Dr. Bernd Kokavecz
An der Schneise 55A
030 436 718 55
0176 70 94 94 79
www.kokavecz.de

Berlin, im September 2019

Ein im Jahr 1976 selbstgebauter Computer

„Nachbau“ einer PDP-8 von DEC

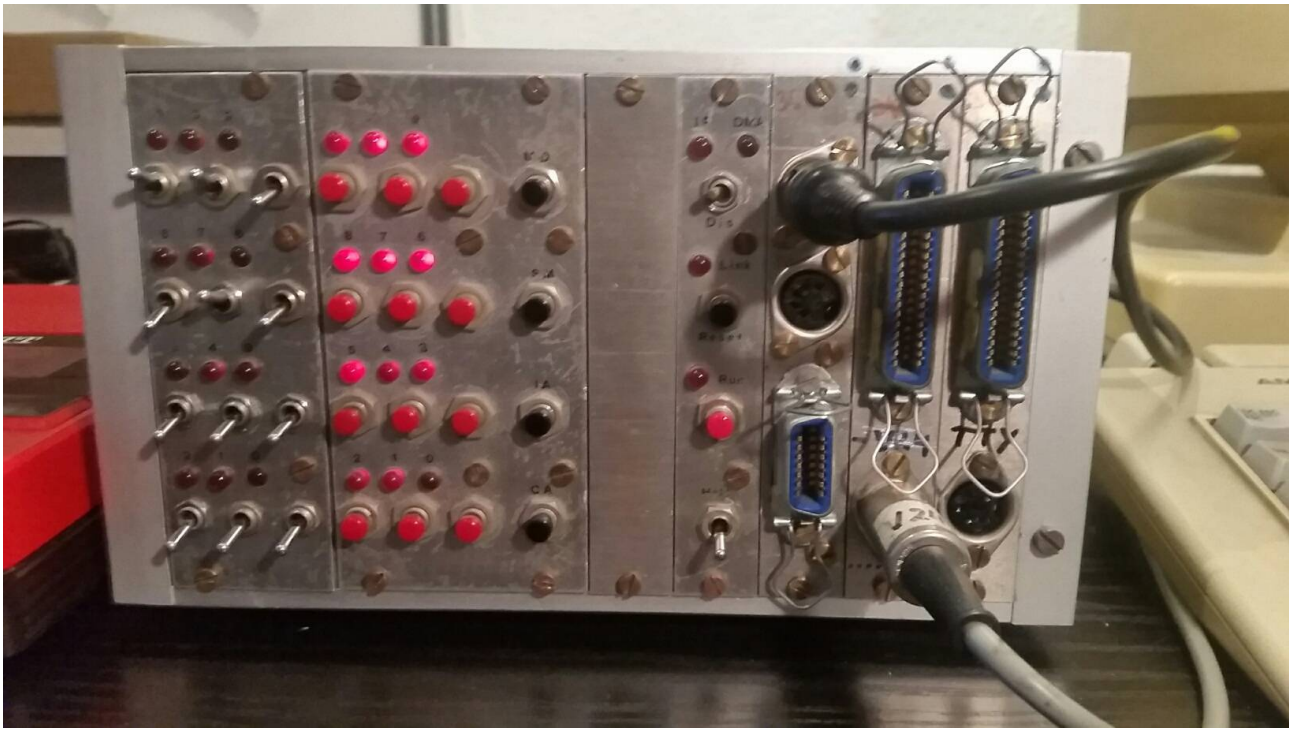
In Jahr 1976 haben mein Freund und Studienkamerad Rainer Joachim Brandenburg und ich zusammen mit Hilfe eines ca. 1973 auf den Markt gekommenen Chipsatzes von Intersil (IM6100-Familie) zwei identische Computer als PDP-8 Clones selber gebaut. Konstruktionsgrundlage waren allein die Datenblätter der Chips und weitere Intersil-Unterlagen. Mit Hilfe von Lochstreifen, die wir an unserem Physik-Institut an der Freien Universität Berlin kopiert hatten, konnten wir eine Reihe von DEC-Programmen nutzen, auch wenn die Ein-Ausgabe- Operationen auf Binärcode-Ebene ein wenig angepasst werden mussten, weil die integrierten Input/Output-Bausteine der Serie nicht ganz kompatibel zur PDP-8 Hardware waren. Die Hardware ist unter Verwendung der ICs eine Eigenentwicklung und wurde mit manuell gezeichneten „gedruckten Schaltungen“ von uns realisiert. Entflechtung, das Zeichnen, die Übertragung auf photographischen Wege auf kupferkaschiertes Basismaterial, das Ätzen, Bohren, Bestücken und Löten haben wir erfolgreich durchgeführt (10 sogenannte Euro-Platinen im Format 10cm x 16cm und ein Netzteil).

Die PDP-8 von DEC war ein Rechner mit 4096 Speicherplätzen à 12 Bit (Grundausrüstung) und ist Ende der 1960er Jahre in Forschungseinrichtungen weit verbreitet gewesen. Ein solcher Rechner kostete damals deutlich mehr als 100 000 DM (50000 €).

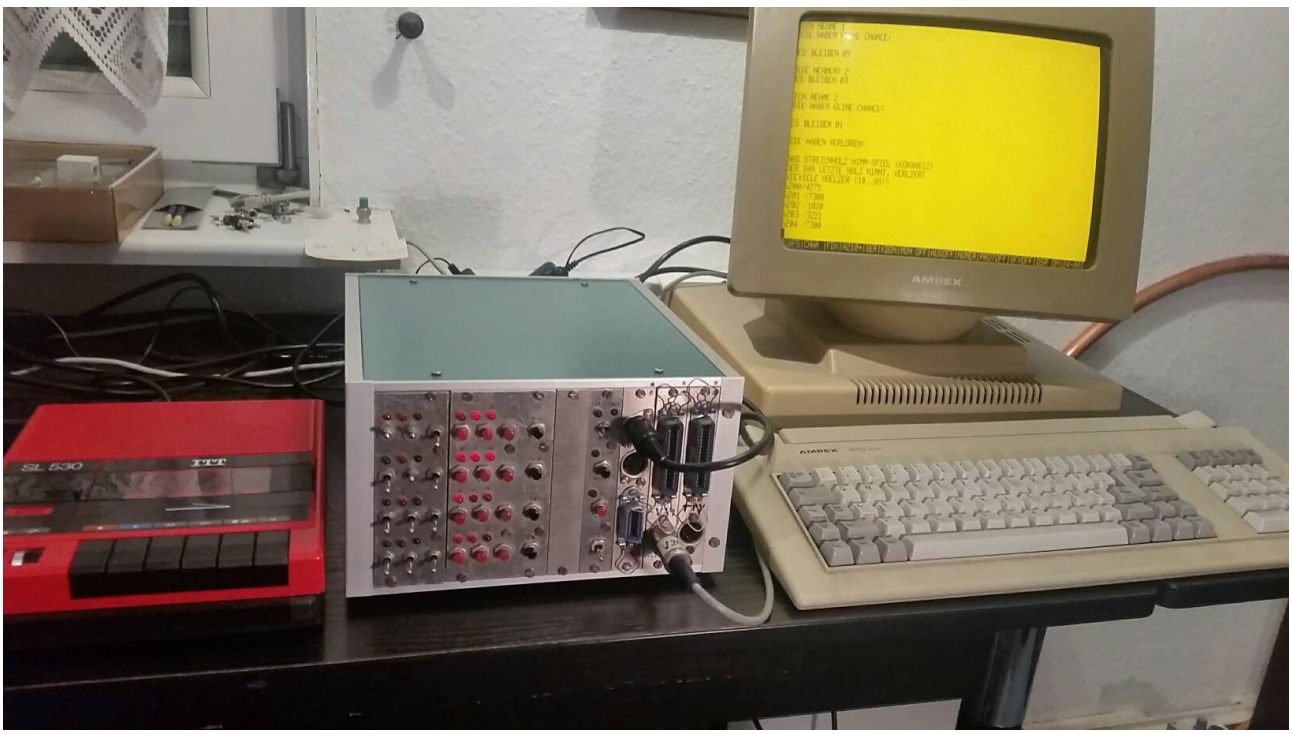
Als Studenten und später als Assistenten waren wir enttäuscht von der umständlichen Arbeit des Programmierens im Rechenzentrum mit Lochkartenstanzer, Abgabe der Karten, Warten, Warten, Warten, dann Rückgabe der Karten und der Ausdrucke - zumeist Fehlerlistings. So entstand die Idee: „Wir brauchen einen eigenen Rechner.“

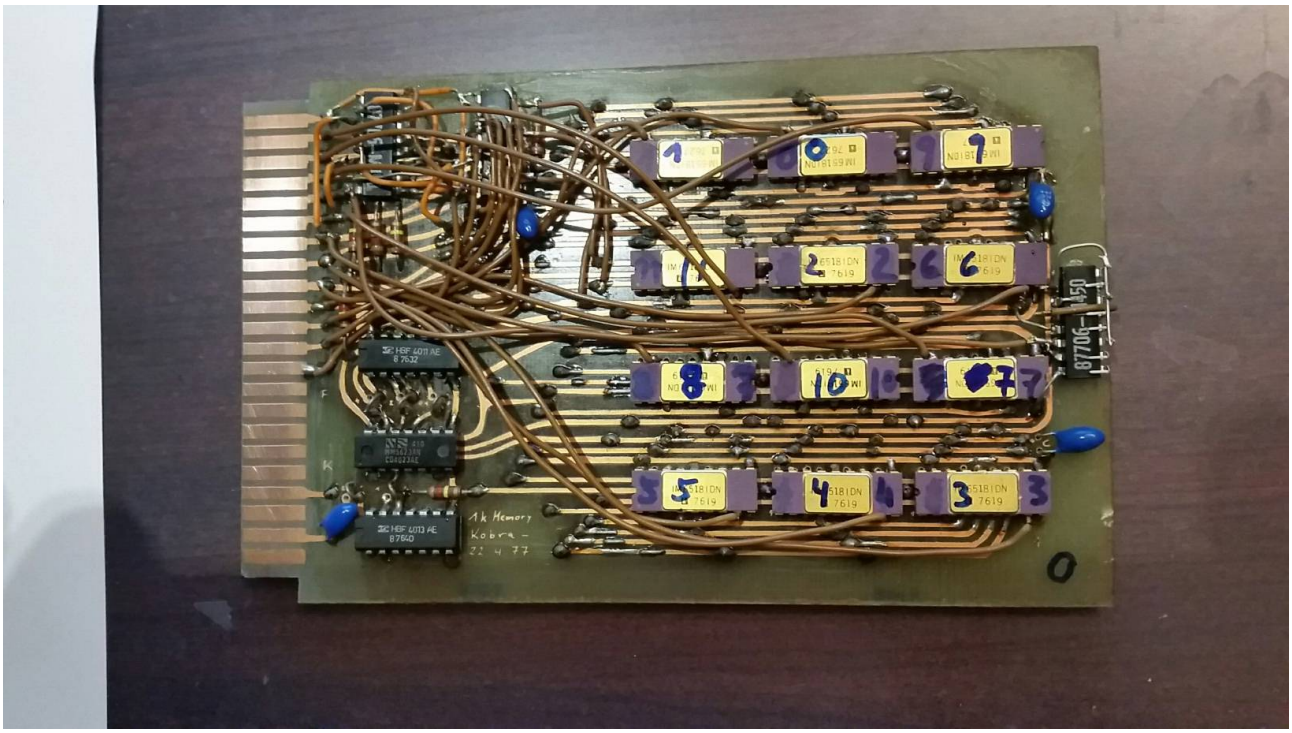
Am Anfang benutzen wir für unseren IM6100-Computer als Ein-Ausgabegerät einen alten Fernschreiber mit Lochstreifenleser und -stanze. Später konnten wir uns ein gebrauchtes Terminal (Ampex 210+) leisten und benutzten einen Kassettenrecorder als Ersatz für die Lochstreifengeräte. Die Interfacekarte hierfür musste natürlich auch von uns entwickelt werden.

Aktuell verwende ich einen **Raspberry-Pi** als Massenspeicher, der über die seriellen Schnittstellen mit dem IM6100 kommuniziert (siehe weiter unten).

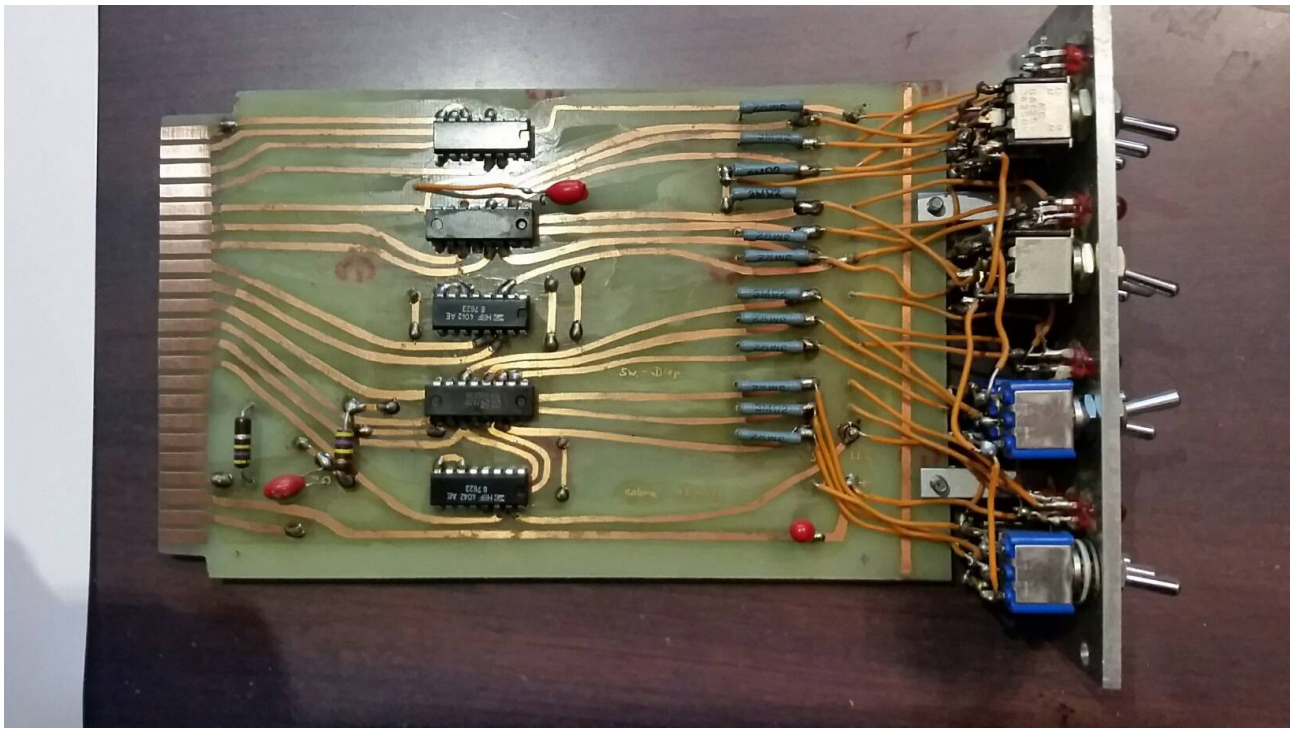


Der Eigenbaurechner auf der Basis des IM6100 Chipsatzes von Intersil – Da die ICs in CMOS-Technik ausgelegt waren, konnten wir mit Hilfe einer Batterie dafür sorgen, dass der Rechner nach dem Ausschalten nicht alles vergisst! Ein E-Prom kannte man da wohl noch nicht.

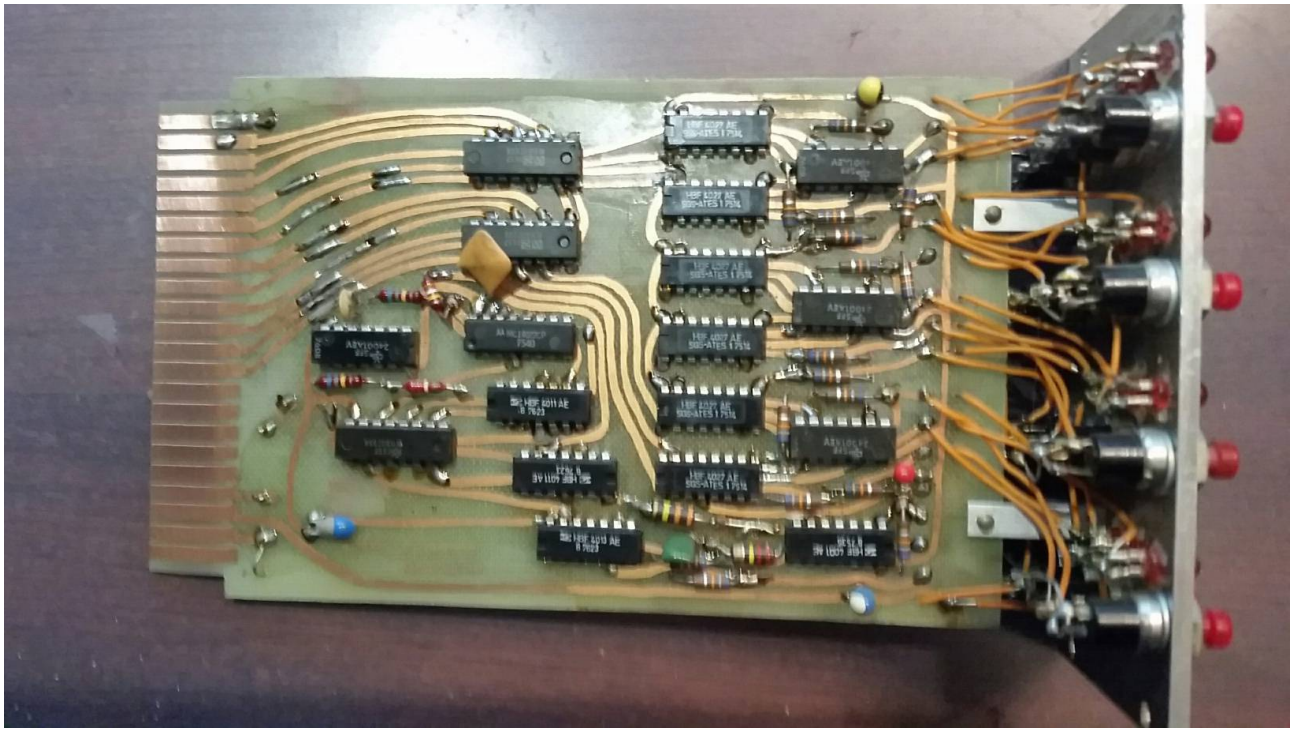




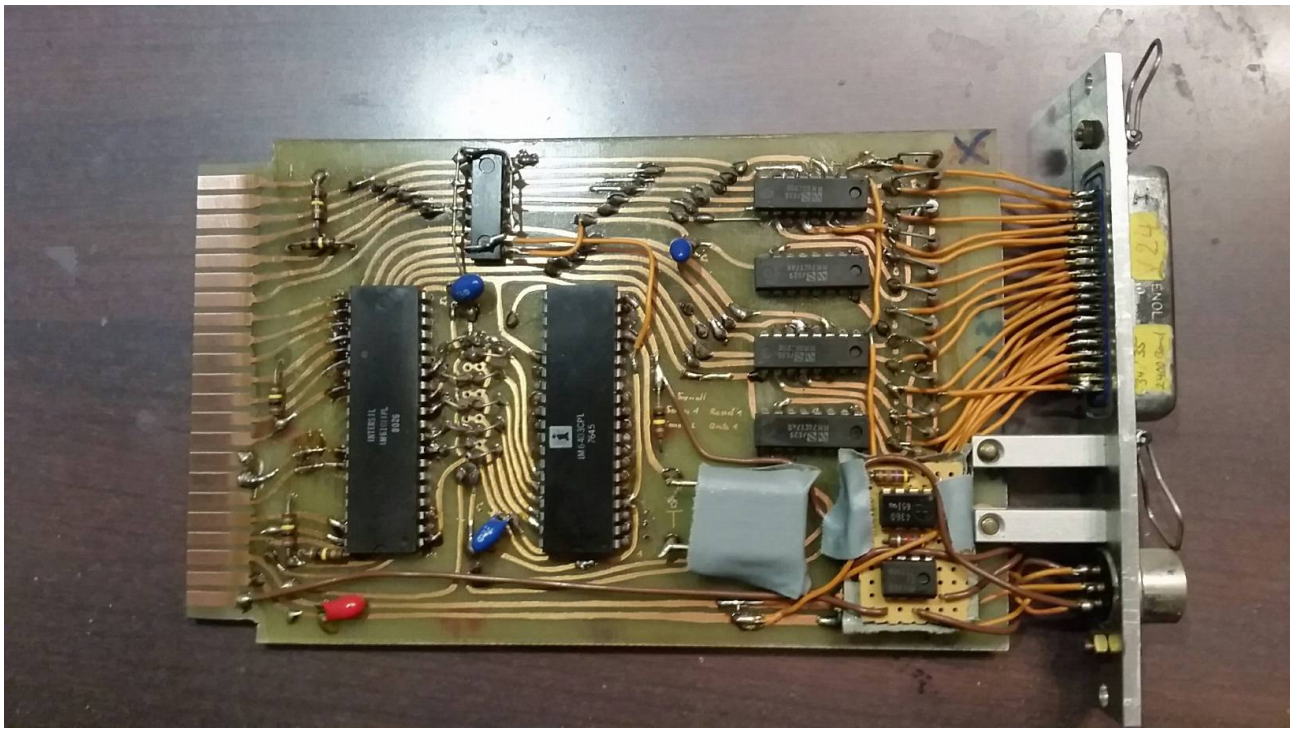
Eine der vier Speicherkarten. 1024 x 12 Bit waren für Studenten oder Assistenten damals auch nicht gerade preiswert.



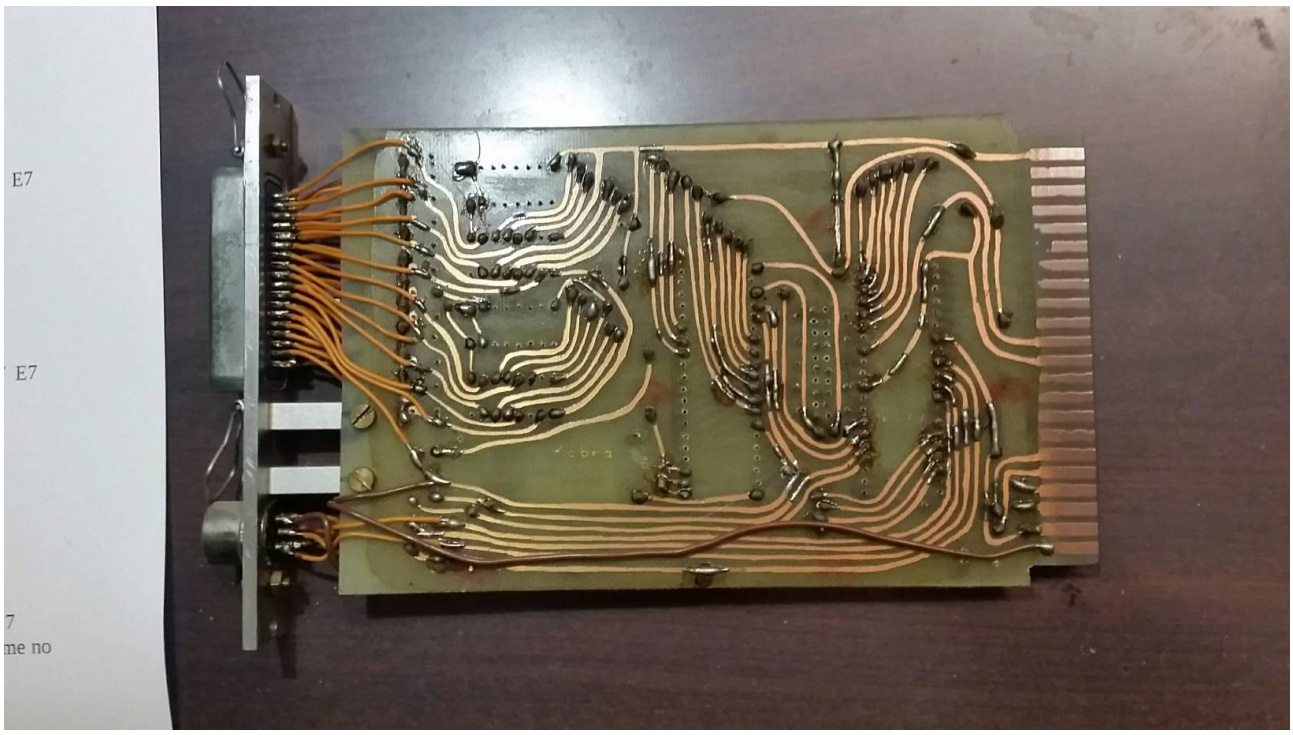
Unser Switch-Register. Die LEDs zeigen den Inhalt der aktuellen Adresse an.



Mit dem Adressregister kann man die gewünschte Speicheradresse anwählen und im CPU-Standby-Betrieb über unsere Hardware mit dem Switchregister den Binärcode eingeben und inspizieren.



Die Interface-Platine für den Anschluss meines Terminals. Auf der Zusatzplatine sind die Pegelwandler für die V24-Schittstelle des Terminals untergebracht. Außerdem steht eine 12 Bit Parallelschnittstelle zur Verfügung

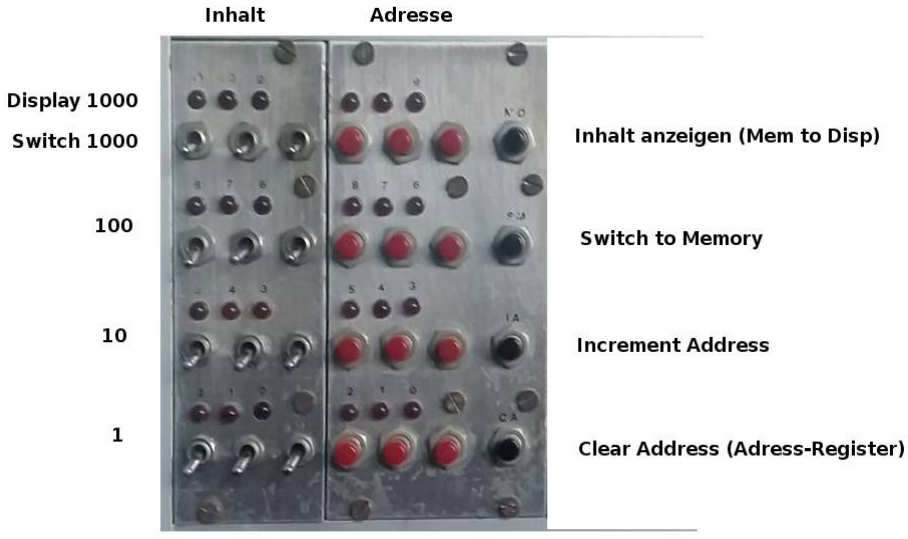


Die Platinenrückseite der Interface-Karte

Später habe ich einige Zeit in einer Firma für elektronische Datenerfassung gearbeitet (Frey Analog-Technik). Dort konnte ich mir auch einmal eine Speicher-Ersatzplatine professionell herstellen lassen.

Erste Schritte auf der elementaren Bit-und Byte Ebene: Das Programm ADDIERE

Mit Hilfe des Switch-Registers werden die in der u.a. Tabelle angegebenen Binärcodes im RAM-Speicher abgelegt. Die CPU befindet sich in dieser Phase im sleep-Modus (alle Aschlüsse Tree-State). Das Switchregister erlaubt die bitweise Eingabe **(Zahlendarstellung: OKTAL).**



Das elementare Programm ADDIERE

Adresse	Befehl	Symb. Adresse	Symb. Befehl	Kommentar
0200	7200	START	CLA	Clear ACCU
0201	1220		TAD 220	Addiere (zum ACCU) den Inhalt von Adresse 220 (transfer and add)
0202	1221		TAD 221	Addiere (zum ACCU) den Inhalt von Adresse 221
0203	3223		DCA 223	Speichere ACCU-Inhalt in Adresse 223 (deposit and clear ACCU)
0204	7402		HLT	Halt
0220	5		-	1. Summand
0221	2		-	2. Summand
0223	0		-	Speicher für Summe
7776	0200			
7777	5776		JMP I . -1	Indirekter Sprungbefehl via 7776 (176 auf akt. Seite)

Nach der Eingabe den Reset-Knopf und den RUN-Knopf drücken. Das Programm stoppt den Rechner nach Ausführung des Codes und mit Hilfe des Adressregisters und dem Knopf MEMORY TO DISPLAY kann die Summe der Berechnung in der Adresse 0233 abgelesen werden.

Nach einem Reset startet der Rechner immer mit der Adresse 7777. Dort steht üblicherweise ein indirekter Sprung über 7776. D.h.: In 7776 steht die Startadresse des auszuführenden Programms.

Um mit fertigen Programmen arbeiten zu können, muss mit dem eben beschriebenen Vorgehen der sogenannte RIM-Loader im Speicher abgelegt werden (Bootstrap-Loader)

Das RIM-LOADER-Programm FÜR KASSETTE (mit Initialisierung SC 36) ist also manuell einzugeben (siehe RIM-Loader Listing).

Alternativ: Das RIM-LOADER-Programm für den Raspberry PI (mit Initialisierung SC 32) ist manuell einzugeben. Es unterscheidet sich nur in den „6er-Befehlen“, aus 6361 wird 6321 usw.

Beim ersten Start ist die Startadresse wegen der Interface-Initialisierung) mit 7743 zu wählen, danach ist die SA des Rim-Loaders: 7755.

Switchregisterabfragen 7404, 7604 (OR/Load) funktionieren auch mit unserem SR (OSR,LAS).

Programm **RIM-LOADER** für den IM6100 (PDP-8-Nachbau, 12-Bit-Rechner) **Startadresse beim 1. Mal 7745**
Select-Code 36 (für Audio-Interface) Select-code 32 (für serielle Schnittstelle 5V) – Adapter 5V/3,3V erforderlich für Raspberry-Anschluss!

Eingabe dieses Programms über Switch-Register (falls Batteriepufferung beendet). SC 32 (oder 36) wird initialisiert. Nach dem ersten Durchlauf (mit Interface-Initialisierung) gilt automatisch **die neue Startadresse für den RIM Loader 7755** (X1) (dann ohne Interface-Initialisierung).

Adresse	Befehl	Symb. Adresse	Symb. Befehl	Kommentar
0006	0	ADR	0	Adresse
7743	7755	W7755	7755	Neue Startadresse
7744	0360	W360	360	Interface init
7745	6007	Start1	CAF	Accu und Link werden gelöscht
7746	6325		WCRA(32)	Write 0 to Control Register A
7747	1344		TAD W360	Addiere zum ACCU W360 (360)
7750	6335		WCRB(33)	Write 360 to Control Register B
7751	6320		READ(32)	1. Schrottzeichen lesen und weg
7752	7300		CLA CLL	
7753	1343		TAD W7755	
7754	3376		DCA 7776	
7755	6322	Start2	SKP1(32)	Skip if Flag 1 is set
7756	5355		JMP . -1	
7757	6320		READ(32)	1. Zeichen
7760	7106		CLL RTL	Clear Link, Rotate 2 Left
7761	7006		RTL	
7762	7510		SPA	
7763	5355		JMP X1	Trailler
7764	7006		RTL	Kein Trailer
7765	7421		MQL	(löscht auch ACCU)
7766	6322		SKP1(32)	
7767	5366		JMP.-1	
7770	6320		READ(32)	2. Zeichen
7771	7501		MQA	OR
7772	7420		SNL	Link == 1: Adresse
7773	3406		DCA I Adr	
7774	3006		DCA Adr	Neue Adresse festlegen in 7775
7775	5355		JMP Start2	
7776	7745		Start1	Startadresse des Rim Loadres
7777	5776		JMP I . -1	

RIM-Format :
Trailler and Leader nur gelocht in Kanal 8 danach, 2 Byte (je 6 Bit) Location, 2 Byte Content
1. Byte der Location: nur hier ist das 7. Bit gesetzt!

Mit Hilfe des RIM-Loaders können nun weitere Programme eingelesen werden. Als erstes sollte man das Interface-Testprogramm (iftest.rim) laden und ausführen, damit alle Interfaces initialisiert werden (Flag-Polarität u.ä. festlegen).

Nun können alle RIM Programme eingelesen werden. Mit den ODT-Programmen können BIN-Programme (als Speicherauszug) erzeugt werden. Zum Einlesen ist der BIN-Loader zu benutzen.

Eine Alternative zum Kassettenrecorder: ZOOM Digitalrecorder verwenden. Die Dateien müssen aber zur Wiedergabe mit Audacity auf 0dB Lautstärke verstärkt werden! Bei der Wiedergabe muss der Pegel auf 100 Skt eingestellt werden (Volume). Mit beiden vorhandenen Kassettenrecordern funktioniert die Speicherung. Wiedergabepegel auf Maximum, ggf Kopfhörerbuchse verwenden statt Diodenbuchse.

Als WAV-Dateien vorhandene Programme (im RIM-Format):

STE-000.wav	**BINLOADER.RIM** (SA 7600, 7600-7713) SC 36, Eigene Version von 1978 zum Einlesen von mit ODT erzeugten Programmen
STE-001.wav	**IFTEST.RIM** (SA 200, 200-232), zur Interface-Initialisierung (SC 32,34,36)
STE-002.wav	**NIMM.RIM** (SA 200, 200-2346) Das Nimm Spiel als **Demo-Programm** für das Terminal (SC 34)
STE-003.wav	**MEM2RIM.RIM** (SA 6600, 6600-6701(20,21)) Speicherinhalt wird im RIM-Format auf Kassette (SC 36) geschrieben. Die erste Mem-Adresse ist in 20 und die Endadresse in 21 vor dem Start abzulegen.
STE-004.wav	**ODT-HIGH.RIM** (SA 7000, 7000-7754) von **DEC** incl ODT-HIGH-Overlay und BIN-Loader !!!! SC36
STE-005.wav	**ODTLOW.RIM** (SA 1000, 1000-1577) von **DEC** muss anschließend durch Overlay-Einspielen ergänzt werden für Kassettenbetrieb über SC36. Ohne Overlay: Nur SC34
STE-006.wav	**ODTLOW_OVERLAY.RIM** (7714-7754) SC 36 Ergänzung für ODTLOW
STE-007.wav	**RIMBIN2MEM.RIM** (SA 200, 200-212) Liest vom SC36 beliebige Zeichenfolgen und legt sie im Speicher ab 1600 ab. Geeignet als Testprogramm für Kassettenaufzeichnungen

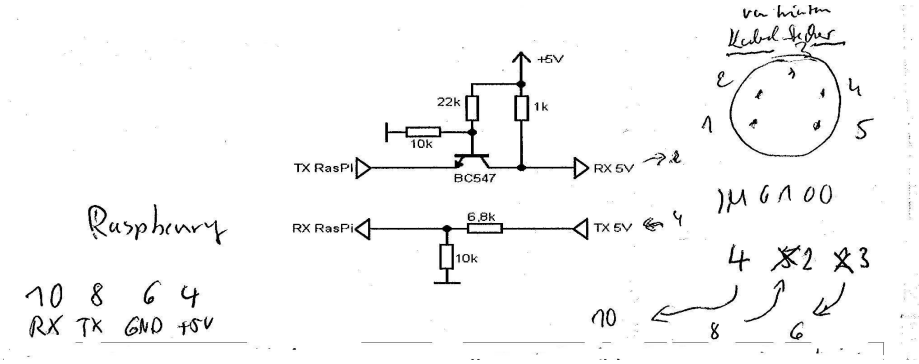
Generell ist festzustellen, dass das Arbeiten mit Kassettenrecordern als Datenspeicher ein sehr unschönes und fehleranfälliges Verfahren ist. Deshalb wurde mit Hilfe eines Raspberry Pi eine Lösung gefunden, die die alte PDP 8 – DEC- Peripherie simuliert:

Moderne Hilfsmittel für das Arbeiten mit unserem PDP-8-Nachbau

1. Der Raspberry PI simuliert Lochstreifenleser (Reader) und Lochstreifenstanze (Punch).

Hardware: Über einen Pegelwandler (3,3 Volt ← → 5 Volt) wird der Raspberry PI mit der seriellen Schnittstelle (TTL-Pegel) unseres PDP-8 Clones verbunden.

Mitte: Raspi und Pegelwandler (in Tablettenröhrchen) am IM6100-Computer (rechts). Links: Tastatur, Maus und Minimonitor für Raspberry Pi



Schaltbild des Pegelwandlers

Weitere Einzelheiten zum Anschluss: siehe unten

Software

Es wurde das Debian raspbian stretch – Image für den Raspbi 2 verwendet. Dessen serielle Schnittstelle wird in rc.local konfiguriert (Anschluss für im6100, SC32) und zwar über das Script **serielle**:

```
# konfiguriert die serielle Schnittstelle des raspian PI auf
# 1200 Baud, 8 Bit, 2 Stopbit no parity
# Raspi: Pin 10 (RX), Pin 8 (TX), Pin 6 (GND), Pin 4 (+5V)
# Dieses Script sollte beim Hochfahren über /etc/rc.local automatisch gestartet werden.
# ACHTUNG!!!! Raspbery arbeitet mit 3,3 V auf den Datenleitungen. Für den Anschluss an
# den IM6100 mit TTL (5V) Pegeln ist ein Pegelwandler zwischenzuschalten!!!!
#
```

```
stty -F /dev/ttyAMA0 1200 cs8 cstopb -parenb
stty -F /dev/ttyAMA0 -echo
stty -F /dev/ttyAMA0 raw
```

Das Script reader:

```
# Simuliert auf Raspberry Pi die Reader-Hardware. Angeschlossen an die serielle Schnittstelle
#
echo "READER für im6100 (SC 32). RIM-Datei ersetzt entsprechenden Lochstreifen."
if test -z $1; then
        echo "FEHLER! Aufruf ./reader dateiname.rim (ersetzt Lochstreifen)"
else
        cat $1 > /dev/ttyAMA0
fi
```

Das Script punch:

```
# Simuliert auf Raspberry Pi die Punch-Hardware. Angeschlossen an die serielle Schnittstelle
#
echo "PUNCH für im6100 (SC 32). Erzeugte RIM-Datei ersetzt entsprechenden Lochstreifen."
if test -z $1; then
        echo "FEHLER! Aufruf ./punch dateiname.rim (ersetzt Lochstreifen)"
else
        if test -e $1; then
                echo "Datei $1 (\"Lochstreifen\") existiert bereits!"
                echo -n "Datei $1 überschreiben? (j/n): "
                read antw
                if test $antw == "j"; then
                        echo "Beenden mit Ctrl. C"
                        cat /dev/ttyAMA0 > $1
                else
                        echo "abgebrochen"
                fi
        else
                echo "Beenden mit Ctrl. C"
                cat /dev/ttyAMA0 > $1
        fi
fi
```

2. Eine komfortable Entwicklungsumgebung für die PDP-8 (IM6100) auf dem Raspberry PI.

Es steht für Linux ein PAL III kompatibler Cross-Assembler (palbart) zur Verfügung. Asm-Programme werden mit palbart -r dateiname.asm ins RIM-Format übersetzt. PALBART gehört zur DEBIAN- und Ubuntu- Bibliothek.

Somit kann die Programmentwicklung mit jedem beliebigen Editor o.ä. komfortabel auf dem Raspberry Pi erfolgen. Die mit palbart generierte rim-Datei wird mit dem Script „reader“ auf den IM6100 (PDP-8) übertragen.

Das Script lstDrucken:

```
# Druckt palII (palbart) - Listings (lst) im Querformat
# und mit konvertiertem Zeichensatz, unterdrückt FormFeed
#
if test -z $1; then
        echo "FEHLER! Aufruf lstDrucken dateiname.lst"
else
        cat $1 | iconv -t ISO8859-1 -f UTF-8 | tr "\014" "\012" > .y
        mv .y $1
        a2ps -1r --chars-per-line=130 $1
fi
```

Das Script lst2asm:

```
# Das script schneidet aus vorliegenden palIII Listings (z.B. odthigh.lst aus dem Internet)
# den Assembler-Code aus
# zwecks Weiterbearbeitung. Das ist eine Möglichkeit um den Disassembler d8tape, der nicht
# mehr aktualisiert wird, zu umgehen.

if test -z $1; then
        echo "FEHLER! Aufruf lst2asm dateiname.lst"
else
        cat $1 | grep -v " errors" | cut -c19-150 | grep -v "  Page "
fi
```

3. Raspberry PI als Terminal

Hierbei wird der Raspberry PI nicht nur zu Punch und Reader, sondern zum „elektronischen Teletypewriter“. Es wird in einem weiteren Fenster das Programm MINICOM gestartet:

minicom -b1200 -D /dev/ttyAMA0 -8

Es ist allerdings eine eigene Zeichenkonvertierungstabelle mit dem Programm zu generieren, da die ASCII-Zeichen bei DEC immer eine gelochte „128“ hatten (oberstes Bit gesetzt).

Beispiel: Für Zeichen 47 [/], in 47, out 175 (47+128) **und** 175 [], in 47, out 175

Inneres und Äußeres müssen für jedes Zeichen definiert werden.

Anleitung für die DEC-ODT-Programme (Octal-Debugging-Technique):

DEBUGGING WITH ODT

Kobra

To begin the debugging run, first be sure that the BIN Loader is in core memory, then load the binary, ODT tape followed by the binary tape of the user program.

Command Summary

nnnn/	Open location designated by the octal number nnnn.
/	Reopen latest opened location.
RETURN key	Close previously opened location.
LINE FEED key	Close location and open the next sequential one for modification.
↑ (SHIFT/N)	Close location, take contents of that location as a memory reference and open it.
← (SHIFT/O)	Close location, open indirectly.
Illegal character	Current line typed by user is ignored, ODT types: ? (CR/LF).
nnnnG	Transfer program control to location nnnn.
nnnnB	Establish a breakpoint at location nnnn.
B	Remove the breakpoint.
A	Open for modification the location in which the contents of AC were stored when the breakpoint was encountered.
C	Proceed from a breakpoint.
nnnnC	Continue from a breakpoint and iterate past the breakpoint nnnn times before interrupting the user's program at the breakpoint location.
M	Open the search mask.
LINE FEED key	Open lower search limit.
LINE FEED key	Open upper search limit.
nnnnW	Search the portion of core as defined by the upper and lower limits for the octal value nnnn.
T	*Start motor, wait, punch trailler and SOM
nnnn;mmmmP	Punch a binary core image defined by the limits nnnn and mmmm.
E	Punch checksum, leader, *stop motor

ODT low 4, 1000 ... 1577 (Overlay 5,07714,07754)
SA: 1000

ODT high 4, 7000 ... 7577 (Overlay 5,07714,07754)
SA: 7000

SC 34 für Terminal, SC 36 für Cassette (Overlay)

* manuell vor "T" bzw nach "E"

Ohne overlay nur SC 34:

Abweichungen von DEC:

X231	6sc1
X232	6sc3
X012	6sc2
X014	6sc0

Weiteres zur Hardware: Interfaces:

Karte 32/33: seriell 0/5Volt: 1200 Baud (Raspberry Pi (Achtung 3,3V)), 8 Bit, no parity, 2 Stopbits

Pin1:
Pin2: in Ansicht Lötseite Einbaubuchse von hinten im Rechner
Pin3: Zählung
Pin4: out 1,2,3,4,5 **5**
Pin5: gegen den Uhrzeigersinn (Mitte 6)

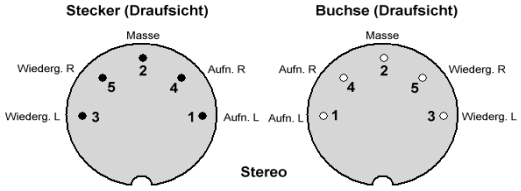
Karte 34/35: seriell -12V/+12V: 2400 Baud (Ampex Terminal), 8 Bit, no parity, 2 Stopbits

Pin1:
Pin2: out Ansicht Lötseite Einbaustecker von hinten im Rechner
Pin3: Zählung
Pin4: in 1,2,3,4,5 **1**
Pin5: im Uhrzeigersinn (Mitte 6)

V24 – 25 pol. Terminal-Anschluss: Pin 2 (Transmit Data) geht nach 4 am Rundstecker

Karte 36/37: seriell 1200 Baud, Frequenzumtastung 2000 Hz/4000 Hz, 8Bit, no parity, 2 Stopbits

Pin1: Audio Ausgang
Pin2: Masse
Pin3: Audio Eingang
Pin4:
Pin5:



links= Einbaubuchse
Lötseite im Computer IM6100

2 parallel geschaltete Buchsen

Audiobuchsen zum Anschluss des Kassettenrecorders oder eines ZOOM-Gerätes (Micro-Eingang!)
Mit dem Programm **iftest** werden alle drei IF-Karten (SC 32,34,36) initialisiert.

Hinweise zum Ampex-Terminal (V24):
==========
SHIFT Setup
Mit Pfeiltasten und Leertaste Parameter einstellen.
Mit SHIFT S (speichern) abschließen (oder mit SHIFT Setup verlassen, ohne zu speichern)

FDX A210+ GER KBON
CR = CR SCROLL ON
2400 Baud NONE BIDIR OFF STOP 2 NOPARCTR PAROFF

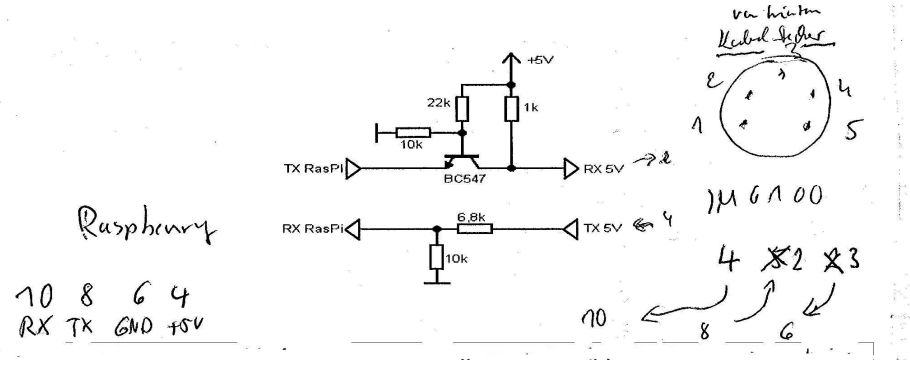
BIT8=1 bei ASCII und Verwendung der DEC-Software (ODT...) ist 8. Loch immer offen (TTY)
BIT8=0 für direkter RAW-Übertragung zum Bsp. auf Raspberry

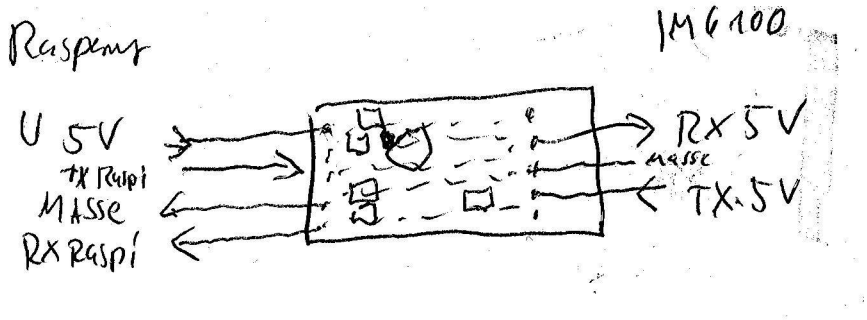
Knopfbatterie im Inneren ist leer, wenn beim Einschalten mehrmals „Checksum Error“ auftritt (Setup neu programmieren). Batterie ggf. austauschen durch Batteriehalter mit 2x 1,5 AA Batterien.

Rasperry Anschluss an ser. Schnittstelle (SC32) über Pegelwandler !!!!
Raspberry Initialisierung: ./seriell:

```
stty -F /dev/ttyAMA0 1200 cs8 cstopb -parenb
stty -F /dev/ttyAMA0 -echo
stty -F /dev/ttyAMA0 raw
```

cat /dev/ttyAMA0 zeigt auf dem Raspberry Monitor die ankommenden Daten
cat dateiname.rim > /dev/ttyAMA0 sendet Datei an IM6100 über die serielle Schnittstelle





Hinweise zu den In-Out-operationen:

IM6100 UART:

6007: CAF Clear all Flags: Accu, Link, IR-Requests werden gelöscht, IR-System ausgeschaltet

6SC0: READ1(SC) Echtes Lesen, keine OR Operation, Read-Sense wird gelöscht
6SC1: Write1(SC) Accu wird in IF geschrieben, aber NICHT gelöscht, Write-Sense wird gelöscht
6SC2: SCP1(SC) Read-sense wird abgefragt, ggf. skip
6SC3: SCP2(SC) Write-sense wird abgefragt, ggf. skip
6SC4: RCRA(SC) OR-Read Control Register A
6SC5: WCRA(SC) Write Control Register A (sollte Wert 0 haben)
6SC6: SFLAG1 Set Flag1 (Transmit line to high mark)
6SC7: CFLAG1 clear Flag1 (Trasmitline to low space)

SK=SC+1 !!!! Parallel-Schnitstelle

6SK0: READ2(SK) Read Parallelschnittstelle (s.o.)
6SK1: Write2(SK) Write Parallelschnittstelle (s.o)
6SK2: SKP3(SK) --- skip on read sense
6SK3: SKP4(SK) --- skip on write sense
6SK4: WVR(SK) --- Write Vectorregister für Interrupt
(die oberen 10 Bit von ACCU nach Register)
6SK5: WCRB(SK) ----- Write Controlregister B (sollte Wert 360 haben)
6SK6: SFLAG 3 – set flag3 (enable paper tape reader)
6SK7: CFLG 3 – clear flag3 (disable paper tape reader)

Original DEC PDP 8: Keyboard DeviceCode (SC) 3, Printer/Punch SC 4

TSF 6041 Skip if printer Flag = 1
TCF 6042 Clear printer flag
TPC 6044 ACCU → printer, print (flag is raised, when completed)
TLS 6046 CLEAR printe flag, AC->printer, print (Flag is raised when completed)
löscht Printer Buffer, setzt flag (bin bereit) Der Vorgang ist eine Art Reset.

TLS is combined TPC und TLS (Drucken und Stanzen zugleich)

KSF 6031 Skip on Keyboard Flag
KCC 6032 Clear ACCU and Keyboard Flag
KRS 6034 Keyboard → ACCU
KRB 6036 Keyboard → ACCU, clear Keyboard Flag

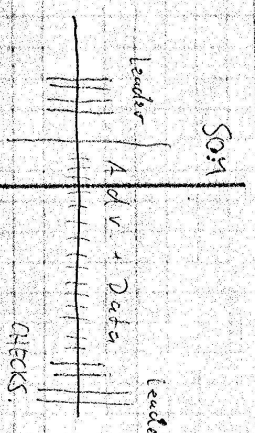
```
Ausgabe eines Zeichens:
*200
OUTPUT,     CLA CLL
            TLS
            TAD ZEICHEN
            JMS TYPE
            HLT
TYPE,       0
            TSF
            JMP .-1
            TLS
            CLA CLL
            JMP I TYPE
HOLD, 0
$

Einlesen eines Zeichens
*200
INPUT,      KCC
            JMS LISN
            DCA STORE
            HLT
LISN,       KSF
            JMP . -1
            KRB
            JMP I LISN
            STORE
$
```

BINLOADR von 1978

SC36 (Tape) getauscht gegen SC 32 für RaspiAnschluss

Address	Instr.	Symb. Addr.	Symb. Instr.	Comment
7600	6007	START,	CAF	Bin Loadre
1	6345		WCRA (34)	
2	1304		TAD W360	SC36
3	6355		WCRB (34)	
4	6360		Read (36)	
5	4244		JMS Input	
6	1306		TAD MSOM	
7	7440		SZA	
7610	5205		JMP .-3	Initialisiert
1	4275		JMS FETCH	SC 34 mit
2	5211		JMP .-1	— 0 — 360 —
3	3310	GO,	DCA CHECKS	
4	1313		TAD CHAR	
5	3311		DCA W1	
6	4251		JMS Read	
7	3312		DCA W2	
7620	4275		JMS FETCH	
1	5235		JMP Exit	Wartet auf SOM
2	4262		JMS ASS	
3	7430		SZL	Prüft Error bits
4	5230		JMP .+4	
5	3707		DCA I Orig	Prüft CHecksum
6	2307		ISZ Orig	
7	7410		SKP	
7630	3307		DCA Orig	HLT:
1	1311		TAD W1	
2	1312		TAD W2	kein Fehler: Link = 7776 = 0
3	1310		TAD CHECKS	
4	5213		JMP GO	CHECKS. ERR: Link = 0
5	4262	EXIT,	JMS ASS	7776 ≠ 0
6	7041		CIA	
7	1310		TAD CHECKS	Bit Error: Link ≠ 0
7640	7100		CLL	7776 = Error word
1	3376	END,	DCA 7776	
2	7402		HLT	
3	5200		JMP START	
4	0	INPUT,	0	
5	6362		SKP1 (36)	
6	5245		JMP .-1	
7	6360		Read (136)	
7650	5644		JMP I Input	
1	0	READ,	0	
2	4244		JMS Input	
3	3313		DCA CHAR	
4	1313		TAD CHAR	
5	3305		AND K7400	
6	7440		SZA	
7	5271		JMP ERR	



Address	Instr.	Symb. Addr.	Symb. Instr.	Comment
7660	1313		TAD CHAR	
1	5651		JMP I READ	
2	0	ASS,	0	
3	1311		TAD W1	
4	7106		CLL RTL	
5	7006		RTL	
6	7006		RTL	
7	1312		TAD W2	
7670	5662		JMP I ASS	
1	7200	ERR,	CLA	
2	1313		TAD CHAR	
3	7120		STL	
4	5241		JMP END	
5	0	FETCH,	0	
6	4251		JMS READ	
7	1303		TAD M200	
7700	7710		SPA CLA	
1	2275		ISZ FETCH	
2	5675		JMP I FETCH	
3	7600	M200	7600	
4	360	W360	360	
5	7400	K7400	7400	
6	7577	MSOM	7577	
7	0	Onism	0	
7710	0	CHECKS	0	
1	0	W1	0	
2	0	W2	0	
3	0	CHAR	0	
4				free: 7714 .. 7775
5				
6				
7				
7720				
1				
2				
3				
4				
5				
6				
7				
7730				
1				
2				
3				
4				
5				
6				
7				

KOBRA KOBRA KOBRA

Speicherbelegung:

ODThigh ODThigh_Overlay	1 2 3 4 5 6 7		ODTlow ODTlow_Overlay
	10 …. 17		Autoincrement-Adressen
	20 …. 177		Base-Page
IFTEST (200-232) SA200: init SC 32, 34, 36 and write - - SA 224: test read tape	200 …. 230		NIMM-Spiel SA 200, 200-2346 (10) RIMBIN2MEM SA 200 (200-212), Zeichenpuffer ab 1600
	226 …. 777		
	1000 …. 1577		ODTlow SA 1000
	1600 …. 6177		
Mem2rim(6600-6701) (20,21) – SA 6600	6200 …. 6777		
	7000 …. 7577		ODThigh SA 7000
BINLoader (1978) SC 36, SA 7600	7600 …. 7713		
ODThigh_Overlay ODTlow_Overlay (sind aber zu unterscheiden, entweder low oder high!)	7714 … 7754	7743 ….. 7754	RIMLOADER SA 7743 zur Initialisierung SC 32/(36) nur beim 1. Mal
		7755 …. 7775	RIMLOADER SA 7755 ohne Interface-Initialisierung
BINLOADER		7776	Fehlermeldung
		7777	

SC32: Raspberry Pi. SC36: Kassettenrecorder

Wir haben Ende der 1970 und Anfang der 1980er Jahre noch viele eigene Programme geschrieben. Joachim Brandenburg hatte sich auch noch mit Hardware zur Datenerfassung mit dem 6100 beschäftigt und eine eigene Programmiersprache entwickelt. Leider sind die meisten Arbeiten nur auf Audio-Kassetten gespeichert worden. Viele davon sind heute nicht mehr lesbar, einige sind auch „verschwunden“, darunter auch der von mir entwickelt kleine BASIC-Interpreter, von dem ich noch ein Manual besitze (s. Anhang).

Dr. Bernd Kokavecz
Berlin

Dr. Rainer Joachim Brandenburg
Köln

ANHANG

B A S I C - A

Manual

Hardware: Terminal SC 34
Kassette SC 36

Einleitung

Basic A ist eine 4 - k Version für den Mikroprozessor IM 6100. Der Interpreter läßt sich auch für den DEC -Rechner PTP - 8 auslegen. Er weicht nur unwesentlich vom Standard-Basic ab, wodurch einige Einschränkungen, aber auch einige zusätzliche Möglichkeiten auftreten.

Es ist beabsichtigt bei Gelegenheit eine weitere Version zu erstellen, die noch einige Verbesserungen aufweisen wird.

Starten des Basic - Interpreters

Das absolute Programm %BASIC wird mit Hilfe des Betriebssystems geladen. Die Startadresse ist 200. Vor dem Start ist das Switch-Register auf 5451_8 zu schalten. Der Rechner meldet sich über das Terminal mit

READY

Soll Basic bei einem Restart den Textbuffer nicht löschen, so kann dann die Startadresse 210 benutzt werden.

Editor - Mode

Jede Programmzeile beginnt mit einer Zeilennummer zwischen 1 und 799.

Beispiel: 10 PRINT "HALLO!"

Inklusive CR können 80 Zeichen pro Zeile eingegeben werden. Das Zeichen _ (Underline)* schiebt den Coursor um eine Position nach links. Mit dem Zeichen Rub out wird die ~~zuletzt~~ gerade eingegebene Zeile gelöscht.

Die Reihenfolge, in der die einzelnen Programmzeilen eingegeben werden, ist beliebig. Sie werden im Textbuffer mit aufsteigenden Zeilennummern abgespeichert. Schon vorhandene Zeilen werden durch Neueingabe überschrieben. Soll eine Zeile gelöscht werden, so ist die entsprechende Zeilennummer allein einzugeben.

* Backspace

- Programmliste: Der Befehl LIST bewirkt die Ausgabe der Programmliste. Man kann die Ausgabe unterbrechen, indem man den Schalter HALT betätigt (Bedienfeld des Rechners). Vor dem RUN - Schalter darf in diesem Fall der Knopf RESET nicht betätigt werden!
- Der Befehl SCRATCH löscht das eingegebene Programm.
- Das Programm kann mit Hilfe des Befehls SAVE auf der Kassette gespeichert werden. Der Recorder ist vorher zu bedienen.
- Mit dem Befehl LOAD kann ein aufgezeichnetes Programm wieder eingelesen werden. Zuvor ist normalerweise das Kommando SCRATCH zu geben. Sollen mehrere Programmteile nacheinander eingelesen werden, so ist darauf zu achten, daß sie aufsteigende Zeilennummern haben, da die Programmteile beim Einlesen von der Kassette hintereinander, ohne Rücksicht auf die Zeilennummern, abgespeichert werden.

Der Befehl RUN startet das eingegebene Programm.

Fehler werden erst während der RUN-Phase erkannt.
Die Fehlerliste muß noch überarbeitet werden.

Variable

Basic - A kennt indizierte und nichtindizierte Real - und Integer- Variable:

A ... Z, AA ... ZA ... ZZ, AØ ... Z9	REAL
§A ... §Z, §Ø ... §9	INEGER
A(5), A(§), B(A), §1(4), §(A), §7(§1)	Indizierte

Als Index dürfen nur Konstanten, und nicht-indizierte Variable benutzt werden.
Integerbereich: ± 2047

REM

10 REM DIES IST EIN TESTPROGRAMM

Remarcs werden im Programmablauf überlesen

STOP

10 STOP

Der Rechner hält. Der Rechner wird mit dem RUN-Schalter
am Bedienfeld wieder gestartet. Vorher nicht RESET drücken!

END

100 END

Das Programm wird beendet, Basic meldet sich wieder mit READY.
END muß nicht das letzte Statement sein.

GOTO

20 GOTO 10

Unbedingte Sprunganweisung: Gehe zur Zeile 10

GOSUB

```
30 GOSUB 50
35 END
50 PRINT"HALLO"
55 RETURN
```

Mit Hilfe des GOSUB - Befehls ist es möglich, Basic-Unterprogramme aufzurufen. Im Beispiel wird nach Zeile 30 die Zeile 50, dann 55, dann 35 ausgeführt. Es können maximal 8 Subroutines ineinander verschachtelt werden.

RETURN

Return beendet ein Basic-Unterprogramm. Es erfolgt der Rücksprung in die dem GOSUB-Aufruf folgende Zeile.

LET

```
10 LET A = 5 + S1/12.3E-1
```

Das Let-Statement weist der Variablen A den Wert des rechten Terms zu. Der Term kann folgende Operatoren enthalten: + - * / ((((( )))))

Punkt-Rechnung geht vor Strichrechnung, Klammern gehen vor. Konstanten können im "freien"-Format angegeben werden, Variable kennen jedoch kein Vorzeichen.

Beispiele für freies Format:

3 3.4 +3.4 -2.1E+5 -2E6 3.987E-7 12.45 0.08

Nicht möglich ist: LET A = - R
Möglich: LET A = Ø - R oder LET A = -1 * R

IF ..... THEN

```
10 IF A = 5 THEN 100
20 IF A ≠ B LET C = 2
```

Es kann auf = ≠ > < abgefragt werden. Ist die Aussage wahr, so wird das zweite Teilstatement ausgeführt, andernfalls wird die nächste Zeile abgearbeitet.

THEN ist gleichbedeutend mit GOTO. Als zweites Teilstatement kann jeder Statementtyp benutzt werden.

Verknüpfung zweier if-Statements ist log. "und" Verknüpfung!

PRINT

```
10 Print "A = "; A,B
```

String-Ausgaben werden in " ..." gesetzt. Bei Variablen wird der Wert der Variablen ausgedruckt.
Ein , (KOMMA) zwischen zwei Variablen sorgt für einen Zwischenraum von 4 Zeichenplätzen.
Ein ; trennt ohne Zwischenraum
Ein ; am Ende des Print-Statements unterdrückt CR u. LF.

Integerformat:
§ 4 Stellen
§: 1 ASCI-ZEICHEN
Real-Format
A Technisches Format (4 Stellen vor, 3 Stellen nach Punkt)
A: Exponentialdarstellung

INPUT

```
12 INPUT A,B,§:
```

Die Werte für A B und § werden vom Terminal aus eingegeben. Die Variablen müssen vorher erklärt sein. Bei Integervariablen ist der Formatpunkt (:) möglich und bedeutet, daß ein ASCI-Zeichen eingelesen werden soll.

READ

```
13 READ Z,L
```

Liest weitere Werte aus dem Input-Buffer, die beim letzten INPUT schon eingegeben wurden. Ist der Input-Buffer leer, werden die Variablen nicht verändert.

DIM

```
10 DIM A(3),B(6)
```

Legt die Dimensionierung von Arrays fest.
DIM STATEMENTS MÜSSEN ALS ERSTE IM PROGRAMM STEHEN

DATA

```
11 DATA A(1)/1,2,4
12 DATA §(1)/2,6,8,9
13 DATA §1(1):ABCDEF
```

Zuordnung von Konstanten. Das 1. Element des Arrays steht hinter Data. Bei Integervariablen ist durch Angabe des Doppelpunktes eine Zuordnung von ASCI-Zeichen möglich. Achtung! Es kann nur eine gerade Anzahl von ASCI-Zeichen zugeordnet werden!!!

CALL

Das Call-Statement erlaubt den Aufruf von Assemblerroutinen. Diese Routinen werden mit Hilfe des Betriebsystems ab Adresse 5451_8 geladen. Vor dem Start des Basic-Interpreters wird mit Hilfe des Switchregisters die nächste freie Adresse hinter den Assemblerroutinen als Startadresse des Basic-Text-Buffers eingegeben. Die Routinen werden mit Hilfe des Call Statements aufgerufen.

```
10 CALL UE,IA,§9,Z
```

UE (Real!) ist die dezimale Startadresse der Routine. Danach stehen bis zu 5 Variable, deren Adressen der Routine übergeben werden. Die Routine hat Zugriff zu den Adressen über die Zellen 2,3,4,5,6,7.

Mit dem Aufruf `JMP I 60` gelangt man von der Assemblerroutine zum nächsten Basic-Statement.