PDP-13 Minicomputer (TA3)

PDP-13 is a 16-bit minicomputer inspired by DEC, IBM, and other computers from the 1960s and 1970s. It's called "Mini" because up until then the computer systems had not only filled cupboards, but entire rooms or halls. It was only with the invention of the first integrated circuits that computers could be reduced to the size of a wardrobe. This computer fits perfectly into the Techage oil age. Because this computer can only be programmed in machine code at the beginning (like the originals at the time), this requires a certain amount of computer knowledge, which cannot be conveyed in this manual.

So the requirements are:

- Calculating with HEX numbers (16 bit system)
- Basic knowledge of the structure of a CPU (register, memory addressing) and assembler programming
- Perseverance and willingness to learn, because PDP-13 is different than anything you may already know

The PDP-13 minicomputer is not needed in the game, it's rather ment as teaching material in computer basics and computer history. However, like the other controllers, it can be used to control machines. The `pdp13` mod also has its own output blocks, so that there are many possibilities for use.

Due to the length of this guide, it is not available in-game, but only via GitHub: `github.com/joe7575/pdp13/wiki`.

Manual

- Craft the 3 blocks "PDP-13 Power Module", "PDP-13 CPU" and "PDP-13 I/O Rack".
- The rack is available in 2 versions, but they only differ from the front design
- Put the CPU block on the power block and the I/O rack directly next to the power block
- The maximum distance between an expansion block and the CPU is 3 blocks. This also applies to Telewriter, Terminal and all other blocks.
- The CPU and the other blocks are switched on via the power block. This means that all blocks register with the CPU. In order for a new block to be recognized, the system must be switched on again.
- The I/O rack can only be configured when the power block is switched off
- The CPU can only be programmed when the power block is switched on (logical)

I/O Rack

The I/O rack connects the CPU with the world, i.e. other blocks and machines from the mods `techage` or `tubelib` (TechPack). Several I/O blocks can be used per CPU.

- The first I/O rack occupies I/O addresses #0 to #7. Block numbers can be assigned to these addresses via the menu of the I/O rack
- Values that are output via the `out` command at addresses #0 to #7 are then passed on from the I/O rack to the corresponding `techage` or `tubelib` block

- Commands that are sent to the CPU or to the number of the CPU (e.g. from a switch) can be read in again using an `in` command
- The Description field is optional and does not need to be described. In the non-configured case, it shows the block name to which the port is connected via the block number
- The "OUT" field shows the value / command that was last output
- The "IN" field shows either the received command or the response to a sent command. If 65535 is output, no response was received (many blocks do not send an answer to an "on" / "off" command)
- The "help" register shows a table with information on converting input / output values from the CPU to Techage commands (the CPU can only output numbers, these are then converted into Techage text commands and vice versa)

PDP-13 CPU

The CPU block is the core of the system. The block has a menu that is modeled on real mini computers. In real computers, machine commands had to be entered via the row of switches, and memory contents were output via the rows of lamps.

Here commands are entered via the 6 keys on the left and machine commands via the input field below. The upper area is only for output.

- The CPU is started with the "start" button. It always starts at the current address of the program counter (PC), which is also displayed above the row of lamps, for example.
- A started CPU is stopped again with the "stop" key. You can see whether the CPU has started or stopped, for example, at the top of the "run" lamp.
- The program counter is set to zero using the "reset" button (CPU must be stopped for this)
- The CPU executes exactly one command via the "step" key. In the output field you can see the register values, the executed machine code and the machine code that is executed with the next "step".
- The program counter can be set to a value using the "address" button.
- A memory area is output via the "dump" key. The start address must have been entered beforehand using the "address" key.

All figures are to be entered in hexadecimal form!

The "help" register shows the most important assembler commands and the respective machine code. You can already work with this subset of commands. Further information on the instruction set can be found here and here.

The system commands are listed at the end of the table. These are quasi operating system calls that execute additional commands that would otherwise not be possible or would be difficult to implement, such as outputting a text on the telewriter.

Performance

The CPU is able to execute up to 100,000 commands per second (0.1 MIPS). This applies as long as only internal CPU commands are executed. There are the following exceptions:

- The `sys` and the `in` command "cost" up to 1000 cycles, since external code is executed here.
- The `out` command interrupts the execution for 100 ms.
- The `nop` command, which can be used for pauses, also interrupts execution for 100 ms.

Otherwise the CPU runs "full speed", but only as long as the area of the world is loaded. This makes the CPU almost as fast as its big model, the DEC PDP-11/70 (0.4 MIPS).

Selftest

The CPU contains a self-test routine which is executed when the computer is switched on and the result is output on the CPU (this is used to check whether everything has been connected correctly):

```
RAM=4K    ROM=8K    I/O=8
Telewriter..ok
Programmer..ok
```

When fully expanded, the output looks something like this:

```
RAM=32K    ROM=24K    I/O=32
Telewriter..ok  Terminal..ok
Programmer..ok
Tape drive..ok  Hard didk..ok
```

"Programmer" can either be the "Telewriter Programmer" or - if available - the "Terminal Programmer". However, only one "programmer" can be connected with the PDP-13 system.

PDP-13 Telewriter

The telewriter was the terminal on a minicomputer. Output was only made on paper, input via the keyboard. Entered characters could be sent to the computer or written on a tape. Holes were punched in the tape. These tapes could then be reinserted and played back so that saved programs could be transferred back to the computer. The tape thus fulfilled the task of a hard drive, a USB stick or other storage media.

Here, too, the terminal is used for input / output and for writing and reading tapes, whereby there are two types of telewriter terminals:

- Telewriter Operator for normal inputs / outputs from a running program
- Telewriter Programmer for programming the CPU via assembler (monitor ROM chip is required)

Both types can be "connected" to a CPU, whereby there can be a maximum of one device per type, i.e. a maximum of two in total.

Programs can be copied from Punch Tape to the computer (switch "tape -> PDP13") and from the computer onto the punch tape (switch "PDP13 -> tape") via the "tape" menu of the Telewriter. In both cases, a punch tape must be "inserted". The CPU must be switched on (power) and stopped. Whether the transfer was successful is shown on paper ("main" menu tab).

Using the "tape" menu, demo programs can also be copied onto a punch tape and then loaded into the computer. These programs show how to program elementary functions of the computer.

The Telewriter can be addressed via the following `sys` commands:

```
; Output text
move    A, #100     ; Load A with the address of the text (zero terminated)
sys     #0          ; Output text on the telewriter

; Read text
move    A, #100     ; Load A with the destination address where the text should
go (32 characters max.)
sys     #1          ; Reading in text from Telewriter (the number of characters
is returned in A, or 65535)

; Read number
sys     #2          ; Reading in number from Telewriter, the result is in A
                    ; (65535 = no value read)
```

Demo Punch Tapes

There are several demo punch tapes that can be "punched" with the help of telewrites. These tapes show basic programming examples. If you use a tape like a book, i.e. click in the air, the code of the tape is displayed. Here is the example for the "Demo: 7-segment"

```
0000: 2010, 0080    move A, #$80  ; 'value' command
0002: 2030, 0000    move B, #00   ; value in B

loop:
0004: 3030, 0001    add  B, #01
0006: 4030, 000F    and  B, #$0F  ; values from 0 to 15
0008: 6600, 0000    out #00, A    ; output to 7-segment
000A: 0000          nop           ; 100 ms delay
000B: 0000          nop           ; 100 ms delay
000C: 1200, 0004    jump loop
```

The code is always output according to the following scheme:

```
<addr>: <opcodes>   <asm code>    ; <comment>
```

PDP-13 Punch Tape

In addition to demo tapes with fixed, small programs, there are also writable and editable punch tapes. These can (in contrast to the original) be written / changed several times.

The punch tapes have a menu so that they can also be written on by hand. This is used to:

- to give the tape a unique name
- to describe how the program can be used (Description)
- to copy a `.h16` file directly into the code window, which was e.g. created on your own PC (vm16asm).

PDP-13 7-Segment

A HEX number, i.e. 0-9 and A-F, can be output via this block by sending values from 0 to 15 to the block using the `value` command. The block must be connected to the CPU via an I/O rack. Values greater than 15 delete the output.

This also works with the Techage Lua controller: `$send_cmnd(num, "value", 0..16)`

Asm:

```
move A, #$80    ; 'value' command
move B, #8      ; value 0..16 in B
out #00, A      ; output on port #0
```

PDP-13 14-Segment

Any combination of the 14 segments can be output via this 14-segment display and thus different characters and letters can be displayed. The block must be connected to the CPU via an I/O rack.

This also works with the Techage Lua controller: `$send_cmnd(num, "value", number)`

Order of the segments:

```
       1
   ---------
  |\    |   /|
 6| 7  8  9 |2
  |  \ | /  |
  |   \|/   |
   --E-   --A--
  |   /|\   |
  |  / | \  |
 5| D  C  B |3
  |/   |   \|
   ---------
       4
```

Assignment of the values to the segments:

hex value	Segment
0x0001	1
0x0002	2
0x0004	3
0x0008	4
0x0010	5
0x0020	6
0x0040	7
0x0080	8
0x0100	9
0x0200	A
0x0400	B
0x0800	C
0x1000	D
0x2000	E

PDP-13 Color Lamp

This lamp block can light up in different colors. To do this, values from 1-64 must be sent to the block using the `value` command. The block must be connected to the CPU via an I/O rack. The value 0 switches the lamp off.

This also works with the Techage Lua controller: `$send_cmnd(num, "value", 0..64)`

Asm:

```
move A, #$80    ; 'value' command
move B, #8      ; value 0..64 in B
out #00, A      ; output on port #0
```

PDP-13 Memory Rack

This block completes the computer structure as the fourth block. The block has an inventory for chips for memory expansion. The computer has 4 KWords of memory (4096 words) internally and can be expanded to 8 KWords with a 4 K RAM chip. With an additional 8 K RAM chip, the memory can then be expanded to 16 KWords. Theoretically, up to 64 KWords are possible.

In the lower row the rack can hold up to 4 ROM chips. These ROM chips contain programs and are quasi the BIOS (basic input / output system) of the computer. You have to have the program for the chip on tape, which is then "burned" onto the chip with the help of the electronics factory (techage) or crafted by hand (tubelib). You can only get these programs if you have solved the corresponding programming tasks (more on this later).

The inventory of the memory block can only be filled in the given order from left to right. The computer must be switched off for this.

Very Short Example

Here is a concrete example that shows how to use the mod. The aim is to switch on the TechAge signal lamp or Tubelib Lamp (not the PDP-13 Color Lamp!). To do this, you have to output the value 1 via an `out` command at the port where the lamp is "connected". The assembler program for this looks like this:

```
move A, #1   ; Load the A register with value 1
out  #0, A   ; output the value from A register to I/O address 0
halt         ; Stop the CPU
```

Since the computer does not understand these assembler commands directly, the program must be translated into machine code. The help page in the menu of the CPU block is used for this. The result looks like this (the assembler code is after it as a comment):

```
2010 0001   ; move A, #1
6600 0000   ; out  #0, A
1C00        ; halt
```

`mov A` corresponds to the value `2010`, the parameter `#1` is then in the second word `0001`. Values from 0 to 65535 (0000 - FFFF) can be loaded into register A via the second word. For example, a `mov B` is `2030`. A and B are registers of the CPU, with which the CPU can calculate, but also all `in` and `out` commands go through these registers. The CPU has other registers, but these are not required for simple tasks.

With all commands with 2 operands, the result of the operation is always in the first operand, with `mov A, #1` that is in A. With `out #0, A`, A is output to I/O port #0. The code for this is `6600 0000`. Since a large number of ports are supported, this value #0 is again in the second word. This means that up to 65535 ports can be addressed again. In order to output the value in A e.g. on port #2, the following sequence must be entered: `6600 0002`.

These 5 machine commands must be entered on the CPU, whereby only `0` may be entered for `0000` (leading zeros are not relevant).

The following steps are necessary for this:

- Set up computer with power, CPU and an IO rack as described above
- Put the TechAge signal lamp close by and enter the number of the block in the menu of the I/O rack in the top line at address #0
- Switch on the computer at the power block
- If necessary, stop the CPU and set it to address 0 with "reset"
- Enter the 1st command and confirm with "enter": `2010 1`
- Enter the 2nd command and confirm with "enter": `6600 0`

- Enter the 3rd command and confirm with "enter": `1C00`
- Press the "reset" and "dump" keys and check the entries
- Press the "reset" key again and then the "start" key

If you have done everything correctly, the lamp lights up afterwards. The "OUT" field in the menu of the I/O rack shows the output 1, the "IN" field shows a 65535, since no response was received from the lamp.

Exercise

Change the program so that the PDP-13 Color Lamp is controlled. You have to enter the number of the PDP-13 Color Lamp on the I/O rack at port #1:

```
2010 0080   ; move A, #80  ('value' command)
2030 0005   ; move B, #5   (value 5 in B)
6600 0001   ; out  #1, A   (output A on port #1)
1C00        ; halt
```

If you have entered everything correctly, the lamp lights up orange.

Monitor ROM

If you have expanded the computer with the "Monitor ROM" chip and connected a "Telewriter Programmer" terminal, you can program the computer in assembler. This is much more convenient and less prone to errors.

To get to the "Monitor ROM" chip, you have to solve task 1 (see at the end of these instructions).

The monitor program on the computer is started by entering the "mon" command on the CPU and can also be stopped again using the "stop" key. All other buttons on the CPU are not active in monitor mode. It is only operated using the "Telewriter Programmer".

The monitor program supports the following commands, which are also output by entering `?` On the telewriter:

Command	Meaning
`?`	Output help text
`st [#]`	(start) Start the CPU (corresponds to the "start" button on the CPU). The start address can optionally be entered
`sp`	(stop) Stop the CPU (corresponds to the "stop" button on the CPU)
`rt`	(reset) Resetting the program counter (corresponds to the "reset" button on the CPU)
`n`	(next) Execute the next command (corresponds to the "step" button on the CPU). If you then press "enter", the next command will be executed.
`r`	(register) Output the content of the CPU register
`ad #`	(address) Set the program counter (corresponds to the "address" button on the CPU). `#` is the address
`d #`	(dump) Output memory (corresponds to the "dump" button on the CPU). `#` is the start address. If "enter" is then pressed, the next memory block is output
`en #`	(enter) Enter data. `#` is the address. Then values (numbers) can be entered and accepted with "enter"
`as #`	(assembler) Start the assembler. The start address must be specified for `#`. After that, assembler commands can be entered. Entering `ex` will exit this mode.
`di #`	(disassemble) Output of a memory area of the CPU in assembler notation. 4 commands are always issued. If "Enter" is pressed, the next 4 commands are issued. This mode is ended by entering another command.
`ct # txt`	(copy text) Copy text into memory. With `ct 100 Hello World`, the text "Hello World" is copied to the address $100 and terminated with a zero
`cm # # #`	(copy memory) Copy memory. The three `#` mean: Values for source address, target address, number of words
`ex`	(exit) Exit Monitor Mode from the terminal

All numbers are to be entered in hexadecimal form (the '$' character can be omitted)!

Now more extensive programs can be entered and tested in assembler on the "Telewriter Programmer", almost like [Dennis Ritchie](). In order to test programs, they can be worked through with the `n` command in a single step process. The command `r` shows you the register values if necessary.

You can also include the assembler command `brk #0` in your `.asm` program. The program is then interrupted at this point and the monitor shows you the next assembler line, so that you can then continue testing with `n` in the single-step process.

When your program is finished, you can replace the `brk #0` instruction with a `move A, A`, so that your program does not stop at this point.

You can copy programs you have written yourself onto punch tape in order to save them or pass them on to other players.

BIOS ROM

If the computer has been expanded with the "BIOS ROM" chip, the computer has access to the terminal and to the file system of the tape drive and and later also to the hard disk. The computer can theoretically boot from one of the drives if it had an operating system, but more on that later.

For the "BIOS ROM" chip you have to solve task 2.

Once task 2 has been completed, the "BIOS ROM" chip is produced and placed in the storage rack, the following additional sys commands, for example, are available (the @ sign means "memory address of"):

sys #	Meaning	Parameter in A	Parameter in B	Result in A
$50	file open	@file name	mode `w` / `r`	file reference
$51	file close	file reference	-	1=ok, 0=error
$52	read file (in the pipe)	file reference	-	1=ok, 0=error
$53	read line	file reference	@destination	1=ok, 0=error
$54	write file (from the pipe)	file reference	-	1=ok, 0=error
$55	write line	file reference	@text	1=ok, 0=error
$56	file size	@file name	-	size in bytes
$57	list files (in the pipe)	@file name pattern	-	number of files
$58	remove files	@file name pattern	-	number of files
$59	copy file	@source file name	@dest file name	1=ok, 0=error
$5A	move file	@source file name	@dest file name	1=ok, 0=error
$5B	change drive	drive character `t` or `h`	-	1=ok, 0=error
$5C	read word	file reference	-	word
$5D	change dir	@dir	-	1=ok, 0=error
$5E	current drive	-	-	drive
$5F	current dir	@destination	-	1=ok, 0=error
$60	make dir	@dir	-	1=ok, 0=error
$61	remove dir	@dir	-	1=ok, 0=error
$62	get files (in the pipe)	@file name	-	1=ok, 0=error
$63	disk space	-	@destination	1=ok
$64	format disk	drive character `t` or `h`	-	1=ok, 0=error

PDP-13 Terminal

- Terminal operator for normal inputs / outputs from a running program
- Terminal programmer for programming / troubleshooting via assembler (monitor ROM chip is required)

Both terminals can be connected to one CPU, whereby there can be a maximum of one device per type, i.e. a maximum of two in total. The Terminal Programmer replaces the Telewriter Programmer, so only one Programmer device can be used.

The terminal also has additional keys with the following coding: `RTN` = 26, `ESC` = 27, `F1` = 28, `F2` = 29, `F3` = 30, `F4` = 31. `RTN` or the value 26 is sent if only "enter" has been pressed without first entering characters in the input line.

The following sys commands are available for the terminal:

sys #	Meaning	Parameter in A	Parameter in B	Result in A
$10	clear screen	-	-	1=ok, 0=error
$11	print char	char/word	-	1=ok, 0=error
$12	print number	number	base: 10 / 16	1=ok, 0=error
$13	print string	@text	-	1=ok, 0=error
$14	print string with newline	@text	-	1=ok, 0=error
$15	update screen	@text	-	1=ok, 0=error
$16	start editor (data from the pipe)	@file name	-	1=ok, 0=error
$17	input string	@destination	-	size
$18	print pipe (data from the pipe)	-	-	1=ok, 0=error
$19	flush stdout (force output)	-	-	1=ok, 0=error
$1A	output prompt	-	-	1=ok, 0=error
$1B	output beep	-	-	1=ok, 0=error

Monitor V2

Version 2 of the monitor runs on the "Terminal Programmer". This offers the following additional commands:

Command	Meaning
`ld name`	(load) Load a `.com` or `.h16` file from tape or hard disk into memory
`sy # # #`	(sys) Calling a `sys` command with number (only sys numbers less than $300), as well as the values for Reg A and Reg B (optional). Example: `sy 1B` for beep
`br #`	(breakpoint) Setting a breakpoint at the specified address. Only one breakpoint can be set at a time
`br`	(breakpoint) Delete the breakpoint
`so`	(step over) Jump over to the next `call` instruction. If the debugger is currently at a `call` instruction, one would follow the call with a `n(ext)` and run through the called function in single step. With `so` the function is completely executed and the debugger stops again in the next line. Technically there are two commands: `br PC + 2` and `st`. This means that the previously set breakpoint is deleted (see also breakpoints)
ps	(pipe size) Output the fill level of the pipe in (number of text lines)

All figures are to be entered in hexadecimal form!

Breakpoints

The monitor program V2 (terminal) supports a software breakpoint. This means that if you enter e.g. `br 100` at address $100, the code will be overwritten with the new value $0400 (`brk #0`). With `br` (delete breakpoint) the original value is entered again. If the opcode `$0400` is executed by the CPU, the program is interrupted and the debugger / monitor displays the address (the asterisk indicates that a breakpoint has been set here). However, the disassembler displays the original code and not the `brk`. In the case of a memory dump, however, you will see the value `$0400`. The opcode at the position is set back to `$0400` after this original statement has been executed, so that the breakpoint can also be used multiple times.

However, the breakpoint is gone again when the program has been reloaded into memory.

Pipe

In order to simplify memory management for an application written in ASM with certain terminal inputs / outputs, there is an additional data buffer, referred to here as a "pipe". This buffer is managed as a FIFO. Texts can be written line by line into the pipe or read from there using sys commands.

sys #	Result	Parameter in A	Parameter in B	Result in A
$80	push pipe (copy string into the Pipe)	@text	-	1=ok, 0=error
$81	pop pipe (read string from the Pipe)	@dest	-	1=ok, 0=error
$82	pipe size (request size in entries)	-	-	size
$83	flush pipe (empty the Pipe)	-	-	1=ok, 0=error

PDP-13 tape drive

The tape drive completes the computer structure as a further block. The computer now has a real mass storage device on which data and programs can be repeatedly saved and read. With the help of the BIOS ROM chip, the computer is also able to boot from this storage medium.

So that the tape drive can be used, it must be equipped with a magnetic tape and started via the menu. If the tape drive is stopped again, the tape with the data can also be removed again. Tapes are also used for data backup and transfer.

A maximum of one tape drive can be connected to the computer. When specifying the path, the tape drive must be addressed via `t/`, e.g. `t/myfile.txt`.

J/OS-13 operating system

Like any real computer, the PDP-13 does nothing without programs or an operating system. Therefore, programs must either be loaded into the computer via the punch tapes, or programs must be available on the tape drive or hard disk that can be loaded into the memory and executed. To start the computer from a drive, the command `boot` must be entered on the CPU. The CPU can be stopped again with the "stop" button. All other buttons on the CPU are not active in `boot` mode. It is operated exclusively via the operator terminal.

installation

The OS tapes are required to be able to install the operating system. However, this is only available if task 3 has been solved. In addition:

- the computer must be switched on
- the computer has "monitor" and "BIOS" ROM chips
- a tape drive must be "connected" and started
- be "connected" to a terminal operator

To install the operating system, you have to do the following:

- Stop the computer if it is running
- Insert the tape "J/OS Installation Tape" in the Telewriter and copy it to the computer

- Start the computer on the CPU with Reset / Start
- Follow the instructions on the terminal
- After completing the installation, stop the computer on the CPU and start the operating system with the "boot" command
- Enter "ls" at the terminal to check whether the installed files are available

That's it! You have successfully installed J/OS!

Commands on the console

With J/OS the following commands can be entered and executed via the operator terminal:

- `ed <file>` to start the editor. The specified text or assembler file is loaded into the editor. If the file does not yet exist, it will be created.
- `<name>.h16` or `<name>.com` to run a program from one of the drives. With `.com` programs the `.com` extension can also be omitted.
- `mv <old> <new>` to rename or move a file
- `rm <file>` to delete file (s). You can also use `rm *.lst` or `rm test.*`
- `ls [<wc>]` to output the filenames of the files on a drive as a list. With `ls` all files of the current drive are output. with `ls t/*`, for example, all files on the tape drive are output. With `ls test.*` Only files with the name "test" and with `ls *.com` only `.com` files.
- `cp <from> <to>` To copy a file, e.g. `cp t/test.txt h/test.txt`
- `cpn <from> <to>` To copy multiple files, e.g. `cpn t/*.asm h/asm/`. It is important here that a path with a final `/` character is entered as the 2nd parameter.
- `cd t/h` To change the drive. For example, `ls h` for the hard drive. With hard drives, directories are also supported. This also works, for example, with `cd bin`.
- `asm <file>` to translate a file to h16
- `cat <file>` to output the content of a text file
- `ptrd <file>` to copy an ASCII file from the punch tape to the file system
- `ptwr <file>` to copy an ASCII file from the file system to a punch tape
- `h16com.h16 <name>` to convert a `.h16` file into a `.com` file. The file name `<name>` must be entered without an extension.
- `disk` to display the used space on the drive.
- `format h` or `format t` to delete all files on the disk/tape

More commands will follow ...

Other Programs

- `hellow.asm` "Hello world" test program that shows how the parameter transfer works. The program can be started with several parameters without the `.com` extension. These are then output line by line.
- `time.asm` sample program to show the time of day on four 7-segment displays. The 7-segment displays must be connected to port #0 - #3 for this. #0/#1 for hours, #2/#3 for minutes.

More programs to follow ...

CPU error messages

If an error occurs while booting the computer, an error number is output on the CPU.

Number	Error
2	The file `boot` could not be found (drive not recognized or turned off?)
3	The file `boot` could not be read correctly (file defective)
4	The `boot` file does not contain a reference to a `.h16` file
5	The `.h16` file referenced in the `boot` file does not exist
6	The `.h16` file referenced in the `boot` file is corrupt (no ASCII format)
7	The `.h16` file referenced in the `boot` file is corrupt (no valid `.h16` format)

Debugging

To debug a J/OS application, the computer on the CPU can also be started with the command `mon`. To do this, proceed as follows:

- Extend the program to be debugged with a `btk #0` at the point where the debugger should stop.
- Recompile the program with the assembler.
- Important: The computer must have been booted normally so that the program `shell1.h16` is in the memory. Alternatively, `shell1.h16` must be loaded into memory using the monitor command `ld`.
- Stop the computer at the CPU and restart it with the command `mon`.
- Enter the command `st 0` in the programmer terminal, the computer runs as in normal operation.
- Start the program from the console via the operator terminal as usual under J/OS. The debugger is then on the inserted `brk #0`.
- The Programmer Terminal can now be used to debug the application.

COMM ROM (Communication)

With the help of the COMM ROM chip the PDP-13 can exchange commands and data with other system. The following commands are available for this:

sys #	Meaning	Parameter in A	Parameter in B	Result in A
$40	send datagram	@data to be sent	I/O port	1=ok, 0=error
$41	recv datagram	@dest for the data	I/O port	1=ok, 0=no data

A data block is always 64 words. If texts/strings are transmitted, it makes sense to also transmit the trailing zero. If binary data are transmitted, there must be a common understanding between the sender and receiver about the number of valid data words in a data block.

With "send datagram" a data block is sent to another participant. The block number is used again from the I/O rack.

PDP-13 Hard Disk

The Hard Disk completes the computer structure as a further block. The computer now has a second mass storage device with more capacity. The computer is also able to boot from this storage medium with the help of the BIOS ROM chip.

A maximum of one hard disk can be connected to the computer. The hard disk is accessed via `h/`, e.g. `h/myfile.txt`

If this block is removed, the data is retained for 40 days. The block can therefore be placed in a chest for a short time. After 40 days (real days, no game days) the data on the hard disk will be lost. So the block should better be placed somewhere. If the block is destroyed in any other way, the data is gone too.

The hard disk supports one level of directories. Directory names must be 2 or 3 characters long, including letters and numbers. It is advisable to create the following directories. Only these directories are searched by J/OS for executable files or `.asm` files:

```
h/bin - for all executable programs, including the boot file
h/lib - for all .asm files that are used as library by the assembler
        (strcpy.asm, etc.)
h/ubn - for all own executable programs
h/ulb - for all own .asm files that shall be used as library by the assembler
```

With the `cd` command you can switch between drives and directories:

```
cd t
cd h
cd bin  (if in h/)
cd ..   (if in folder)
```

What doesn't work is, for example, `cd ../asm` or `cd h/bin`, i.e. all combinations.

Moving with J/OS from `t` to `h/bin`

When all executable files from `t` to `h/bin` have been copied and the boot file has been adapted, the computer can also boot from the hard drive. Here are the commands that must be executed in sequence:

```
cd h
mkdir am
cd t
cp boot h/bin/boot
cpn * .h16 h/bin/
cpn * .com h/bin/
cd h
cd am
ed boot -> change from "t/shell1.h16" to "h/bin/shell1.h16"
-> stop tape drive
-> reboot CPU
```

Keep the tape with the boot files. If the computer does not boot from the hard drive ...

Assembler Programming

An introduction to assembler programming is an extensive topic. The page Assembly Programming Tutorial is generally recommended. This deals with a completely different CPU and many things are certainly not transferable. But with the knowledge from the tutorial you may get along better with the short documentation on the VM16 CPU and the assembler.

The assembler is available both in-game, i.e. on the PDP-13 system, and for installation on your own PC. It is advisable to install the assembler on your own computer, provided you have installed Python3. Developed and assemblable files can then be copied / pasted into the editor of the PDP-13 terminal.

J/OS Basics

file `boot`

If there is a text file `boot` on one of the run values, this is read in and interpreted by the BIOS. The file has only one line of text with the file name of the program which is to be executed first. With J/OS-13 this file looks like this on the tape drive:

```
t/shell1.h16
```

When booting, the file `boot` is always searched for first on `t`, then `h` and finally in `h/bin` and then found, loaded and executed.

file `shell1.h16`

The next step is to load the file `shell1.16` from the tape drive and execute it from address $0000. `shell1.h16` is a loader program that nests in the address range $0000 - $00BF and remains there. Every application must return to this loader if it is terminated. This is usually done with the instruction `sys #$71`.

Here is the famous "Hello World" program for J/OS-13 in assembler:

```
; Hello world for the Terminal in COM format

    .org $100          ; start at address $100
    .code

    move  A, B         ; com v1 tag $2001
    move  A, #TEXT
    sys   #$14         ; println
    sys   #$71         ; warm start


    .text
TEXT:
    "Hello "
    "World\0"
```

The line `move A, B` does nothing useful except that the value $2001 is generated in memory. If this value is at the beginning of a `.com` file, this file is accepted, loaded and executed as an executable file (com v1 version tag).

File `shell2.com`

Since the loading program does not have any commands (the address range $0000 - $00BF is much too small for this), a second part is reloaded into the address range from $0100 after a cold start. This program has a command line with commands and can load and execute other programs from a drive.

3 types of executable programs are supported:

- `.h16` files are text files in H16 format. This format allows a program to be loaded to a defined address, as is the case with `shell1.h16`, for example. All punch tape programs are also in H16 format. Programs can only be exchanged using punch tapes in this format.
- `.com` files are files in binary format. The binary format is much more compact (approx. Factor 3) and therefore the better choice for programs. `.com` files are always loaded from address $0100 and must be prepared accordingly (instruction `.org $100`).
- `.bat` files are text files with an executable command in the first line (more is not possible yet). E.g. the file `help.bat` contains the text `cat help.txt`. If `help` is entered, the batch file is opened and the command `cat help.txt` is executed so that the help text is displayed.

The following applies to `.com` and `.h16` files: The application must start at address $0100, must not change the address range below $00C0 and must return to the operating system at the end via `sys #$71`.

List of ASM files

The following `.asm` files are also installed with the operating system:

File	Description
asm.asm	Assembler program
cat.asm	Tool to output text files on the terminal
cmdstr.asm	Library function for parameter handling in `.com` files
cpyfiles.asm	Part of the shell2 to copy files
h16com.asm	Tool to convert `.h16` files to `.com`
hellow.asm	"Hello World" sample program
less.asm	Library function to output text screen by screen
nextstr.asm	Library function for string handling
ptrd.asm	Tool to copy `.txt` / `.h16` files from the punch tape to the file system
ptwr.asm	Tool to write `.txt` / `.h16` files on a punch tape
disk.asm	Tool to output information to tape/disk
format.asm	Tool for delete all files on tape/disk
shell1.asm	Part of J/OS
shell2.asm	Part of J/OS
strcat.asm	Library function for string handling
strcmp.asm	Library function for string handling
strcpy.asm	Library function for string handling
strlen.asm	Library function for string handling
strrstr.asm	Library function for string handling
strsplit.asm	Library function for string handling
strstrip.asm	Library function for string handling
time.asm	Sample program to show the time on 7-segment displays

Programming Tasks

In order to be able to produce a ROM chip, the program for the chip on tape is required. Solving this task in real life would be a challenge, but hardly possible for 99.9% of the players.

Therefore, the programming should be simulated here by solving a (simple) programming task, which is still not that easy. But you get an impression of how time-consuming it was to write a program back then.

Ranges of values and two's complement

But first some theory, which is needed for the following tasks.

PDP-13 is a 16-bit machine and can therefore only store values from 0 to 65535 in CPU registers. All calculations are also limited to this range of values. In the case of a representation with a sign, there are only values from -32768 to +32767. Here is an overview of the value ranges:

16 bit value (hex)	Unsigned integer	Signed integer
$0000	0	0
$0001	1	+1
$7FFF	32767	+32767
$8000	32768	-32768
$FFFF	65535	-1

If the sign is to be changed, e.g. if a negative number is to be converted into a positive, the two's complement procedure is used:

```
pos_num = not(neg_num) + 1
```

Or in Assembler:

```
    bpos  A, next    ; positive number: jump to next
    not   A          ; A = not A
    inc   A          ; A = A + 1
next:
```

When calculating with numbers, you must therefore always keep an eye on the range of values. The addition of two 16-bit numbers can lead to a range overflow.

```
    35567        $8AEF
  + 46123      + $B42B
  =========    =========
    81690        $13F1A
```

This overflow value is stored in `addc A, val` (add with carry / overflow) in the B register and can be taken into account in a second calculation step.

Task 1: PDP-13 Monitor ROM

To get the tape for the PDP-13 Monitor ROM, you have to solve the following task:

Calculate the value in Fahrenheit from a value in degrees Celsius using the formula: F = (C ∗ 9/5) + 32

The range of values must be observed. If it is first multiplied by 9, it must be checked whether the result can overflow. If the Celsius values are close to zero and are first divided by 5, the integer calculation results in a large calculation error. With Celsius values less than 5 and greater than 0, the result is always 0.

Possible values for the degrees Celsius are 0 to 400

The Celsius value must be requested via `sys #$300`. After calculating the result, the Fahrenheit value must be output via `sys #$301`. The necessary machine commands for `mul`, `div` and `add` can be found in the help menu of the PDP-13 CPU or in the manual "VM16 Opcodes".

Here is the framework of the program:

```
0B00        ; sys  #$300  (request the start value, the value is then in A)

....        ; Your code has to go here

0B01        ; sys  #$301  (the result must have been saved in A beforehand)
1C00        ; halt        (stop the program execution)
```

Complete this program and enter it in the menu of the PDP-13 CPU. Then start the CPU from address 0. It is recommended to first develop the program on paper, check it again and only then enter it into the CPU, check it again with `dump` and only then execute it.

If the calculation fits and there is an empty tape in the "Telewriter Operator", the PDP-13 monitor program is written to the tape if the result is suitable. In any case, there is a chat output about the calculated value.

Task 2: PDP-13 BIOS ROM

To get the tape for the PDP-13 BIOS ROM you have to solve the following task:

Calculate the sum of two unsigned 32-bit numbers.

The numbers are chosen so that there is no 32-bit overflow. The two numbers must be requested via `sys #$302`. After calculating the result, this must be output via `sys #$303`.

```
; after sys #$302 the numbers are in the memory as follows
addr+0  number 1 low word (low-value part)
addr+1  number 1 high word (high-value part)
addr+2  number 2 low word
addr+3  number 2 high word

; before sys #$303 the result must be in the memory
addr+0  result low word
addr+1  result high word
```

Here is the framework of the program:

```
move A, #$100  ;Load target address for the numbers
sys #$302      ;request the 4 values, these are then in $100- $103

....           ; Your code has to go here

move A, #$100  ;Load the address with the results
sys #$303      ;the calculation result must have been saved in $100 and $101
               beforehand
halt           ;important, otherwise the program will continue to run
               uncontrolled
```

Complete this assembler program and enter it on the "Telewriter Programmer". It is advisable to first develop the program on paper and only then to enter it into the CPU via the "Telewriter Programmer". To do this, you must first enter the command `mon` on the CPU to start the monitor program. Then you can enter the ASM commands on the "Telewriter Programmer". The values are directly transferred to the CPU line by line. Check the entered program with the help of the disassembler. You can also run the program step-by-step. Make sure, however, that you only have one second to calculate the result.

If the calculation fits and there is an empty tape in the "Telewriter Operator", the PDP-13 BIOS program is written to the tape if the result is suitable. In any case, there is a chat output about the calculated value.

Task 3: PDP-13 OS Install Tapes

To get the OS Install Tapes, you have to solve the following task:

Calculate the distance between two points in space, whereby the distance should be calculated in blocks, so as if a route with blocks from pos1 to pos2 had to be built. The blocks for pos1 and pos2 also count. pos1 and pos2 consist of x, y, z coordinates, with all values in the range from 0 to 1000, if you had to build a line from (0,0,0) to (1000,1000,1000), for example you need 3001 blocks.

The tapes are placed in a tape chest, therefore a tape chest must be placed near the CPU (max. 3 blocks distance from the CPU)!

The program must first request the 6 values (x1, y1, z1, x2, y2, z2) via `sys #$304` and at the end output the result via `sys #$305`.

Here is the framework of the program:

```
move  A, #$100  ; Load target address
sys   #$304     ; request the 6 values, these are then in $100-$105


....            ; Your code has to go here

sys   #$305     ; the calculation result must have been saved in A beforehand
halt            ; important, otherwise the program will continue to run
uncontrolled
```

Complete this assembler program and enter it at the "Terminal Programmer". The "Telewriter Programmer" is no longer necessary and can be dismantled. You have to enter the command `mon` on the CPU again to start the monitor program on the terminal. Then you can enter the ASM commands on the "Terminal Programmer". Check the entered program with the help of the disassembler. You can also run the program step by step and set a breakpoint (see "Monitor V2"). Make sure, however, that you only have one second to calculate the result.

If the conversion fits and there is an empty "Tape Chest", the tapes are placed in the chest if the result is suitable. In any case, there is a chat output about the calculated values.

Task 4: PDP-13 COMM Tape

To get the COMM Tape, you have to solve the following task:

Convert the transferred value (0..65535) into a string with the decimal representation of the number (which, for example, also does the Lua function `tostring()`).

The tape is placed in a tape chest, therefore a tape chest must be placed near the CPU (max. 3 blocks distance from the CPU)!

You have to create the program under J/OS using the `ed` editor, compile it with `asm` and then run it from the console. This looks like this, for example:

```
ed exam4.asm
asm exam4.asm
exam.h16
```

In order to be able to run the program under J/OS, the start address of the program must be $100 (hex). The program must first request the value via `sys #$306` and then return the result via `sys #$307`.

Caution: The result string must be in memory after the program so that the program is not overwritten by the string.

Here is the framework of the program:

```
.org $100        ; start address for a J/OS program
sys  #$306       ; request the value, this is then in A


....             ; Your code has to go here


move  A, #$nnn   ; Load A with the string address
sys   #$307      ; Transfer result
sys   #$71       ; Return to J/OS
```

If the conversion fits and there is an empty "Tape Chest", the tape is placed in the chest if the result is suitable. In any case, there is an output on the PDP-13 Terminal Operator.

Task 5: PDP-13 Hard Disk Tape

To get the hard disk tape, you have to solve the following task:

Write a Collatz function that saves all calculated values including the start value and the end value 1 in an array and returns the start address of the array.

The problem is about sequences of numbers that are constructed according to a simple law of formation:

- Start with the passed number in the range 11 - 520
- If n is even, take n / 2 next
- If n is odd, then take 3 * n + 1 next
- Repeat the procedure with the number you received

For example, for the starting number n = 4, the sequence is: 4, 2, 1

For example, the Collatz function in Lua looks like this:

```lua
function collatz(n)
    local tbl = {}
    while n ~= 1 do
        table.insert(tbl, n)
        if n % 2 == 0 then
            n = n / 2
        else
            n = 3 * n + 1
        end
    end
    table.insert(tbl, n)
    return tbl
end
```

You have to create the program under J/OS using the `ed` editor, compile it with `asm` and then run it from the console. The program must first request the value via `sys #$308` and then return the result via `sys #$309`.

Here is the framework of the program:

```
.org $100        ; start address for a J/OS program
sys   #$308      ; request the value, this is then in A


....             ; Your code has to go here

move  A, #$nnn   ; Load A with the array address
sys   #$309      ; Transfer result
sys   #$71       ; Return to J/OS
```

If the conversion fits and there is an empty "Tape Chest", the tape is placed in the chest if the result is suitable. In any case, there is an output on the PDP-13 Terminal Operator.

Good luck! :)