the same precedence are evaluated left to right.

The default precedences may be overridden by the use of brackets.

Note: **a < b** means *shift a, b places left.* Similarly, **a > b** means *shift a, b places right.*

2.4. Instructions

The assembler can handle two types of instructions: *mnemonics* and *pseudo-ops.*

2.4.1. PDP-11 Mnemonics

The mnemonics recognised by this assembler are listed below.

adc	adcb	add	asl	aslb	asr	asrb
bcc	bcs	beq	bge	bgt	bhi	bic
bicb	bis	bisb	bit	bitb	ble	blos
blt	bmi	bne	bpl	bpt	br	bvc
bvs	ccc	clc	cln	clr	clrb	clv
clz	cmp	cmpb	com	comb	dec	decb
emt	halt	inc	incb	iot	jmp	jsr
mark	mov	movb	neg	negb	nop	reset
rol	rolb	ror	rorb	rti	rts	rtt
sbc	sbcb	scc	sec	sen	sev	sez
sob	sub	swab	sxt	trap	tst	tstb
wait	xor					

It is beyond the scope of this manual to explain the operation of these instructions. Refer to your lecture notes, or to the *DEC PDP-11 Processor Handbook* for further details.

2.4.2. Assembler Directives (Pseudo-Ops)

Instructions starting with a '.' are all commands to the assembler:

.org *expr*

Sets the Assembler's Location Counter to the value of the given expression.

The space between the current location counter and the new value is left undefined.

No check is made by the assembler for overlapping areas (but the emulator makes some checks when loading in the executable image).

symbol **=** *expr*

Sets the given symbol to the given expression.

.even

Causes the location counter to be adjusted to the next even address.

.byte *expr* { , *expr* }

Store the given sequence of 8-bit wide data into successive bytes. Remember that strings are also allowed for this pseudo-op.

.word *expr* { , *expr* }

Store the given sequence of 16-bit wide data into successive words.

Currently, the assembler does not support the usual *.blk* and *.ascii* pseudo-ops.

2.5. Addressing Modes

The assembler accepts all the standard PDP-11 addressing modes. That is:

2.5.1. Basic Addressing Modes [0..7]

Name	Syntax	Effect
Register	Rn	op = Rn
Register Deferred	(Rn)	op = mem[Rn]
Autoincrement	(Rn)+	op = mem[Rn] inc Rn
Autoincrement Deferred	@(Rn)+	addr = mem[Rn] inc Rn op = mem[addr]
Autodecrement	-(Rn)	dec Rn op = mem[Rn]
Autodecrement deferred	@-(Rn)	dec Rn addr = mem[Rn] op = mem[addr]
Index	x(Rn)	x = mem[PC] inc PC op = mem[x + Rn]
Index Deferred	@x(Rn)	x = mem[PC] inc PC addr = mem[x + Rn] op = mem[addr]

2.5.2. PC Addressing Modes

Name	Syntax	Effect
Immediate	#n	op = mem[PC] inc PC
Absolute	@#n	addr = mem[PC] inc PC op = mem[addr]
Relative	n	x = mem[PC] inc PC op = mem[x + PC]
Relative Deferred	@n	x = mem[PC] inc PC addr = mem[x + PC] op = mem[addr]

2.5.3. Note:

Autoincrement and autodecrement modes change the register contents by one for **byte** instructions and two for **word** instructions with the following exceptions :

1). R6 and R7 are always changed by 2.

2). In *deferred* modes, a register is always incremented by 2.

3. The Emulator

Em11 is an interactive source level PDP-11 emulator with built-in debugging facilities. It is invoked by:

em11 file

This loads the *.e11* file you name on the command line into the emulator, provides the prompt **(pdp)** and waits for a command.

The emulated PDP-11 has the following characteristics:

3.1. Memory

16K [40000 octal] bytes of RAM.

3.2. Polled I/O System

Every I/O device [detailed below] has a *status* register and one or more *data buffer* registers associated with it. These device registers are mapped into memory at a high address - beyond the end of the RAM.

All devices use a common format for their status register:

Error	Unused	Busy	Unused	Start
15	14..8	7	6..1	0

The general way of using an IO device is:

1). Set up any relevent data registers.

2). Set the *start* bit [bit 0 in the status register]. This causes the action to start.

3). Poll the *busy* bit [bit 7 in the status register] until it goes low.

4). Check the *error* bit [bit 15 in the status register].

5). Read any results out of data registers.

3.2.1. Delays

All the devices, when active, take a certain amount of time [actually, number of PDP-11 instructions] to finish. Different devices will take different lengths of time to complete an operation. Indeed, the same I/O device may take slightly differing times to complete two identical operations.

In fact, these are set up from a configuration file, read in at start up.

This is the reason why you have to *poll* the *busy* bit repetitively.

3.2.2. Errors

For all the I/O devices, the error bit will be set by the user performing the above steps in an incorrect way - the most typical error is probably to start a new operation while the previous operation is incomplete. If this happens, both the active operation and the erroneous operation are aborted, which will typically cause characters or sectors to be omitted.

The error bit is cleared when the start bit is next set.

For disks [only], there is another error situation, discussed in the section on disks.

The devices are:

3.2.3. Keyboard

The PDP-11 keyboard is, naturally enough, simulated by the UNIX keyboard. However, you can take keyboard input from a file, by using the Emulator's "<" command.

The keyboard has a *status register,* known as **KBD_STAT,** which is at location 77560.

In addition, it has a single byte *data register,* known as **KBD_BUF,** at location 77562.

After a successful keyboard operation has completed, this location contains the next character read from the keyboard.

3.2.4. Screen

The PDP-11 screen is simulated by the UNIX screen. You can capture screen output in a file using the Emulator's ">" command.

The screen has a *status register,* known as **SCR_STAT,** at location 77564.

In addition, it has a single byte *data register,* known as **SCR_BUF,** at location 77566.

You should deposit the character to be displayed in SCR_BUF before starting the I/O operation.

3.2.5. Line printer

The PDP-11 line printer is simulated by the UNIX line printer. Each run of a program which writes to the line printer will generate a single job of line printer output.

The printer has a *status register,* known as **LPR_STAT,** at location 77514.

In addition, it has a single byte *data register,* known as **LPR_BUF,** at location 77516.

You should deposit the character to be printed into LPR_BUF before starting the I/O operation.

3.2.6. 2 disks:

These disks have 256 byte sectors, and have 8 cylinders, 8 surfaces, and 8 sectors per track.

When the PDP-11 emulator starts running, two Unix files: *disk1.dat* and *disk2.dat* are read in by the emulator and used as initial disk images. These files are simply text files, which the user should create with vi.

If the Unix file finishes before the entire PDP-11 disk is full, then the rest of that sector has zero bytes written into it and the rest of the disk is left undefined.

When the emulator terminates, the disk images are written out again onto the Unix files *disk1.dat* and *disk2.dat*

The only I/O operation that the disks are capable of is reading or writing an entire sector, from memory to the disk or vice versa. Both disks may be simultaneously active.

Each disk has its own complete set of registers - we will now describe those for disk 1, but disk 2 has an identical set, which start at location 77530. See the table at the end for a summary.

A *status register,* known as **DSK1_STAT,** located at 77520.

A *sector select register,* known as **DSK1_SEL,** located at 77522. This register is laid out as follows:

R/w	Unused	Cylinder	Surface	Sector
15	14..9	8..6	5..3	2..0

The r/w bit should be set for writing, and reset for reading.

A *buffer address register,* known as **DSK1_BAR,** located at 77524. This register stores the PDP-11 address of the start of the buffer to read/write data from/to.

3.2.6.1. Disk R/W Errors

The configuration file which governs the delays of the I/O operations may also mark specific sectors on one or both disks with a probability of failure. This probability governs whether each individual read/write operation on that sector succeeds or fails.

If a read operation fails, then the buffer will be filled with garbage: currently exclamation marks! If a write operation fails, then the relevent sector will be filled in a similar fashion.

Such an error, of course, sets the error bit in the status register.

3.2.7. I/O Address summary

To summarise, the I/O devices have the following registers:

Name	Located at
KBD_STAT	77560
KBD_BUF	77562
SCR_STAT	77564
SCR_BUF	77566
LPR_STAT	77514
LPR_BUF	77516
DSK1_STAT	77520
DSK1_SEL	77522
DSK1_BAR	77524
DSK2_STAT	77530
DSK2_SEL	77532
DSK2_BAR	77534

3.2.8. An example of polled I/O.

Here is the rest of the example program we presented a while back. It will serve as an illustration of the use of a typical I/O device [the screen].

Note that the following routines take their arguments from the stack, and do not corrupt any registers. This will make them safer and easier to use in different circumstances.

```
; PROCEDURE scr_mesg( mesg : ADDRESS );
;
; Purpose: displays message terminated by a zero byte.

scr_mesg:
                mov     r0,-(sp)            ; Save r0
                mov     4(sp),r0            ; r0 = message address
sm_l1:
                tstb    (r0)                ; while (r0) # 0 do
                beq     sm_rts

                movb    (r0)+,-(sp)
                jsr     pc,scr_char         ;   call scr_char( (r0)+ )
                add     #2,sp
                br      sm_l1               ; end while
sm_rts:
                mov     (sp)+,r0            ; restore original r0
                rts     pc
```

```
; PROCEDURE scr_char( ch : CHAR );
;
; Purpose: Print character on screen

scr_stat        =       77564
scr_buf         =       77566

scr_char:
                movb    2(sp),scr_buf       ; copy char to screen buffer
                movb    #1,scr_stat         ; start i/o operation
sc_lp:                                      ; repeat
                tstb    scr_stat            ; until not busy
                bmi     sc_lp
                rts     pc
```

3.3. Emulator Expressions

Emulator expressions are almost identical to assembler expressions, with only the following differences:

1). Two extra unary operators are available:

   a). ***** [the *byte-dereference* operator] which returns the byte stored at the given PDP-11 memory address.

   b). **!** [the *word-dereference* operator] which returns the word stored at the given PDP-11 memory address - which must be even.

2). The assembler location counter [the symbol "."] has no meaning for the emulator.

3.4. Constants

As in the assembler, the emulator can accept constants in four bases, octal by default, or single and double quoted character-constants.

3.5. The Commands

The emulator provides a wide variety of commands with which to manipulate the PDP-11 environment:

help

Displays a short list of the commands available. In addition, the on-line manual page **em11(L)** describes the commands available.

go [*expr*] or **run** [*expr*]

Runs the program at the given address [or the current value of the PC if the address is not given]

You may interrupt the program [should it get into an infinite loop, for example] by pressing CTRL-C.

quit or **exit**

Quits the emulator.

register

Displays all the registers, and the system flags.

trace

Displays the current status of the trace flag.

trace (**on** | **off**)

Switches the trace flag on or off.

When the trace is on, each instruction is displayed before it is executed and the contents of the destination are printed out in octal after the execution.

list [*expr1*]

A full page of the original PDP-11 source code, starting at address *expr1* is displayed.

Notice that the *original* source is listed - the emulator does not actively disassemble a portion of memory. If your program modifies itself, you will have to rely on byte, word and ASCII dumps.

dump (**byte** | **word** | **ascii**) [*expr1* [, *expr2*]]

The contents of the memory are dumped on the screen as octal bytes, octal words, or as ASCII characters.

Expr1, if given, specifies where the dump is to start from (if not given, from where the previous dump stopped - initially address 0) and *expr2*, if given, specifies how many items the dump should print. (if omitted, 32 items will be printed).

break

Displays all current breakpoints.

break (**on** | **off**) *expr*

Adds or removes a breakpoint at the specified address. When the running program hits a breakpoint, execution terminates with the program counter pointing to the address of the breakpoint. If the first instruction to be executed after a go command is a breakpoint it is ignored. Breakpoints are also ignored during the execution of the step command.

step [*expr*]

Executes the given number of instructions, or a single instruction if no number is given.

print *expr*

Evaluates the given expression and prints the result in octal, hexadecimal, decimal and binary.

lhs **=** *expr1*

Here, the left hand side may either be:

1). A register.

2). A byte-dereferenced expression * *expr2* [ie. the byte-contents of the given memory location]

3). A word-dereferenced expression *! expr2* [ie. the word-contents of the given memory location]

The appropriate value is stored in the lhs. This is like an assignment in Modula-2.

move *expr1* **,** *expr2* **,** *expr3*

Moves the section of memory in the address range *expr1* to *expr2* to address *expr3.*

> *filename*

Redirects PDP-11 screen output to the file specified.

< *filename*

Redirects PDP-11 keyboard input from the file specified.

<> *device*

Redirects PDP-11 keyboard input and output to the device specified. This feature can be used to simplify debugging when using trace by redirecting all the interaction with the running PDP-11 program to a different window.

3.6. Emulator Errors

The emulator will, on occasion, produce an *error message* and stop the program running. Two such errors are:

1). A *bus error* : which is caused by an attempt to access a UNIBUS address that does not exist, and

2). An *odd address error* : which occurs if a register used in a **word** autoincrement or autodecrement instruction becomes odd. [This often occurs because SP or PC are set to an odd value]