

PDP-13 Minicomputer (TA3)

PDP-13 ist ein 16-Bit-Minicomputer, inspiriert von DEC, IBM und anderen Computer aus den 60er und 70er Jahren. "Mini" deshalb, weil die Rechenanlagen bis dahin nicht nur Schränke, sondern ganze Räume oder Hallen gefüllt hatten. Erst mit der Erfindung der ersten integrierten Schaltkreisen ließen sich die Rechner auf Kleiderschrankgröße reduzieren. Damit passt dieser Computer ideal in das Techage Ölzeitalter. Dadurch dass dieser Computer zu Beginn nur in Maschinencode programmiert werden kann (wie die Originale damals auch), setzt dies einiges an Computerwissen voraus, was nicht in dieser Anleitung vermittelt werden kann.

Voraussetzungen sind damit:

- Grundkenntnisse in Englisch (weitere Dokumente nur in englisch)
- Rechnen mit HEX-Zahlen (16 Bit System)
- Grundkenntnisse im Aufbau einer CPU (Register, Speicheradressierung) und Assemblerprogrammierung
- Ausdauer und Lernbereitschaft, denn PDP-13 ist anders, als alles, was du evtl. schon kennst

Der PDP-13 Minicomputer wird aber im Spiel auch nicht benötigt, sondern dient eher als Lehrmaterial in Computergrundlagen und Computergeschichte. Er kann aber wie die anderen Controller zur Steuerung von Maschinen eingesetzt werden. Die `pdp13` Mod bringt auch eigene Ausgabeblöcke mit, so dass sich viele Möglichkeiten zur Anwendung bieten.

Aufgrund der Länge dieser Anleitung ist diese nicht in-game, sondern nur über GitHub verfügbar:

github.com/joe7575/pdp13/wiki.

Anleitung

- Crafte die 3 Blöcke "PDP-13 Power Module", "PDP-13 CPU" und "PDP-13 I/O Rack".
- Das Rack gibt es in 2 Varianten, die sich aber nur vom Frontdesign unterscheiden
- Setze den CPU Block auf den Power Block und das I/O-Rack direkt neben den Power Block
- Der maximale Abstand zwischen einem Erweiterungsblock und der CPU beträgt 3 Blöcke. Dies gilt auch für Telewriter, Terminal und alle weiteren Blöcke.
- Über den Power Block wird die CPU und die weiteren Blöcke eingeschaltet. Damit registrieren sich alle Blöcke bei der CPU. Damit ein neuer Block erkannt wird, muss das System neu eingeschaltet werden.
- Das I/O-Rack kann nur konfiguriert werden, wenn der Power Block ausgeschaltet ist
- Die CPU kann nur programmiert werden, wenn der Power Block eingeschaltet ist (logisch)

I/O Rack

Das I/O-Rack verbindet die CPU mit der Welt, also anderen Blöcken und Maschinen. Es können mehrere I/O-Blöcke pro CPU genutzt werden.

- Das erste I/O-Rack belegt die I/O-Adressen #0 bis #7. Diesen Adressen können über das Menü des I/O-Racks Blocknummern zugeordnet werden
- Werte, welche über das `out` Kommando an Adresse #0 bis #7 ausgegeben werden, werden dann vom I/O-Rack an den entsprechenden Block weitergegeben

- Kommandos, die an die CPU bzw. an die Nummer der CPU gesendet werden (bspw. von einem Schalter) können so über ein `in` Kommando wieder eingelesen werden
- Das Description Feld ist optional und muss nicht beschrieben werden. Im nicht-konfigurierten Fall zeigt es den Blocknamen an, mit dem der Port über die Blocknummer verbunden ist
- Das "OUT" Feld zeigt den zuletzt ausgegebenen Wert/das zuletzt ausgegebene Kommando
- Das "IN" Feld zeigt entweder das empfangene Kommando oder die Antwort auf ein gesendetes Kommando. Wird 65535 ausgegeben, wurde kein Antwort empfangen (viele Blöcke senden keine Antwort auf ein "on"/"off" Kommando)
- Das "help" Register zeigt eine Tabelle mit Informationen zur Umsetzung von Ein/Ausgabe Werten der CPU zu Techage Kommandos (Die CPU kann nur Nummern ausgeben, diese werden dann in Techage Text-Kommandos umgesetzt und umgekehrt)

PDP-13 CPU

Der CPU Block ist der Rechenkern der Anlage. Der Block besitzt ein Menü, das echten Minicomputern nachempfunden ist. Über die Schalterreihe mussten bei echten Rechnern die Maschinenbefehle eingegeben werden, über die Lampenreihen wurden Speicherinhalte ausgegeben.

Hier werden Kommandos aber über die 6 Tasten links und Maschinenbefehle über das Eingabefeld unten eingegeben. Der obere Bereich dient nur zur Ausgabe.

- Über die Taste "start" wird die CPU gestartet. Sie startet dabei immer an der aktuellen Adresse des Program Counters (PC), welche bspw. auch oben über die Lampenreihe angezeigt wird.
- Über die Taste "stop" wird eine gestartete CPU wieder gestoppt. Ob die CPU gestartet oder gestoppt ist, sieht man bspw. oben an der "run" Lampe.
- Über die Taste "reset" wird der Program Counter auf Null gesetzt (CPU muss dazu gestoppt sein)
- Über die Taste "step" führt die CPU genau einen Befehl aus. Im Ausgabefeld sieht man dann die Registerwerte, den ausgeführten Maschinencode sowie den Maschinencode, der mit dem nächsten "step" ausgeführt wird.
- Über die Taste "address" kann der Program Counter auf einen Wert gesetzt werden.
- Über die Taste "dump" wird ein Speicherbereich ausgegeben. Die Startadresse muss zuvor über die Taste "address" eingegeben worden sein.

Alle Zahlenangaben sind in hexadezimaler Form einzugeben!

Das "help" Register zeigt die wichtigsten Assemblerbefehle und jeweils den Maschinencode dazu. Mit diesem Subset an Befehlen kann man bereits arbeiten. Weitere Informationen zum Befehlssatz findest du [hier](#) und [hier](#).

Am Ende der Tabelle werden die System-Kommandos aufgeführt. Dies sind quasi Betriebssystemaufrufe, welche zusätzliche Befehle ausführen, die sonst nicht möglich, oder aufwändig zu implementieren wären, wie bspw. einen Text auf dem Telewriter auszugeben.

Performance

Die CPU ist in der Lage, bis zu 100.000 Befehle pro Sekunde (0.1 MIPS) auszuführen. Dies gilt, solange nur interne CPU-Befehle ausgeführt werden. Dabei gibt es folgende Ausnahmen:

- Der `sys` und der `in` Befehl "kosten" bis zu 1000 Zyklen, da hier externer Code ausgeführt wird.

- Der `out` Befehl unterbricht die Ausführung für 100 ms, sofern sich der Wert am Ausgang ändert und eine externe Aktionen in der Spielwelt durchgeführt werden muss. Anderenfalls sind es auch nur die 1000 Zyklen.
- Der `nop` Befehl, der für Pausen genutzt werden kann, unterbricht die Ausführung auch für 100 ms.

Ansonsten läuft die CPU "full speed", aber nur solange der Bereich der Welt geladen ist. Damit ist die CPU fast so schnell wie ihr großes Vorbild, die DEC PDP-11/70 (0.4 MIPS).

Selbsttest

Die CPU beinhaltet eine Selbsttest Routine, die beim Einschalten des Rechners ausgeführt und das Ergebnis an der CPU ausgegeben wird (dies dient zur Überprüfung, ob man alles korrekt angeschlossen hat):

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

PDP-13 Telewriter

Der Telewriter war das Terminal an einem Minicomputer. Ausgaben erfolgten nur auf Papier, Eingaben über die Tastatur. Eingegebene Zeichen konnten an den Rechner gesendet, oder auch auf ein Band (tape) geschrieben werden. Dabei wurden Löcher in das Tape gestanzt. Diese Tapes konnten dann wieder eingelegt und abgespielt werden, so dass gespeicherte Programme wieder an den Computer übertragen werden konnten. Das Tape erfüllte damit die Aufgabe einer Festplatte, eines USB-Sticks oder sonstige Speichermedien.

Auch hier dient das Terminal zur Ein-/Ausgabe und zum Schreiben und Lesen von Tapes, wobei es zwei Typen von Telewriter Terminals gibt:

- Telewriter Operator für normale Ein-/Ausgaben aus einem laufenden Programm
- Telewriter Programmer für die Programmierung der CPU über Assembler (Monitor ROM Chip wird benötigt)

Beide Typen können an einer CPU "angeschlossen" sein, wobei es pro Typ maximal ein Gerät sein darf, also in der Summe maximal zwei.

Über das "tape" Menü des Telewriters können Programme von Punch Tape zum Rechner (Schalter "tape -> PDP13") und vom Rechner auf das Punch Tape (Schalter "PDP13 -> tape") kopiert werden. In beiden Fällen muss dazu ein Punch Tape "eingelegt" sein. Die CPU muss dazu eingeschaltet (power) und gestoppt sein. Ob die Übertragung geklappt hat, wird auf Papier ausgegeben ("main" Menü-Register).

Über das "tape" Menü können auch Demo Programme auf ein Punch Tape kopiert und anschließend in den Rechner geladen werden. Diese Programme zeigen, wie man elementare Funktionen des Rechners programmiert.

Der Telewriter kann über folgende `sys` Befehle angesprochen werden:

```

; Ausgabe Text
move    A, #100      ; Lade A mit der Adresse des Textes (Null terminiert)
sys     #0            ; Ausgabe Text auf dem Telewriter

; Einlesen Text
move    A, #100      ; Lade A mit der Zieladresse, wo der Text hin soll (32
Zeichen max.)
sys     #1            ; Einlesen Text vom Telewriter (In A wird die Anzahl der
Zeichen zurück geliefert, oder 65535)

; Einlesen Zahl
sys     #2            ; Einlesen Zahl vom Telewriter, das Ergebnis steht in A
; (65535 = kein Wert eingelesen)

```

Demo Punch Tapes

Es gibt mehrere Demo Punch Tapes, die mit Hilfe des Telewriters "gestanzt" werden können. Diese Tapes zeigen grundlegende Programmierbeispiele. Wenn man ein Tape wie ein Buch nutzt, also in die Luft klickt, wird der Code des Tapes angezeigt. Hier das Beispiel zum "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

```

Die Ausgabe des Codes erfolgt immer nach folgendem Schema:

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

PDP-13 Punch Tape

Neben den Demo Tapes mit festen, kleinen Programmen gibt es auch die beschreibbaren und editierbaren Punch Tapes. Diese können (im Gegensatz zum Original) mehrfach geschrieben/geändert werden.

Die Punch Tapes besitzen ein Menü, so dass diese auch von Hand beschrieben werden können. Dies dient dazu:

- dem Tape einen eindeutigen Namen zu geben
- zu beschreiben, wie das Programm genutzt werden kann (Description)
- direkt ein `.h16` File in das Code-Fenster zu kopieren, welches bspw. am eigenen PC erstellt wurde (vm16asm).

PDP-13 7-Segment

Über diesen Block kann eine HEX-Ziffer, also 0-9 und A-F ausgegeben werden, indem Werte von 0 bis 15 über das Kommando `value` an den Block gesendet werden. Der Block muss dazu über ein I/O-Rack mit der CPU verbunden sein. Werte größer 15 löschen die Ausgabe.

Dies geht auch mit dem Techage Lua Controller: `$send_cmd(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 Color Lamp

Dieser Lampenblock kann in verschiedenen Farben leuchten. Dazu müssen Werte von 1-64 über das Kommando `value` an den Block gesendet werden. Der Block muss dazu über ein I/O-Rack mit der CPU verbunden sein. Der Werte 0 schaltet die Lampe aus.

Dies geht auch mit dem Techage Lua Controller: `$send_cmd(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

Dieser Block vervollständigt als vierter Block den Rechneraufbau. Der Block hat ein Inventar für Chips zur Speichererweiterung. Der Rechner hat intern 4 KWords an Speicher (4096 Worte) und kann durch einen 4 K RAM Chip auf 8 KWords erweitert werden. Mit einem zusätzlichen 8 K RAM Chip kann der Speicher dann auf 16 KWords erweitert werden. Theoretisch sind bis zu 64 KWords möglich.

In der unteren Reihe kann das Rack bis zu 4 ROM Chips aufnehmen. Diese ROM Chips beinhalten Programme und sind quasi das BIOS (basic input/output system) des Rechners. ROM Chips kann man nur auf der TA3 Elektronikfabrik produzieren. Das Programm für den Chip muss man dazu auf Tape besitzen, welches dann mit Hilfe der Elektronikfabrik auf den Chip "gebrannt" wird. An diese Programme kommt man nur, wenn man entsprechende Programmieraufgaben gelöst hat (dazu später mehr).

Das Inventar des Speicherblocks lässt sich nur in der vorgegebenen Reihenfolge von links nach rechts füllen. Der Rechner muss dazu ausgeschaltet sein.

Minimal Beispiel

Hier ein konkretes Beispiel, das den Umgang mit der Mod zeigt. Ziel ist es, die TechAge Signallampe (nicht die PDP-13 Color Lamp!) einzuschalten. Dazu muss man den Wert 1 über ein `out` Befehl an dem Port ausgeben, wo die Lampe "angeschlossen" ist. Das Assembler-Programm dazu sieht aus wie folgt:

```
move A, #1    ; Lade das A-Register mit den Wert 1
out  #0, A    ; Gebe den Wert aus dem A-Register auf I/O-Adresse 0 aus
halt         ; Stoppe die CPU nach der Ausgabe
```

Da der Rechner diese Assemblerbefehle nicht direkt versteht, muss das Programm in Maschinencode übersetzt werden. Dazu dient die Hilfeseite im Menü des CPU-Blocks. Das Ergebnis sieht dann so aus (der Assemblercode steht als Kommentar dahinter):

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

`mov A` entspricht dem Wert `2010`, der Parameter `#1` steht dann im zweiten Wort `0001`. Über das zweite Wort lassen sich so Werte von 0 bis 65535 (0000 - FFFF) in das Register A laden. Ein `mov B` ist beispielsweise `2030`. A und B sind Register der CPU, mit denen die CPU rechnen kann, aber auch alle `in` und `out` Befehle gehen über diese Register. Die CPU hat noch weitere Register, diese werden für einfache Aufgaben aber nicht benötigt.

Bei allen Befehlen mit 2 Operanden steht das Ergebnis der Operation immer im ersten Operand, bei `mov A, #1` also in A. Beim `out #0, A` wird A auf den I/O-Port #0 ausgegeben. Der Code dazu ist `6600 0000`. Da sehr viele Ports unterstützt werden, steht dieser Wert #0 wieder im zweiten Wort. Damit lassen sich wieder bis zu 65535 Ports adressieren.

Diese 5 Maschinenbefehle müssen bei der CPU eingegeben werden, wobei für `0000` auch nur `0` eingegeben werden darf (führende Nullen sind nicht relevant).

Dazu sind die folgenden Schritte notwendig:

- Rechner mit Power, CPU, und einem IO-Rack aufbauen wie oben beschrieben
- TechAge Signallampe in die Nähe setzen und die Nummer des Blockes im Menü des I/O-Racks in der obersten Zeile bei Adresse #0 eingeben
- Den Rechner am Power Block einschalten
- Die CPU gegebenenfalls stoppen und mit "reset" auf die Adresse 0 setzen
- Den 1. Befehl eingeben und mit "enter" bestätigen: `2010 1`
- Den 2. Befehl eingeben und mit "enter" bestätigen: `6600 0`
- Den 3. Befehl eingeben und mit "enter" bestätigen: `1C00`
- Die Tasten "reset" und "dump" drücken und die Eingaben überprüfen
- Nochmals die Taste "reset" und dann die Taste "start" drücken

Wenn du alles richtig gemacht hast, leuchtet danach die Lampe. Das "OUT" Feld im Menü des I/O-Racks zeigt die ausgegebene 1, das "IN" Feld zeigt eine 65535, da von der Lampe keine Antwort empfangen wurde.

Übungsaufgabe

Ändere das Programm so, dass die PDP-13 Color Lamp angesteuert wird. Die Nummer der PDP-13 Color Lamp musst du am I/O-Rack bei Port #1 eingeben:

```
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
```


Hast du alles richtig eingegeben, leuchtet die Lampe orange.

Monitor ROM

Hat man den Rechner mit dem "Monitor ROM" Chip erweitert und ein "Telewriter Programmer" Terminal angeschlossen, kann man den Rechner in Assembler programmieren. Dies ist deutlich komfortabler und weniger fehleranfällig.

Um an den "Monitor ROM" Chip zu kommen, musst du Aufgabe 1 lösen (siehe am Ende dieser Anleitung).

Das Monitor Programm auf dem Rechner wird durch Eingabe des Kommandos "mon" an der CPU gestartet und kann über die Taste "stop" auch wieder gestoppt werden. Alle anderen Tasten der CPU sind im Monitor-Mode nicht aktiv. Die Bedienung erfolgt nur über den "Telewriter Programmer".

Das Monitor Programm unterstützt folgende Kommandos, die auch mit Eingabe von  am Telewriter ausgegeben werden:

Kommando	Bedeutung
<code>?</code>	Hilfetext ausgeben
<code>st [#]</code>	(start) Starten der CPU (entspricht der "start" Taste an der CPU). Die Startadresse kann optional eingegeben werden
<code>sp</code>	(stop) Stoppen der CPU (entspricht der "stop" Taste an der CPU)
<code>rt</code>	(reset) Rücksetzen des Programm Counters (entspricht der "reset" Taste an der CPU)
<code>n</code>	(next) Nächsten Befehl ausführen (entspricht der "step" Taste an der CPU). Wird danach "enter" gedrückt, wird der nächste Befehl ausgeführt.
<code>r</code>	(register) Inhalt der CPU Register ausgeben
<code>ad #</code>	(address) Setzen des Programm Counters (entspricht der "address" Taste an der CPU). <code>#</code> ist dabei die Adresse
<code>d #</code>	(dump) Speicher ausgeben (entspricht der "dump" Taste an der CPU). <code>#</code> ist dabei die Startadresse. Wird danach "enter" gedrückt, wird der nächste Speicherblock ausgegeben
<code>en #</code>	(enter) Daten eingeben. <code>#</code> ist dabei die Adresse. Danach können Werte (Zahlen) eingegeben und mit "enter" übernommen werden
<code>as #</code>	(assembler) Starten des Assemblers. Für <code>#</code> muss die Startadresse angegeben werden. Danach können Assemblerbefehle eingegeben werden. Durch Eingabe von <code>ex</code> wird dieser Modus beendet.
<code>di #</code>	(disassemble) Ausgabe eines Speicherbereichs der CPU in Assemblerschreibweise. Es werden immer 4 Befehle ausgegeben. Wird danach "Enter" gedrückt, werden die nächsten 4 Befehle ausgegeben. Durch Eingabe eines anderen Kommandos wird dieser Modus beendet.
<code>ct # txt</code>	(copy text) Kopieren von Text in den Speicher. Mit <code>ct 100 Hallo Welt,</code> wird der Text "Hallo Welt" an die Adresse \$100 kopiert und mit einer Null abgeschlossen
<code>cm # # #</code>	(copy memory) Speicher kopieren. Die drei <code>#</code> bedeuten: Quell-Adresse, Ziel-Adresse, Anzahl Worte
<code>ex</code>	(exit) Monitor Mode vom Terminal aus beenden

Alle Zahlenangaben sind in hexadezimaler Form einzugeben (auf das '\$' Zeichen kann dabei verzichtet werden)!

Jetzt können auch umfangreichere Programme in Assembler am "Telewriter Programmer" eingegeben und getestet werden, fast wie [Dennis Ritchie](#). Um Programme zu testen, können diese mit dem `n` Kommando im Einzelschrittverfahren durchgearbeitet werden. Das Kommando `r` zeigt dir bei Bedarf die Registerwerte.

Du kannst aber auch den Assemblerbefehl `brk #0` in dein `.asm` Programm einbauen. Das Programm wird dann an dieser Stelle unterbrochen und der Monitor zeigt dir die nächste Assemblerzeile an, so dass du dann mit `n` weiter im Einzelschrittverfahren testen kannst.

Ist dann dein Programm fertig, kannst du die `brk #0` Anweisung bspw. durch ein `move A, A` ersetzen, so dass dein Programm an dieser Stelle nicht mehr anhält.

Selbst geschriebene Programme kannst du auf Punch Tape kopieren, um diese zu sichern, oder an andere Spieler weiterzugeben.

BIOS ROM

Hat man den Rechner mit dem "BIOS ROM" Chip erweitert, hat der Rechner Zugriff auf das Terminal und auf das Filesystem des Bandlaufwerks und der Festplatte. Der Rechner kann damit theoretisch von einem der Laufwerke booten, wenn er denn eine Betriebssystem hätte, aber dazu später mehr.

Für den "BIOS ROM" Chip musst du Aufgabe 2 lösen.

Zur Verfügung stehen ab sofort bspw. folgende zusätzliche sys-Kommandos (Das Zeichen `@` bedeutet "Speicheradresse von"):

sys #	Bedeutung	Parameter in A	Parameter in B	Ergebnis 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 die pipe)	file reference	-	1=ok, 0=error
\$53	read line	file reference	@destination	1=ok, 0=error
\$54	write file (aus der 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 die 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 die 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

Sofern das BIOS ROM verfügbar ist, kann am Rechner auch ein Terminal angeschlossen und angesteuert werden. Auch hier gibt es zwei Typen von Terminals:

- Terminal Operator für normale Ein-/Ausgaben aus einem laufenden Programm
- Terminal Programmer für die Programmierung/Fehlersuche über Assembler (Monitor ROM Chip wird benötigt)

Beide Terminals können an einer CPU angeschlossen sein, wobei es pro Typ wieder maximal ein Gerät sein darf, also in der Summe maximal zwei. Das Terminal Programmer ersetzt dabei den Telewriter Programmer, es kann also nur ein Programmer Gerät genutzt werden.

Das Terminal besitzt 2 Betriebsarten:

- Editor-Mode
- Terminal-Mode 1 mit zeilenweise Ausgabe von Texten

Das Terminal besitzt auch zusätzliche Tasten mit folgenden Codierung: RTN = 26, ESC = 27, F1 = 28, F2 = 29, F3 = 30, F4 = 31. RTN oder der Wert 26 wird gesendet, wenn nur "enter" gedrückt wurde, ohne dass zuvor Zeichen in die Eingabezeile eingegeben wurden.

Für das Terminal stehen folgende sys-Kommandos zur Verfügung:

sys #	Bedeutung	Parameter in A	Parameter in B	Ergebnis 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 (Daten aus der pipe)	@file name	-	1=ok, 0=error
\$17	input string	@destination	-	size
\$18	print pipe (Daten aus der pipe)	-	-	1=ok, 0=error
\$19	flush stdout (Ausgabe erzwingen)	-	-	1=ok, 0=error
\$1A	prompt ausgeben	-	-	1=ok, 0=error
\$1B	beep ausgeben	-	-	1=ok, 0=error

Monitor ROM V2

Auf dem "Terminal Programmer" läuft die Version 2 des Monitors. Diese bietet folgende zusätzliche Kommandos:

Kommando	Bedeutung
<code>ld name</code>	(load) Laden einer <code>.com</code> oder <code>.h16</code> Datei in den Speicher
<code>sy # # #</code>	(sys) Aufrufen eines <code>sys</code> Kommandos mit Nummer (nur sys-Nummern kleiner \$300), sowie die Werte für Reg A und Reg B (optional). Beispiel: <code>sy 1B</code> für beep
<code>br #</code>	(breakpoint) Setzen eines Breakpoints an der angegebenen Adresse. Es kann immer nur ein Breakpoint gesetzt werden
<code>br</code>	(breakpoint) Löschen des Breakpoints
<code>so</code>	(step over) Springe über die nächste <code>call</code> Anweisung. Steht der Debugger aktuell an einer <code>call</code> Anweisung, würde man mit einem <code>n(ext)</code> dem call folgen und die aufgerufene Funktion im Einzelschritt durchlaufen. Mit <code>so</code> wird die Funktion komplett ausgeführt und der Debugger bleibt in der nächsten Zeile wieder stehen. Technisch sind dies zwei Kommandos: <code>br PC+2</code> und <code>st</code> . Das bedeutet, der zuvor gesetzte Breakpoint ist damit gelöscht (siehe auch Breakpoints)
<code>ps</code>	(pipe size) Den Füllstand der Pipe in (Anzahl von Textzeilen) ausgeben

Alle Zahlenangaben sind in hexadezimaler Form einzugeben!

Breakpoints

Das Monitor Programm V2 (Terminal) unterstützt einen Software Breakpoint. Dies bedeutet, dass bei Eingabe von bspw. `br 100` an der Adresse \$100 der Code mit dem neuen Wert \$0400 (`brk #0`) überschrieben wird. Mit `br` (Breakpoint löschen) wird wieder der Originalwert eingetragen. Wird von der CPU der Opcode `$0400` ausgeführt, wird das Programm unterbrochen und der Debugger/Monitor zeigt die Adresse an (der Stern zeigt an, dass hier ein Breakpoint gesetzt wurde). Allerdings wird vom Disassembler der Original Code angezeigt und nicht der `brk`. Bei einem Speicher-Dump sieht man allerdings den Wert `$0400`. Der Opcode an der Position wird nach Ausführung dieser Originalanweisung wieder auf `$0400` gesetzt, so dass der Breakpoint auch mehrfach genutzt werden kann.

Der Breakpoint ist allerdings wieder weg, wenn das Programm neu in den Speicher geladen wurde.

Pipe

Um die Speicherverwaltung für eine in ASM geschriebene Anwendung bei bestimmten Terminal-Ein-/Ausgaben zu vereinfachen, gibt es einen zusätzlichen Datenpuffer, hier als "pipe" bezeichnet. Dieser Puffer wird als FIFO verwaltet. Über sys-Kommandos kann man Texte zeilenweise in die Pipe schreiben bzw. von dort lesen.

sys #	Bedeutung	Parameter in A	Parameter in B	Ergebnis in A
\$80	push pipe (string in die Pipe kopieren)	@text	-	1=ok, 0=error
\$81	pop pipe (string von der Pipe lesen)	@dest	-	1=ok, 0=error
\$82	pipe size (Größe in Einträgen anfordern)	-	-	size
\$83	flush pipe (Pipe leeren)	-	-	1=ok, 0=error

PDP-13 Tape Drive

Das Tape Drive vervollständigt als weiterer Block den Rechneraufbau. Damit verfügt der Rechner jetzt über einen echten Massenspeicher, auf dem Daten und Programme wiederholt gespeichert und gelesen werden können. Der Rechner ist mit Hilfe des BIOS ROM Chips auch in der Lage, von diesem Speichermedium zu booten.

Damit das Tape Drive genutzt werden kann, muss es mit einem Magnetic Tape bestückt und über das Menü gestartet werden. Wird das Tape Drive wieder gestoppt, kann das Tape mit den Daten auch wieder entnommen werden. Damit dienen Tapes auch der Datensicherung und Weitergabe.

Es kann maximal ein Tape Drive am Rechner angeschlossen werden. Das Tape Drive muss bei der Pfadangabe über `t/`, also bspw. `t/myfile.txt` angesprochen werden.

J/OS-13 Betriebssystem

Wie jeder echte Rechner macht auch PDP-13 ohne Programme oder Betriebssystem gar nichts. Deshalb müssen entweder Programme über die Punch Tapes in den Rechner geladen werden, oder es müssen Programme auf dem Tape Drive oder Hard Disk vorhanden sein, die in den Speicher geladen und ausgeführt werden können. Um den Rechner von einem Laufwerk zu starten, muss an der CPU das Kommando `boot` eingegeben werden. Über die Taste "stop" kann die CPU wieder gestoppt werden. Alle anderen Tasten der CPU sind im `boot`-Mode nicht aktiv. Die Bedienung erfolgt ausschließlich über das Operator Terminal.

Installation

Um das Betriebssystem installieren zu können, werden die OS Tapes benötigt. Diese gibt es aber nur, wenn Aufgabe 3 gelöst wurde. Außerdem muss:

- der Rechner eingeschaltet sein
- der Rechner über "Monitor" und "BIOS" ROM Chips verfügen
- ein Tape Drive "angeschlossen" und gestartet sein
- ein Terminal Operator "angeschlossen" sein

Um das Betriebssystem zu installieren, musst du wie folgt vorgehen:

- Den Rechner stoppen, sofern er läuft
- Das Tape "J/OS Installation Tape" im Telewriter einlegen und in den Rechner kopieren
- Den Rechner an der CPU mit Reset/Start starten
- Den Anweisungen am Terminal folgen
- Nach Abschluss der Installation den Rechner an der CPU stoppen und das Betriebssystem über das Kommando "boot" starten
- Am Terminal "ls" eingeben um zu prüfen, ob die installierten Files vorhanden sind

Das wars! Du hast J/OS erfolgreich installiert!

Kommandos auf der Konsole

Mit J/OS können über das Operator Terminal die folgenden Kommandos eingegeben und ausgeführt werden:

- `ed <file>` um den Editor zu starten. Das angegebene Text- oder Assembler-File wird dabei in den Editor geladen. Existiert das File noch nicht, wird es angelegt.
- `<name>.h16` bzw. `<name>.com` um ein Programm von einem der Laufwerke auszuführen. Bei `.com` Programmen kann die `.com` Endung auch weggelassen werden.
- `mv <old> <new>` um ein File umzubenennen oder zu verschieben
- `rm <file>` um File(s) zu löschen. Hier geht auch `rm *.lst` oder `rm test.*`
- `ls [<wc>]` um die Filenamen der Files eines Laufwerks als Liste auszugeben. Mit `ls` werden alle Files des aktuellen Laufwerks ausgegeben. mit `ls t/*` werden bspw. alle Files des Tape Drives ausgegeben. Mit `ls test.*` nur Files mit dem Namen "test" und mit `ls *.com` nur `.com` Files.
- `cp <from> <to>` Um eine Datei zu kopieren, also bspw. `cp t/test.txt h/test.txt`
- `cpn <from> <to>` Um mehrere Dateien zu kopieren, also bspw. `cpn t/*.asm h/asm/`. Wichtig ist hier, das als 2. Parameter ein Pfad mit einem abschließenden `/`-Zeichen eingegeben wird.
- `cd t/h` Um das Laufwerk zu wechseln. Also bspw. `ls h` für das Hard Drive. Beim Hard Drive werden auch Verzeichnisse unterstützt. Damit geht auch bspw. `cd bin`.
- `asm <file>` um ein File zu h16 zu übersetzen
- `cat <file>` um den Inhalt einer Text-Datei auszugeben
- `ptrd <file>` um ein ASCII File vom Punch Tape in das Filesystem zu kopieren
- `ptwr <file>` um ein ASCII File vom Filesystem auf ein Punch Tape zu kopieren
- `h16com.h16 <name>` um ein `.h16` File in ein `.com` File umzuwandeln. Der Dateiname `<name>` muss ohne Endung eingegeben werden.
- `disk` um den belegten Speicherplatz auf dem Laufwerk auszugeben.
- `format h` oder `format t` um alle Files auf HDD oder Tape zu löschen.

Weitere Kommandos folgen...

Weitere Programme

- `hellow.asm` "Hello world" Testprogramm, das zeigt, wie die Parameterübergabe funktioniert. Das Programm kann ohne `.com` Erweiterung mit mehreren Parametern gestartet werden. Diese werden dann zeilenweise wieder ausgegeben.
- `time.asm` Beispielprogramm, um die Uhrzeit (time of day) an vier 7-Segment-Anzeigen auszugeben. Die 7-Segment-Anzeigen müssen dazu an Port #0 - #3 angeschlossen werden. #0/#1 für Stunden, #2/#3 für Minuten.

Weitere Programme folgen...

CPU Fehlermeldungen

Kommt es beim Booten des Rechners zu einem Fehler, wird eine Fehlernummer an der CPU ausgegeben.

Nummer	Fehler
2	Die Datei <code>boot</code> konnte nicht gefunden werden (Laufwerk nicht erkannt oder ausgeschaltet?)
3	Die Datei <code>boot</code> konnte nicht korrekt gelesen werden (Datei defekt)
4	Die Datei <code>boot</code> enthält keine Referenz auf ein <code>.h16</code> File
5	Das in der Datei <code>boot</code> referenzierte <code>.h16</code> File existiert nicht
6	Das in der Datei <code>boot</code> referenzierte <code>.h16</code> File ist korrupt (kein ASCII Format)
7	Das in der Datei <code>boot</code> referenzierte <code>.h16</code> File ist korrupt (kein gültiges <code>.h16</code> Format)

Debugging

Um eine J/OS Anwendung zu debuggen, kann der Rechner an der CPU auch über das Kommando `mon` gestartet werden. Dazu wie folgt vorgehen:

- das Programm, das debuggt werden soll, mit einem `btck #0` an der Stelle erweitern, an der der Debugger anhalten soll
- das Programm mit dem Assembler neu übersetzen
- wichtig: der Rechner muss einmal normal gebootet worden sein, so dass sich das Programm `shell1.h16` im Speicher befindet. Alternativ muss `shell1.h16` über das Monitor-Kommando `ld` in den Speicher geladen werden.
- den Rechner an der CPU stoppen und mit dem Kommando `mon` neu starten
- Im Programmer Terminal das Kommando `st 0` eingeben, der Rechner läuft wie im Normalbetrieb
- Das Programm über das Operator Terminal starten. Der Debugger steht danach auf dem eingefügten `brk #0`

PDP-13 Hard Disk

Die Hard Disk vervollständigt als weitere Block den Rechneraufbau. Damit verfügt der Rechner jetzt über einen zweiten Massenspeicher mit mehr Kapazität. Der Rechner ist auch hier mit Hilfe des BIOS ROM Chips in der Lage, von diesem Speichermedium zu booten.

Es kann maximal eine Hard Disk am Rechner angeschlossen werden. Der Zugriff auf die Hard Disk erfolgt über `h/`, also bspw. `h/myfile.txt`

Wird dieser Block abgebaut, bleiben die Daten erhalten. Wird der Block zerstört, sind die Daten auch weg.

Die Hard Disk unterstützt eine Ebene von Verzeichnissen. Verzeichnisnamen müssen 2 oder 3 Zeichen lang sein, wobei Buchstaben und Zahlen zulässig sind. Es empfiehlt sich, folgende Verzeichnisse anzulegen. Nur diese werden nach ausführbaren Files bzw. `.asm` Files durchsucht:

```
h/bin - für alle ausführbaren Programme, inklusive dem boot File (Achtung: boot selbst muss auch angepasst werden)
h/lib - für alle .asm Files, die vom Assembler als Bibliothek genutzt werden (strcpy.asm, usw.)
h/ubn - für eigene ausführbaren Programme
h/ulb - für eigene .asm Files, die vom Assembler als Bibliothek genutzt werden sollen
```

Mit dem `cd` Kommando kann zwischen Laufwerken und Verzeichnissen gewechselt werden:

```
cd t
cd h
cd bin (sofern auf h/)
cd .. (sofern in einem Verzeichnis)
```

Was nicht geht, ist bspw. `cd ../asm` oder `cd h/bin`, also alle Kombinationen.

Umziehen mit J/OS von `t` nach `h/bin`

Wenn alle ausführbaren Files von `t` nach `h/bin` kopiert sind und das boot File angepasst ist, kann der Rechner auch vom Hard Drive booten. Hier die Kommandos, die der Reihe nach ausgeführt werden müssen:

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

Das Tape mit den boot-Files gut aufheben. Falls der Rechner mal nicht vom Hard Drive bootet...

Assembler Programmierung

Eine Einführung in die Assembler Programmierung ist ein umfangreiches Thema. Grundsätzlich empfehlenswert ist die Seite [Assembly Programming Tutorial](#). Diese behandelt zwar eine ganz andere CPU und viele Dinge sind sicher nicht übertragbar. Aber mit dem Wissen aus dem Tutorial kommt man evtl. mit der knappen Dokumentation zur VM16 CPU und zum `vm16asm` Assembler besser zurecht.

Der `vm16asm` Assembler ist sowohl in-game, also auf dem PDP-13 System, als auch zur Installation auf dem eigenen Rechner verfügbar. Es empfiehlt sich, den Assembler auf dem eigenen Rechner zu installieren, sofern man Python3 installiert hat. Entwickelte und assemblierbare Files können dann per copy/paste in den Editor des PDP-13 Terminals kopiert werden.

Grundlagen zu J/OS

Datei `boot`

Befindet sich auf einem der Laufwerke eine Textdatei `boot`, so wird diese vom BIOS eingelesen und interpretiert. Die Datei hat nur eine Textzeile mit dem Dateinamen des Programms, welches als erstes ausgeführt werden soll. Bei J/OS-13 sieht diese Datei auf dem Tape Drive so aus:

```
t/shell11.h16
```

Die Datei `boot` wird bei Booten immer zuerst auf `t`, dann `h` und zuletzt in `h/bin` gesucht und so fern gefunden, geladen und ausgeführt.

Datei `shell11.h16`

Damit wird als nächstes die Datei `shell11.16` vom Tape Drive geladen und ab Adresse \$0000 ausgeführt. `shell11.h16` ist ein Ladeprogramm, das sich im Adressbereich \$0000 - \$00BF einnistet und dort auch verbleibt. Jede Anwendung muss, sofern sie beendet wird, wieder zu diesem Ladeprogramm zurückkehren. Dies erfolgt normalerweise über die Anweisung `sys #$71`.

Hier das berühmte "Hello World" Programm für 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"
```

Die Zeile `move A, B` macht nichts sinnvolles, außer dass der Wert \$2001 im Speicher generiert wird. Steht dieser Wert am Anfang eines `.com` Files, wird dieses File als ausführbare Datei akzeptiert, geladen und ausgeführt (com v1 version tag).

Datei `shell2.com`

Da das Ladeprogramm über keine Kommandos verfügt (der Adressbereich \$0000 - \$00BF ist dafür viel zu klein), wird nach einem Kaltstart ein zweiter Teil in den Adressbereich ab \$0100 nachgeladen. Dieses Programm besitzt eine Kommandozeile mit Kommandos und kann andere Programme von einem Laufwerk laden und ausführen.

Es werden 3 Typen von ausführbaren Programmen unterstützt:

- `.h16` Files sind Textfiles im H16 Format. Dieses Format erlaubt ein Programm an eine definierte Adresse zu laden, wie dies bspw. bei `shell1.h16` der Fall ist. Auch alle Punch Tape Programme sind im H16 Format. Nur in diesem Format lassen sich auch Programme über Punch Tapes austauschen.
- `.com` Files sind Files im Binärformat. Das Binärformat ist deutlich kompakter (ca. Faktor 3) und deshalb für Programme die bessere Wahl. `.com` Files werden immer ab Adresse \$0100 geladen und müssen dafür entsprechend vorbereitet sein (Anweisung `.org $100`).
- `.bat` Files sind Textfiles mit einem ausführbaren Kommando in der ersten Zeile (mehr geht bis jetzt noch nicht). Bspw. die Datei `help.bat` beinhaltet den Text `cat help.txt`. Wird `help` eingegeben, wird das Batchfile geöffnet und das Kommando `cat help.txt` ausgeführt, so dass der Hilfetext ausgegeben wird.

Für `.com` und `.h16` Files gilt: Die Anwendung muss bei Adresse \$0100 starten, darf den Adressbereich unterhalb von \$00C0 nicht verändern und muss am Ende über `sys #$71` wieder zum Betriebssystem zurückkehren.

Liste der ASM-Files

Mit dem Betriebssystem werden auch die folgenden `.asm`-Files installiert:

File	Beschreibung
asm.asm	Assemblerprogramm
cat.asm	Tool um Textfiles am Terminal auszugeben
cmdstr.asm	Library Funktion für Parameter Handling in <code>.com</code> Files
cpyfiles.asm	Teil der shell2 um Files zu kopieren
h16com.asm	Tool um <code>.h16</code> Files nach <code>.com</code> zu konvertieren
hellow.asm	"Hello World" Beispielprogramm
less.asm	Library Funktion um Text Bildschirmweise auszugeben
nextstr.asm	Library Funktion zum String Handling
ptrd.asm	Tool, um <code>.txt</code> / <code>.h16</code> Files vom Punch Tape ins Filesystem zu kopieren
ptwr.asm	Tool, um <code>.txt</code> / <code>.h16</code> Files auf ein Punch Tape zu schreiben
disk.asm	Tool, um Informationen zum Tape/zur Platte auszugeben
format.asm	Tool, um Files auf Tape/Platte zu löschen
shell1.asm	Teil von J/OS
shell2.asm	Teil von J/OS
strcat.asm	Library Funktion zum String Handling
strcmp.asm	Library Funktion zum String Handling
strcpy.asm	Library Funktion zum String Handling
strlen.asm	Library Funktion zum String Handling
strrstr.asm	Library Funktion zum String Handling
strsplit.asm	Library Funktion zum String Handling
rstrip.asm	Library Funktion zum String Handling
time.asm	Beispielprogramm, um die Uhrzeit auf 7-Segment Anzeigen auszugeben

Programmieraufgaben

Um einen ROM Chip herstellen zu können, wird das Programm für den Chip auf Tape benötigt. Diese Aufgabe in echt zu lösen wäre zwar eine Herausforderung, aber für 99,9 % der Spieler kaum zu lösen.

Deshalb soll die Programmierung hier simuliert werden, in dem man eine (einfache) Programmieraufgabe löst, was immer noch nicht ganz einfach ist. Aber man bekommt einen Eindruck, wie aufwändig es damals war, ein Programm zu schreiben.

Wertebereiche und Zweierkomplement

Zuvor aber etwas Theorie, was für die folgenden Aufgaben benötigt wird.

PDP-13 ist eine 16-Bit Maschine und kann daher nur Werte von 0 bis 65535 in CPU Registern speichern. Alle Berechnungen sind auch auf diesen Wertebereich beschränkt. Bei einer Darstellung mit Vorzeichen sind es nur Werte von -32768 bis +32767. Hier eine Übersicht der Wertebereiche:

16-Bit wert (hex)	Ganzzahl ohne Vorzeichen	Ganzzahl mit Vorzeichen
\$0000	0	0
\$0001	1	+1
\$7FFF	32767	+32767
\$8000	32768	-32768
\$FFFF	65535	-1

Soll das Vorzeichen geändert werden, also bspw. eine negative Zahl in eine Positive gewandelt werden, so wendet man das [Zweierkomplement](#) Verfahren an:

```
pos_zahl = not(neg_zahl) + 1
```

Oder in Assembler:

```
bpos  A, weiter ; positive Zahl: springe zu weiter
not   A         ; A = not A
inc   A         ; A = A + 1
weiter:
```

Beim Rechnen mit Zahlen muss daher immer der Wertebereich im Auge behalten werden. Bei der Addition von zwei 16-Bit Zahlen kann es zu einem Bereichsüberlauf kommen.

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

Dieser Überlaufswert wird bei `addc A, val` (addiere mit carry/Überlauf) im B Register gespeichert und kann in einem zweiten Rechenschritt berücksichtigt werden.

Aufgabe 1: PDP-13 Monitor ROM

Um das Tape für das PDP-13 Monitor ROM zu erhalten, musst du folgende Aufgabe lösen:

Berechne die Summe zweier 32-Bit Zahlen ohne Vorzeichen.

Die Zahlen sind so gewählt, dass es zu keinem 32-Bit Überlauf kommt. Die zwei Zahlen müssen über `sys #300` anfordert werden. Nach Berechnung des Ergebnisses muss dieses über `sys #301` ausgegeben werden.

```

; nach sys #$300 stehen die Zahlen wie folgt im Speicher
addr+0  number 1 low word (niederwertige Anteil)
addr+1  number 1 high word (höherwertige Anteil)
addr+2  number 2 low word
addr+3  number 2 high word

; vor sys #$301 muss das Ergebnis so im Speicher stehen
addr+0  result low word
addr+1  result high word

```

Wenn die Berechnung passt und im "Telewriter Operator" befindet sich ein leeres Tape, dann wird bei passendem Ergebnis das Tape geschrieben. In jedem Falle erfolgt eine Chat-Ausgabe über die berechneten Werte. Hier der Rahmen des Programms:

```

2010 0100 ; move A, #$100 (Zieladresse für die Zahlen laden)
0B00      ; sys #$300      (die 4 Werte anfordern, diese stehen dann in
$100-$103)
....
2010 0100 ; move A, #$100 (Adresse mit den Ergebnissen laden)
0B01      ; sys #$301      (das Rechenergebnis muss zuvor in $100 und $101
gespeichert sein)
1C00      ; halt           (wichtig, sonst läuft das Programm unkontrolliert
weiter)

```

Aufgabe 2: PDP-13 BIOS ROM

Um das Tape für das PDP-13 BIOS ROM zu erhalten, musst du folgende Aufgabe lösen:

Berechne den Abstand zwischen zwei Punkten im Raum, wobei der Abstand in Blöcken berechnet werden soll, also wie wenn eine Hyperloop-Strecke von pos1 zu pos2 gebaut werden müsste. Die Blöcke für pos1 und pos2 zählen mit. pos1 und pos2 bestehen aus x, y, z Koordinaten, wobei sich alle Werte im Bereich von 0 bis 1000 bewegen, Wenn man bspw. von (0,0,0) nach (1000,1000,1000) eine Strecke bauen müsste, würde man 3001 Blöcke benötigen.

Das Programm muss zuerst die 6 Werte (x1, y1, z1, x2, y2, z2) über `sys #$302` anfordern und am Ende das Ergebnis wieder über `sys #$303` ausgeben. Wenn die Berechnung passt und im "Telewriter Operator" befindet sich ein leeres Tape, dann wird bei passendem Ergebnis das Tape geschrieben. In jedem Falle erfolgt eine Chat-Ausgabe über die berechneten Werte. Hier der Rahmen des Programms:

```

move A, #$100 ; Zieladresse laden
sys  #$302    ; die 6 Werte anfordern, diese stehen dann in $100-$105
....
sys  #$303    ; das Rechenergebnis muss zuvor in A gespeichert sein
halt          ; wichtig, sonst läuft das Programm unkontrolliert weiter

```

Aufgabe 3: PDP-13 OS Install Tapes

Um die OS Install Tapes zu erhalten, musst du folgende Aufgabe lösen:

Wandle die übergebenen Wert (0..65535) um in einen String mit der dezimalen Darstellung der Zahl (das was bspw. auch die Lua-Funktion `tostring()` macht).

Die Tapes werden in eine Tape Chest gelegt, daher muss eine Tape Chest bei der CPU platziert sein (max. 3 Blöcke Abstand von der CPU)!

Das Programm muss zuerst den Wert über `sys #304` anfordern und am Ende das Ergebnis wieder über `sys #305` zurück liefern. Wenn die Umwandlung passt und eine leere "Tape Chest" vorhanden ist, dann werden bei passendem Ergebnis die Tapes in die Kiste gelegt. In jedem Falle erfolgt eine Chat-Ausgabe mit dem String. Hier der Rahmen des Programms:

```
sys    #304      ; den Werte anfordern, dieser steht dann in A
....
move   A, #nnn   ; A mit der String-Adresse laden
sys    #305      ; Ergebnis übergeben
halt   ; wichtig, sonst läuft das Programm unkontrolliert weiter
```

Aufgabe 4: PDP-13 Hard Disk Tape

Um das Hard Disk Tape zu erhalten, musst du folgende Aufgabe lösen:

Schreibe eine Collatz-Funktion, die alle berechneten Werte inklusive dem Startwert und dem Endwert 1 in ein Array speichert und die Startadresse des Array zurückliefert.

Bei dem Problem geht es um Zahlenfolgen, die nach einem einfachen Bildungsgesetz konstruiert werden:

- Beginne mit übergebenen Zahl im Bereich 11 - 520
- Ist n gerade, so nimm als nächstes $n/2$
- Ist n ungerade, so nimm als nächstes $3*n + 1$
- Wiederhole die Vorgehensweise mit der erhaltenen Zahl

So erhält man zum Beispiel für die Startzahl $n = 4$ die Folge: 4, 2, 1

Die Collatz-Funktion in Lua sieht bspw. aus wie folgt:

```
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
```

Du musst das Programm unter J/OS über den `ed` Editor erstellen, mit `asm` übersetzen und dann von der Konsole ausführen. Das Programm muss zuerst den Wert über `sys #306` anfordern und am Ende das Ergebnis wieder über `sys #307` zurück liefern. Außerdem müssen die Anforderungen an eine ausführbare Datei und J/OS erfüllt sein. Wenn die Umwandlung passt und eine leere "Tape Chest" vorhanden ist, dann wird bei passendem Ergebnis das Tape in die Kiste gelegt. In jedem Falle erfolgt eine Ausgabe auf dem PDP-13 Terminal Operator. Hier der Rahmen des Programms:

```
.org $100      ; Startadresse für ein J/OS Programm
move A, B     ; Ergennungsmuster für ein .com File

sys #306      ; den Werte anfordern, dieser steht dann in A

.....      ; Hier muss dein Code hin

move A, #nnn  ; A mit der Array-Adresse laden
sys #307      ; Ergebnis übergeben
sys #71       ; Ruecksprung nach J/OS
```

Viel Glück :)