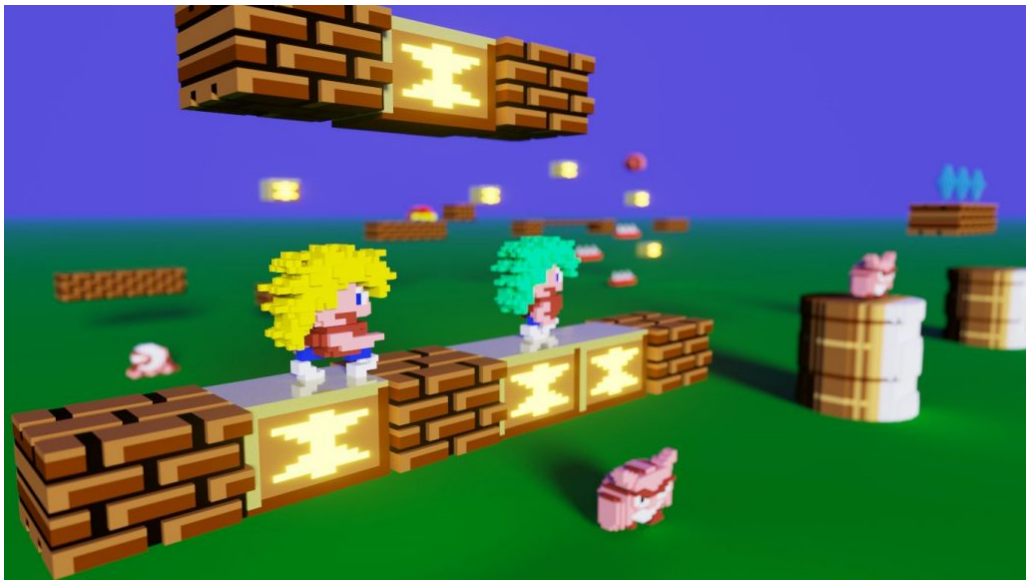


Entzauberung der Magie – Wie aus Bits Spiele werden

Gerrit Ludwig am Donnerstag, dem 1. Juli 2021

Ich habe schon als Kind gerne gebastelt. Damals waren das häufig Pappautos und Häuser aus den Micky Maus Heften oder halt Lego Steine, später dann „Lego Technik“. Ja, das darf ich sagen, denn damals gab es ja nur Lego.

Manchmal ist mir der Drang zu basteln auch zum Verhängnis geworden, wenn ich beispielsweise wissen wollte, wie etwas funktioniert und ich es zerlegt habe – natürlich ohne es wieder zusammen zu bekommen. Dabei hat ein wunderschöner Tischflipper sein Leben beendet.

Der C64

Es muss um 1985 oder sogar früher gewesen sein, als ich mit meinem Bruder zusammen „unseren“ C64 bekam. Ich konnte kaum richtig lesen und englisch schon gar nicht.

Also hat er mir Schritt für Schritt aufgeschrieben, was ich tun musste, um ein Spiel zu starten. Die Kassette in der Datasette zurückspulen, den Counter auf 0 stellen, bis kurz vor die Counterzahl spulen wo das Spiel ist, C= und RUN/STOP drücken, auf die PLAY Taste drücken, usw usf. Die Spiele waren einfach hintereinander abgespeichert, der Zählerstand stand neben den Spielnamen.

Das hat mein Bruder so bei seinen Freunden gelernt und wir haben das beibehalten bis wir eine Floppy bekamen. Kann auch sein, dass da noch was mit Turbo Tape war …

Der C64 war damals reine Magie für mich! Schon mein Bruder hatte viele Freunde mit einem C64, und ich natürlich auch. Also hatten wir wirklich wahnsinnig viele Spiele. Nicht alle waren gut, aber es gab ja immer wieder Nachschub.

Mein Vater hatte aus Holz eine Art „Tragerl“ mit 2 Stufen gebaut. Unten der Computer, oben drauf eine Art Regal für die Datasette, ein paar Kassetten, das C64 Netzteil (die Datasette hat ja keins) und sogar Einfräsungen für die Saugnäpfe der ersten Joysticks (erst später hat mich mein Kumpel vom „Competition Pro“ überzeugt). So war immer alles aufgeräumt und der C64 wurde gelegentlich im Wohnzimmer über ein langes Antennenkabel am Fernseher genutzt.

Ich hatte immer die tollen Farben hoch gedreht, der Ton war natürlich überirdisch damals.

Nach vielleicht 1 oder 2 Jahren kam der C64 dann in mein Zimmer. Erst an einen eigenen Fernseher (mein Onkel war Fernsehtechniker und hatte einen eigenen kleinen Laden), später dann sogar an einen Monitor.

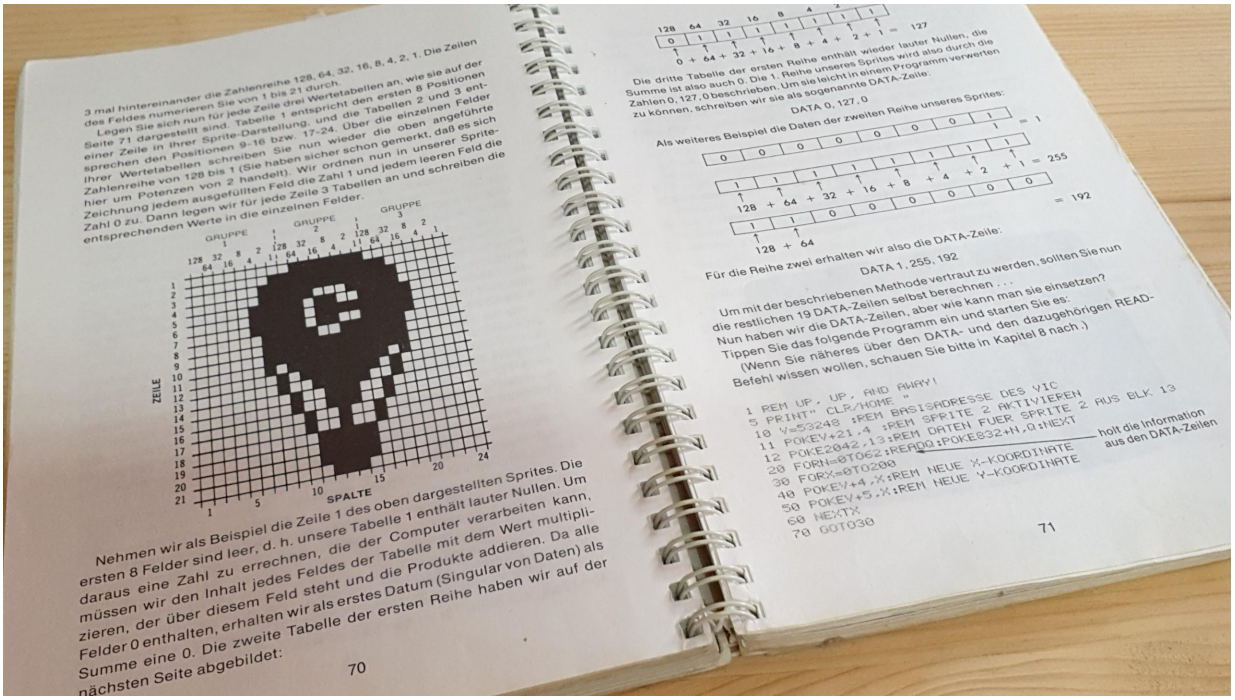
Und als sich mein Bruder einen HiFi Turm kaufte, hat er mir netterweise die alten Decks seiner Stereo Anlage überlassen.

Der C64 hing dann natürlich am Verstärker und das „Go, go go go!“ Von den BMX Kids aus den 30 cm (oder mehr?) Lautsprechern zu hören war das Größte damals.

M a g i e !

Homecomputer hatten damals so weit ich weiß alle ein mehr oder weniger gutes Benutzerhandbuch in dem das eingebaute BASIC mit vielen Beispielprogrammen zum Abtippen beschrieben wurde. So auch der C64.

Viele kennen vielleicht noch das berühmte Sprite Programm, in dem ein Ballon mit Commodore Logo über den Bildschirm flog. Wahnsinn, so etwas selbst gemacht zu haben!



Das berühmte C64-Ballon Sprite aus dem Handbuch. (Bild: Gerrit Ludwig)

Irgendwie zumindest, denn es war auch irgendwie ziemlich kompliziert. Zumindest dachte ich das damals mit 8 oder 9 Jahren. Vor allem natürlich die Binärzahlen, wenn man selbst gerade in der 3. oder 4. Klasse ist.

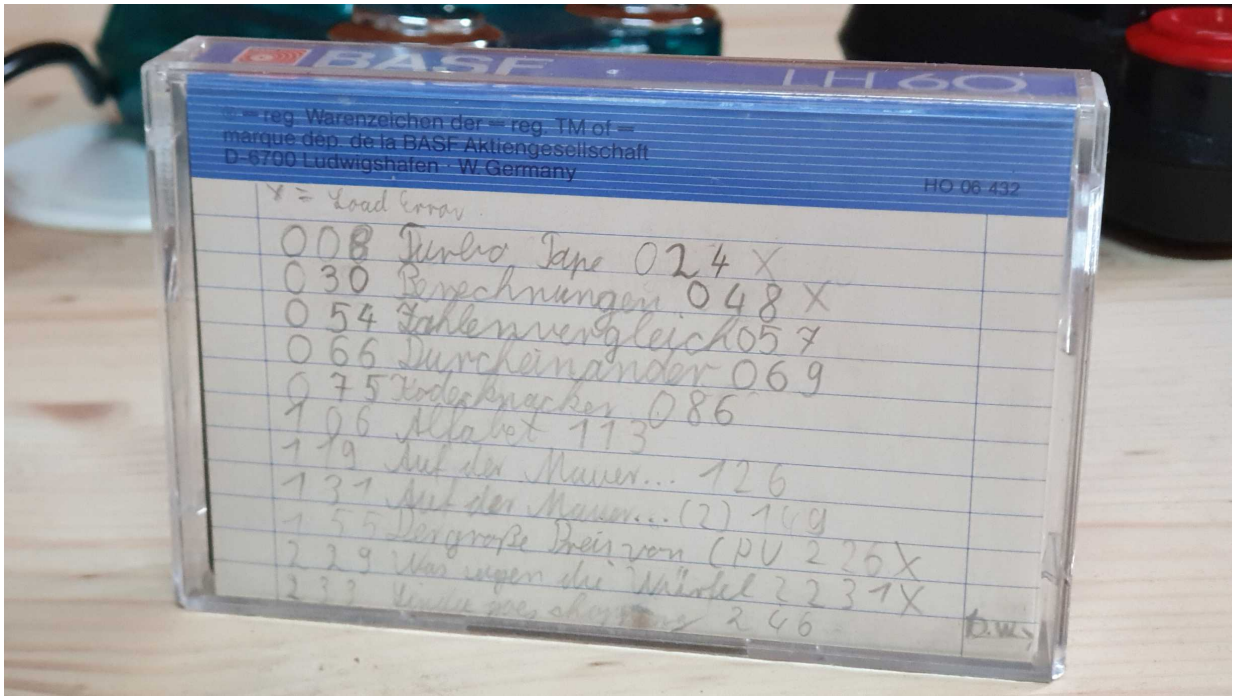
Und in diesen Magazinen, die ich mir sporadisch mal vom Taschengeld leistete, war immer zu lesen, dass das BASIC des C64 eigentlich totaler Schrott war um damit Spiele zu programmieren. Nicht mit diesen Worten, aber es war so gemeint. Und ich zeige später auch, warum.

Um Spiele zu programmieren musste man Assembler benutzen. Aber wie lernt man das als 11, 12 oder 13-Jähriger?

Ich leider gar nicht, obwohl ich schon immer wissen wollte, wie das alles entsteht.

Ihr wisst so gut wie ich, was es für Knaller damals gab. Gian(n)a Sisters, The Last Ninja, Test Drive, Ghostbusters (! Hahahahahaaaa) usw.

Das Beste, das ich mit meinem BASIC zustande bekommen hatte, war eine Art „Animation“. Also eine Folge von immer neuen PETSCII-Bildern, die ich auf den Monitor brachte. Ein Humpty Dumpty – muss wohl auch irgendwo abgetippt worden sein – und dann ein „Rennen“ mit 2 Punkten, zusammen mit dem eintönigen Brummen eines Motors, auch eine Demo aus dem Benutzerhandbuch. An mein geliebtes „Ralley Speedway“ (das auch in einer frühen Folge „Knight Rider“ zu sehen ist) kam das alles natürlich nicht ran.



Meine „frühen Werke“. (Bild: Gerrit Ludwig)

Letztes Jahr habe ich auf Archive.org herausgefunden, dass es in England sogar Kinderbücher gab um Assembler zu lernen (!) Die englischen Kinder haben halt auch den Vorteil, dass sie schon Englisch können. Und mit der richtigen Erklärung ist Assembler kaum schwieriger als das C64 BASIC, da man hier sehr viel sowieso nur mit PEEK und POKE erledigen kann. Genau das selbe, was Assembler macht.

Wenn ich mich richtig erinnere, habe ich mein erstes Assembler Buch in der sogenannten „Studienbücherei“ an meinem Gymnasium gesehen. Ein Mathe/Physik Lehrer (wer auch sonst …) hatte seine alten Bücher dafür gespendet, ebenso seine alte Sammlung „64er“ Magazine, immerhin 3 vollständige Jahrgänge. Ich habe heute noch einige Fotokopien von Artikeln und Lösungen aus diesen Heften.

Ich hatte zwar später in das Buch gesehen und ein bisschen mit dem Maschinensprache Monitor meines (damals immerhin 120 DM teuren) „Action Replay Mark VI“ Freezer Moduls herum experimentiert, aber der Funke zündete nicht mehr so richtig. Das Buch war unnötig umständlich geschrieben und dann auch noch schlecht erklärt. So wurden zum Beispiel auf etwa 50 Seiten erst mal alle Assembler Befehle vorgestellt, ohne dass man wusste, wie man damit etwas anstellen kann. Wie soll man mit einem Akku Spiele programmieren?

Ich weiß nicht mehr genau, wann es einsetzte, aber langsam ebbte bei mir ja auch das Interesse am C64 ab. Obwohl das aus heutiger Sicht auch schon relativ spät war, hatte ich doch erst 1990 das Geld für einen Amiga mit viel Zubehör zusammen.

Die jugendlichen Jahre damals waren irgendwie intensiver als das langweilige Erwachsenenleben, in dem jeder Bürotag gleich ist und die Jahre so vor sich hin dümpeln. Man hat heute das Gefühl, in einem Jahr Kindheit ist so viel mehr geschehen.

Ein Amiga 2000, vom Commodore Fachhändler in einer 60km entfernten „Großstadt“ eingerichtet mit Flickerfixer, Multisync Monitor, Tintenstrahl Drucker, Festplatte und der

Händler hat mich sogar von einer 286er Brückenkarte „überzeugt“. MS-DOS, richtiges Arbeiten, riesige professionelle Softwarebibliothek, Bürotauglich, usw.

Die habe ich leider nicht richtig zu nutzen gewusst weil die ja nur eine CGA Grafik emuliert hat und ich das Konzept von dem ganzen Zusatzgedöhns, das man beim PC brauchte, damals auch noch nicht so richtig verstand. Ich wollte ja zuerst nur spielen. Und dieser eine, grelle Andy-Warhol-Cyan-Magenta-Albtraum den ich hatte war furchtbar.

Den Amiga hat man ein geschaltet, sein Spiel geladen und gut wars. 32 oder mehr Farben, 8 Kanal Stereo Sound aus der Stereo Anlage, freie Sprachausgabe mit „Say“, ein Traum!

Oder halt mit der Workbench, die mein Händler schön eingerichtet hatte, gearbeitet – Texte geschrieben, gemalt, 3D Bilder gerendert, mit CLI und Shell herumexperimentiert, die Zeit einfach genossen.

Und weil es beim Amiga „ein bisserl mehr“ sein durfte, hatte ich auch später lange kein Geld mehr, um neben den Amiga noch einen PC zu stellen. Bei den Anzeigen in Spielemagazinen für Gaming-PCs war „Nomen est Omen“. Ein 386er Komplettpaket wurde für 3000, ein 486 für 4.000 DM angeboten.

Den Amiga allerdings dafür weg zu geben kam nicht in Frage. Zum einen hat man ja kaum etwas dafür bekommen, zum anderen wäre das, als wenn man seinen besten Freund verkauft. Und statt zu sparen habe ich immer wieder Geld für den Amiga ausgegeben.

Ein „richtiger“ PC, ein AMD K6-200, kam dann erst mit Hängen und Würgen zur Facharbeit ins Haus. Diese 200 Mhz haben halt Word leichter gewuppt als mein mit 25 Mhz Turbokarte aufgerüsteter Amiga sein „Amiga Office“ (sic).

Aber den Zauber eines Amiga oder C64 hatte diese seelenlose Kiste nicht mehr.

Obwohl ich mir für das Ding eine 3Dfx-Karte kaufte und die Spiele wirklich klasse waren. Es hat Spaß gemacht, aber es war nichts Magisches dran.

Die ersten richtigen Schritte mit Assembler

In der Berufsschule lernte ich dann richtig Assembler. Bei der Durchsicht alter Unterlagen musste ich feststellen, dass ich tatsächlich eine ganze Menge Wissen im Berufsleben vergessen habe. Mittlerweile bin ich mir ziemlich sicher, dass es sich hier um 8080 Assembler handelt, was mir zwischenzeitlich völlig entfallen war. Es ist allerdings dem 6502 – und auch dem Z80 – Assembler recht ähnlich, da es sich bei allen um 8-Bit Prozessoren handelt. Wie eben jede Landessprache und ihre Dialekte auch.

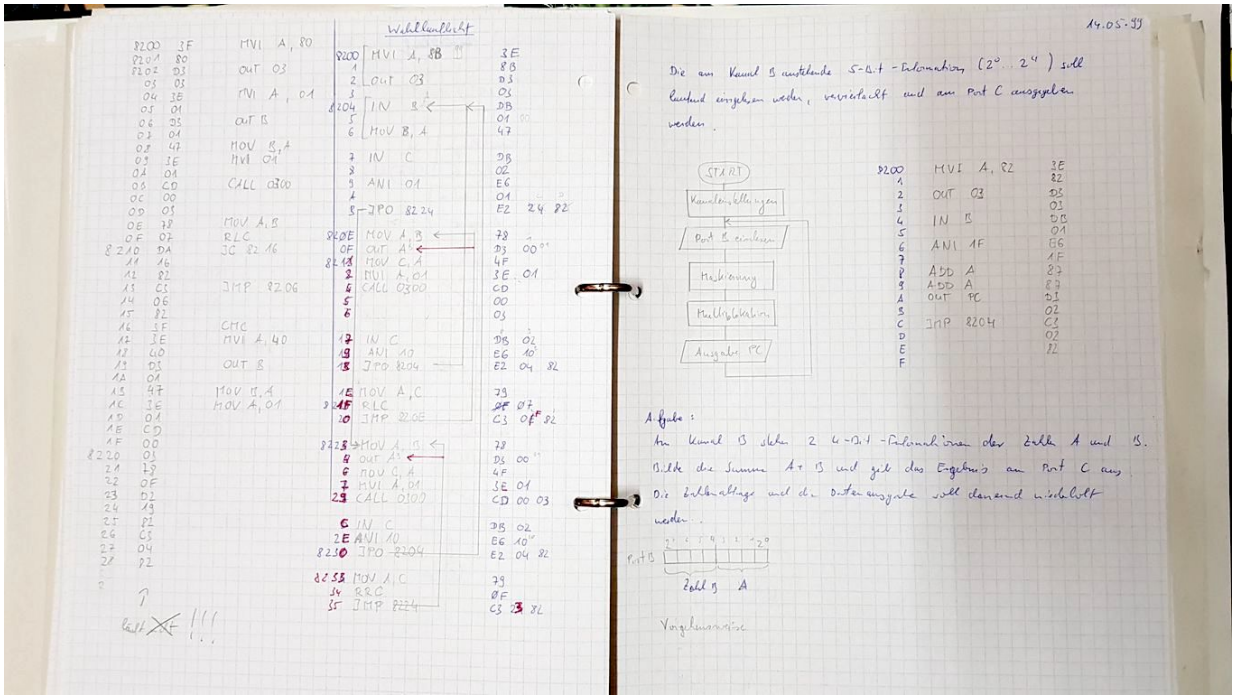
Wir haben die Programme in den sogenannten „Mnemonics“, also den Assemblerbefehlen, auf Papier geschrieben, diese per Hand (bzw. Taschenrechner) in Hex Werte umgerechnet (eine Art Do-it-yourself-Compiler) und sie dann in das Tastenfeld der „Assembler Kiste“ getauften Geräte, die auch nur ein Hex Tastenfeld und ein 8 Bit Hex Display hatten, eingegeben.

Die Dinger hatten glaube ich 3 parallele Ein- und Ausgänge und mein ganzer Stolz war mein Lauflicht-Programm, das wie KITT‘s Scanner über zwei 8 Bit Felder an zwei Ausgängen immer hin- und zurück lief und per Schalter an einem Eingang zusätzlich gesteuert werden konnte.

Speichern konnte man an den Kisten übrigens nichts, wenn der Strom aus war, war das ganze Programm weg.

Auf der Arbeit musste ich dann Java lernen und damit programmieren und der Assembler war schnell wieder vergessen.

Bis mich 2018 dann die Retrowelle traf. Ich hatte einen vermeintlich guten und sicheren Job, eine annehmbare Wohnung und mir in den Kopf gesetzt, jetzt endlich mal einen C64-II zu kaufen. Und dann nicht irgendeinen, sondern das Modell, das die „PETSCII“ genannten Sonderzeichen wie das Original vorne an den Tasten hat. Leider wurden es dann durch unsachgemäße Handhabung drei solcher 64er … Original Netzteil böse! Pfui! Aus!



Ein Lauflicht-Programm aus meiner Schulzeit, allerdings nicht ganz das hier beschriebene. (Bild: Gerrit Ludwig)

Moderne Retro–Programmierung in 6502 Assembler

Ausgelöst hatte diese Retro Euphorie die Entdeckung einer deutschen (!) Webseite, die sich mit genau diesem Thema beschäftigte: alternder Programmierer entdeckt seinen Original C64 aus der Kindheit neu und lernt Assembler. Und schreibt darüber!

Die Aktivität auf der Seite ist mittlerweile leider wieder eingefroren, und das Spieleprojekt wurde nicht zu Ende geführt, aber es ist immer noch ein wirklich wunderschönes Anfänger- und Nachschlagewerk, das bei mir fast ständig in mindestens einem Tab offen ist: www.retro-programming.de

Mittlerweile nahm mein Leben eine scharfe Linkskurve und hat mich etwas aus der Bahn

geworfen, so dass wieder einige Zeit verging bis ich im letzten Jahr in einer Dokumentation über mein Lieblingsspiel „Elite“ das Buch sah, mit dem David Braben das Programmieren gelernt haben will. Ein kurzer Check beim großen A und – ein Antiquariat hatte es für ein paar Euro im Angebot. Meins.

Natürlich ist es ein Buch über den BBC Micro, der allerdings wie der C64 mit einem 6502-Prozessor ausgestattet ist. Und er hatte ein Traum von einem BASIC! Man hatte dort schon Prozeduren mit Parametern und lokalen Variablen, while Schleifen und was weiß ich noch alles eingebaut. Und natürlich Befehle zur Grafik. Und er hat einen 80-Spalten-Modus (!) Perfekt für lange Texte. Er hatte einen Expansion Port, für den Tüftler heute tatsächlich Raspberry Pies als Zusatzprozessor anbauen, aber zahlreiche Emulatoren können den Charme der alten Geräte kostengünstiger einfangen.

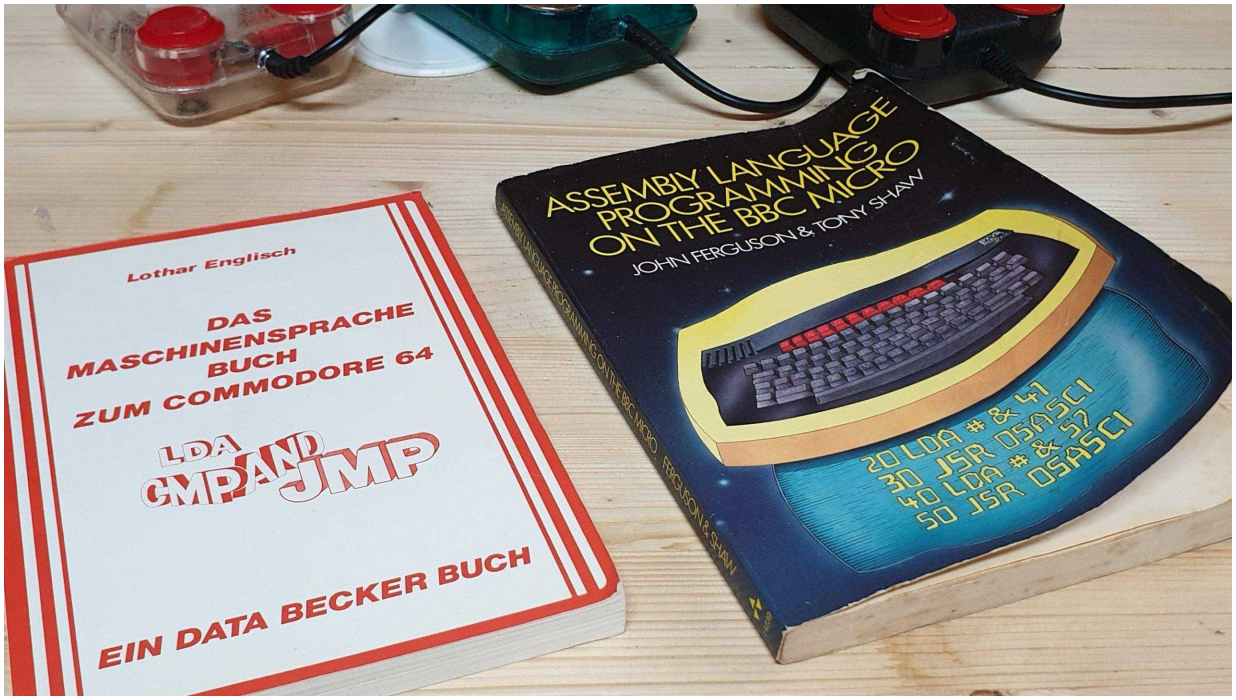
Das britische öffentliche Fernsehen, die BBC, hatte in den 80er Jahren eine Technikoffensive gestartet, da man Angst hatte, von den Amerikanern abgehängt zu werden. Dazu hat es Programmiersendungen kreiert, und mit den Schulen zusammen gearbeitet, um schon den Kleinen das Arbeiten mit dem Computer und das Programmieren beizubringen. Diese Sendungen waren auf den BBC Micro abgestimmt, der wohl auch in jeder Schule zu finden war. Dieser Computer wurde extra für dieses Projekt entwickelt und hatte eben auch den damals üblichen 6502 Prozessor.

Folgt man allerdings ein paar britischen Twitter Accounts bekommt man schnell mit, das diese Computer für den damaligen Normalhaushalt allerdings viel zu teuer waren und viele „nur“ einen ZX Spectrum besaßen, der wesentlich günstiger war (und durch den Z80 Prozessor ein etwas anderes Assembler) hat.

Die komplette Sendereihe wurde vor einiger Zeit veröffentlicht und sollte eigentlich dauerhafter Bestandteil des BBC Archivs sein. Leider ist es zurzeit offline: bzw. BBC Computer Literacy Project Archive

Die beste Eigenschaft des Acorn BASIC, das auf dem BBC Micro läuft ist allerdings die Tatsache, dass man ohne zusätzlichen Compiler ein ganz normales 6502 Assembler Programm mittels BASIC schreiben oder beides auch mischen kann. Auf dem C64 undenkbar. Hier ist eine externe Assembler-Suite nötig oder die gefürchteten BASIC-Lader, in denen gaaaaaaanz viele DATA zeilen ins RAM gepoked werden.

Nachdem ich mit den wichtigsten Grundlagen des Buches durch war und nur noch die seltsame Grafik des Micro behandelt wurde, kam ich wieder zu meiner damals gefunden Webseite über den C64 zurück und begann, das Tutorial durch zuarbeiten.



Zwei meiner C64 und BBC Micro Assembler Bücher. (Bild: Gerrit Ludwig)

Das ist witzig, dachte ich mir, damit könnte man doch ein bisschen mehr machen und somit war die Idee für ein eigenes Spiel in Assembler geboren. Ich habe also nach gut 30 Jahren endlich meinen Traum verwirklicht und angefangen, dahinter zu kommen, wie denn diese tollen Spiele entstehen.

Zugegeben, das entstandene Spiel sieht nicht wirklich toll oder aufwändig aus und hat nicht mal Ton, aber es steckt doch sehr viel Arbeit – und viel Freude – drin.

Ich habe es veröffentlicht, obwohl ich weiß, dass es nicht mit den anderen, teils atemberaubenden Projekten von heute mit halten kann (Beispiel „Sam‘s Journey“), und die wenigen Downloadzahlen bestätigen mir, dass kaum jemand daran interessiert ist. Aber hey, ich habe ein richtiges Assembler Spiel veröffentlicht.

Und ich bin immer noch dankbar, dass es den „besungenen Helden“ gefallen hat und mich niemand verklagt ? Mehr Infos gibt‘s im Spiel.

Wer mich von Twitter kennt (@CptSparky360) könnte die Adresse schon mal in meinem Profil oder diversen Tweets Ende letzten Jahres gesehen haben: cptsparky360.itch.io/puzzlewuzzle

„Wat iss en Dampfmaschin?“

Jetzt aber: wie kommt ein Bit auf den Bildschirm? Und was ist das überhaupt?

Nun, ich weiß eigentlich, dass die Videospielgeschichten–Fans genau wissen, was Bits, Bytes und Sprites sind. Aber ich möchte ja aus der Perspektive schreiben, die nur „Modul einschieben und einschalten“ oder LOAD „*“,8,1 kennt.

Ein flinkes und irgendwie freches Bit haben vielleicht die älteren unter uns schon sehr früh in dem wunderschönen Film „Tron“ gesehen. Genau diese Filme haben auch die Magie der Computerspielpioniere befeuert. Das war die Fantasie, die uns heute verloren

gegangen ist.

Flynn‘s Bit antwortet auf seine Fragen mit „Ja“ und „Nein“, mehr kann es nicht. Die perfekte Art, ein Bit darzustellen.

Ich bin kein Ingenieur, deshalb kann ich auch nur das wiedergeben, was schon in so vielen Büchern und anderen Medien beschrieben wurde: ein Bit stellt irgendwo in einem Computerspeicher eine kleine Einheit in einem Transistor dar, die entweder mit Strom „durchflossen“ wird oder auch nicht:

Es hat den Wert 1 oder 0. Mehr nicht.

Organisiert man diese Bits nun richtig, kommt man zu den ganzen tollen Spielen, angefangen bei „Pong“ bis hin zu „Cyberpunk 2077“ udgl.

Nochmal kurz zur Erinnerung: wir fangen „hinten“ an zu zählen und gehen von rechts nach links. Ein Computer zählt immer ab 0, aber es ist natürlich trotzdem die erste Stelle. Manchmal ist das ein bisschen verwirrend.

Also, das erste Bit hat die Wertigkeit 1, jedes weitere das Doppelte. Die Wertigkeit der „gesetzten“ Bits (die auf 1 stehen) zählt man zusammen.

```
0000 = 0
0001 = 1
0010 = 2
0011 = 3 (1 + 2)
0101 = 5 (1 + 4)
1011 = 11 (1 + 2 + 8)
```

Hier stoßen wir schon auf ein kleines Problem, denn 11 braucht schon wieder 2 Stellen zur Darstellung. Deshalb hat man zur Erleichterung das Hexadezimalsystem erfunden. Es zählt 1-stellig mit Buchstaben weiter.

```
a = 10 = 1010 = 2 + 8
b = 11 = 1011 = 1 + 2 + 8
c = 12 = 1100 = 4 + 8
d = 13 = 1101 = 1 + 4 + 8
e = 14 = 1110 = 2 + 4 + 8
f = 15 = 1111 = 1 + 2 + 4 + 8
```

Jeder weiß, dass 1 Byte 8 Bits hat. Einige wissen vielleicht, dass es auch Halb-Bytes gibt, die man „Nybble“ oder „Nibble“ nennt und aus 4 Bits bestehen. Irgendwo und irgendwann habe ich mal den Ursprungs des Wortes gelesen und wieder vergessen, weil er unwichtig ist.

Ich meine, es bedeutet so viel wie „eine kleine Menge“, ein Bissen, oder etwas in der Art.

Für uns hat es hauptsächlich die Bedeutung, große Mengen von Daten möglichst platzsparend zu schreiben. Jeder kennt das Beispiel, dass ein Computer nur lange Folgen von Einsen und Nullen „sieht“, also wie bereits erklärt Strom oder keinen Strom.

Für uns ist das extrem unübersichtlich. Hier stelle man sich ein DIN A4 Blatt voller 0en und 1en vor. Keine Leerzeichen, keine Absätze. Das ist der Speicher unseres Computers.

Mit der hexadezimalen Darstellung können wir nun 1 Byte auf 2 Zeichen verkürzen, das macht ungeheuer viel aus und je nachdem wie lange man sich damit beschäftigt, geht das früher oder später genauso in Fleisch und Blut über wie der erste Zahlenstrahl in der Grundschule. Manche Menschen können sogar in Hex Kopfrechnen.

Wir nehmen immer ein Nibble und machen daraus eine Stelle unseres Hex Wertes.

```
1110|1001 = e9.
```

Beim C64 ist es gängig, Hex Werte mit einem $ Zeichen zu kennzeichnen, binäre Werte mit einem % Zeichen und dezimale gar nicht. Diese Darstellung trifft aber nicht überall zu!

In vielen C-ähnlichen Hochsprachen werden hexadezimale Werte meistens mit 0xE9 gekennzeichnet, wobei Klein- und Großschreibung oft egal ist und nur der Lesbarkeit (oder der Tippfaulheit) dienen. Der BBC Micro kennzeichnet diese wiederum mit einem ? Zeichen (!)

Gerade als ich diesen Text tippe, merke ich, wie schwer das auf einem deutschen Linux-PC unter Libre Office Writer zu bewerkstelligen ist. Grüße gehen raus an den guten @Finselbix.

In den 60er Jahren gab es noch keine (oder kaum?) Hochsprachen. Jeder Programmierer war ein Ingenieur und musste per Maschinensprache mit seinem Computer reden. Es war auch gar nicht unüblich, dies nur über Schalter zu tun, die quasi jedes Byte einzeln in den Computerspeicher übernahmen. Die Ausgabe erfolgte dann über Lämpchen, weshalb Computerschränke in alten (für die jüngeren Lesenden: SEHR alten) Filmen gerne so schön durcheinander blinken.

Es gab kein 104 Tasten-DIN–Keyboard.

Wer sich dafür interessiert, kann nach Altair 8800 und PDP-7 (und vermutlich früher) suchen und staunen. Und das sind meines Wissens schon moderne und bekannte Geräte, es gab eine Vielzahl an Computerherstellern, die alle ihr eigenes Süppchen kochten.

Hier war er – der mir lange unklare Unterschied zwischen Maschinensprache und Assembler. In den Zeitungen und Büchern wurde er früher auch gerne gleichbedeutend genutzt.

Doch ich habe es weiter oben schon erwähnt: die lange Folge von Nullen und Einsen im Speicher ist die Sprache der Maschinen. Zu unserer Erleichterung haben wir diese kodiert und uns leicht merkbare – natürlich englische – Begriffe dafür ausgedacht und sie „Mnemonics“ genannt. Geben wir aber diese ein, müssen wir sie für den Computer assemblieren. Widerstand ist hier wohl zwecklos …

Bevor es hier richtig losgeht … zur Entspannung … ein Diskettenlocher. (Bild: Gerrit Ludwig)

Jetzt aber: Hands on!

Und damit komme ich endlich zu unseren ersten drei Assembler-Befehlen.

```
LDA #$41
STA $0400
RTS
```

Das ist ein lauffähiges Assemblerprogramm. Ich bin mit so etwas jedes mal so leicht zu begeistern, dass ich am Liebsten „Komm, mach doch auch mit!“ in die Welt hinaus rufen möchte.

Und genau das mach ich jetzt auch, denn es ist gar nicht so schwer … Nur eben auf dem C64 etwas umständlich. Ich könnte jetzt zu einem C128 wechseln, denn der hat einen „Maschinensprache-Monitor“ eingebaut – nicht zu verwechseln mit dem Röhrenteil, das früher das Bild angezeigt hat – doch zum einen bin ich nicht damit aufgewachsen und zum anderen geht es auch mit dem C64 ganz gut.

Und zwar mit Hilfe von VICE, dem „Versatile Commodore Emulator“.

Hier muss ich schweren Herzens auf die letzte native Windows Version zurückgreifen, denn diese funktioniert für unseren Zweck am einfachsten und ist am leichtesten zu bedienen. (ich höre gerade Tim lachen).

Aus mir unbekannten Gründen haben die Entwickelnden angefangen, ihren VICE auf das „Gimp Toolkit“ (GTK3) umzubauen und damit auch irgendwie die Komfortfunktionen der alten Windows Versionen weg gelassen.

Wer mit machen will und sich mit VICE gut auskennt, nimmt natürlich, was er/sie will.

Wer mitmachen will (was ich nur empfehlen kann, denn dafür schreibe ich ja diesen

Beitrag) und sich nicht so gut aus kennt, macht am besten folgendes.

Wir nehmen die native Windows Version 3.2. Diese können wir aus dem offiziellen VICE Archiv runter laden:

Hier empfehle ich das „Binary for MS-Windows 32bit: WinVICE-3.2-x86.7z“ auszuwählen, zu entpacken und den guten alten x64.exe zu starten.

Wer will, kann schnell in den Einstellungen das Bild verbessern. Für die Lesbarkeit hilft es schon ungemein, die CRT Emulation abzuschalten.

Jetzt kommt der interessante Teil: wir öffnen den Monitor, über das erste Menü oder Alt-M.

Es sollte jetzt ein weiteres, kleineres Fenster aufgehen, das euch vielleicht an ein Linux Terminal oder ein DOS-Fenster erinnert. Es meldet sich mit einem seltsam aussehenden Prompt, der euch vielleicht schon ein bisschen an einen Hex Wert erinnert.

Ja, das ist eine Adresse im Speicher unseres emulierten C64. Sie ist 4-stellig weil sie nicht aus 8, sondern 16 Bit besteht.

Der Unterschied ist nicht so schwer, mit 8 Bits können wir 256 Zahlen darstellen (0 bis 255).

Das sind 2^8 Möglichkeiten. Mit 16 Bit kommen wir auf 2^16 mögliche Zahlen und das ist genau der Speicher eines C64, nämlich 64 Kibi-Byte.

Diejenigen, die wie ich die Verarschung durch die Speichermedienindustrie nicht mitmachen wollen, dürfen auch gerne weiterhin bei Kilobyte bleiben.

1 kByte war früher und ist auch heute noch 1024 Byte (!)

Wir „adressieren“ den Speicher des C64 von $0000 bis $FFFF. Und schon bin ich bei einem weiteren Vorteil von Windows: der Taschenrechner. (Tim, ich kann dich nicht höööreen – lalalala …)

Startet ihn mal und stellt ihn auf „Programmierer“. Damit haben wir immer alle Zahlen im Blick – dezimal, hexadezimal und binär.

Zusätzlich gibt es noch oktal, das als Zahlenbasis nicht 10, 16 oder 2 hat, sondern 8.

Hab ich allerdings noch nie gebraucht und kenne es nur von alten Kollegen, die erzählt haben, dass es mal Großrechner gab, die damit arbeiteten.

Einer der mittlerweile defekten C64-II mit SD2IEC und dem bösen original Netzteil. Außerdem hatte ich mir mal eine Final Cartridge III angesehen. (Bild: Gerrit Ludwig)

Wir gehen jetzt ins Menü des Monitors und aktivieren unter dem Punkt „View“ in VICE noch weitere Fenster: das „Disassembly“ und das „Memory“ Window.

In diesen beiden Fenstern suchen wir den Speicherbereich ab $0801. (Hier allerdings ohne $ dargestellt.)

Das Memory-Fenster scheint hier Leerstellen anzuzeigen, im Disassembly-Fenster sehen wir, dass hier überall der Befehl BRK steht. Das bedeutet einfach nur BREAK und hat vorerst für uns keine Bedeutung. Tatsächlich habe ich diesen Befehl bisher gar nicht bewusst eingesetzt, aber man sollte ihn kennen.

Wenn wir uns die Spalten mal ansehen, merken wir, dass wir ganz oft FF und 00 sehen. Das macht für mich die Faszination von Assembler aus. Es kommt nämlich immer darauf an, als was es der Rechner sieht!

$00 kann z.B. die Farbe schwarz eines Pixels oder des Hintergrundes bedeuten. Oder halt meistens einfach nur die Zahl 0 zum Rechnen und Zählen.

Aber der übersetzte BRK Befehl hat ebenfalls die Zahl $00. So hatte ich einmal den Fall, dass sich mein Programm beendete, obwohl es das gar nicht sollte. Ihr könnt euch denken, ich hatte es geschafft, eine $00 in den Speicher zu schreiben, die an dieser Stelle nicht hätte sein dürfen und der C64 hat es beendet – BREAK.

Wir nehmen jetzt unseren mühsam hervor gezauberten Monitor und schreiben das erste Programm direkt in den Speicher! Das ist etwas, was man normalerweise natürlich nie macht.

Aber es geht.

Dazu tippen wir …

```
a 0801 lda #$41
```

Wow, moment, das ist aber nicht das, was oben steht. Richtig, aber fast.

a ist ein Monitorbefehl und bedeutet „assemble“, dahinter kommt die Speicherstelle in der assembliert werden soll und dann erst unsere Mnemonics.

Leider springt bei mir das Disassembly-Fenster nach dem Druck auf Return an eine andere Speicherstelle, aber wenn wir zurück zu $0801 gehen, stehen da unsere eben eingegeben Werte.

```
0801 A9 41    LDA #$41
```

Der Monitor springt gleich an die nächste verfügbare Speicherstelle: $0803.

Das liegt daran, dass LDA ein Byte belegt und $41 noch ein Byte. Im Speicher stehen jetzt also 2 Byte Code.

Wir tippen weiter …

```
sta $0400
```

Jetzt haben wir einen Befehl und eine Speicheradresse eingegeben, die 16 bittig ist. Sie braucht also 2 Bytes. Der nächste freie Speicherplatz ist $0806, an dem wir noch

```
rts
```

eingeben. Fertig!

Gehen wir im Disassembly Fenster zurück zu $0801, sehen wir unseren Code. Auf der linken Seite sehen wir die in Hex kodierten Befehle für den C64. Es sind wieder nur Zahlen, aber wenn der Code hier gestartet wird, führt er unser Programm aus. Diese Zahlen kommen von den Entwicklern des 6502, es sind sozusagen die Befehle der Assemblersprache. Andere Prozessoren haben ähnliche, aber eben nicht zwingend dieselben Befehle.

Doch Moment, etwas stimmt nicht. Jede Zahl entspricht einem Befehl oder einer „echten“ Zahl.

Hier steht allerdings etwas seltsames.

```
0803 8D 00 04    STA $0400
```

Der C64 macht aus $0400 seltsamerweise nicht 04 00, sondern 00 04!

Das ist die sogenannte Prozessorarchitektur und diese macht jeder Hersteller, wie er will. Es gibt also auch Prozessoren, die wirklich 04 00 im Speicher stehen haben, aber eben nicht der C64.

Komischerweise wird das „Little Endian“ genannt, also die kleine Stelle endet hinten, was aber so wiederum nicht stimmt, denn hinten steht ja 04. Wikipedia sagt dazu treffend, es

müsse eigentlich „Little Starter“ heißen.

Hier kann ich leider auch nur sagen, ich bin kein Ingenieur und nehme es einfach so hin. Man gewöhnt sich dran. Es hatte wohl früher, als die Prozessoren noch in Kilohertz statt in Gigahertz gedacht haben, etwas mit der Verarbeitungsgeschwindigkeit zu tun.

Wichtig für uns ist, wir nennen es „Most significant Byte“ und „Least significant Byte“. Und dabei stimmt die Reihenfolge: 04 ist das MSB, 00 das LSB.

Wenn wir damit programmieren, kommt es darauf an, was wir machen. Wichtig ist, dass uns bewusst ist, dass im Speicher des C64 zuerst der kleinere Wert einer Adresse steht, denn in Assembler ist es kein Problem, irgendwo in den Speicher einfach eine andere Zahl zu schreiben und damit z.B. den Programmcode zu ändern.

Mit diesem kurzen Code-Beispiel haben wir schon den allergrößten Teil der Assemblerprogrammierung abgedeckt. Meistens werden nur Zahlen im Speicher hin- und her geschoben. Wichtig ist wieder, mit was diese Speicherstelle „verknüpft“ ist.

Also beispielsweise kann ich meinen Finger auf der Tastatur hin- und her bewegen und damit Buchstaben eingeben. Oder ich bewege ihn zum Ausschalter und der PC fährt herunter.

Ein Floppy Transportschutz. Die Laufwerke wurden damals damit verriegelt verkauft, auch heute noch habe ich gebrauchte Floppies bekommen, die damit versendet wurden. Vielleicht weiß ja jemand von den Lesenden, ob das wirklich etwas bringt? (Bild: Gerrit Ludwig)

Was haben wir jetzt eigentlich programmiert?

Sehen wir uns an, was das Programm macht. Wir können jetzt spaßeshalber noch einmal Return im Monitor drücken.

Wir haben keinen weiteren Befehl eingegeben und sehen wieder den Prompt vom Anfang. Nur kommt uns diesmal die Zahl etwas bekannter vor. Es ist die Speicherstelle,

an der wir zuletzt standen. Wir tippen ein …

```
g 0801
```

Schade, jetzt ist der Monitor weg und der C64 blinkt READY.

Haben wir etwas falsch gemacht? Nicht wirklich.

Wer will, kann nochmal den Monitor mit Alt-M öffnen. Es ist alles noch da.

Mit Cursor Up holen wir uns (wie unter Linux) den letzten Befehl zurück und führen ihn nochmal aus.

Diesmal schauen wir aber auf den C64 und sehen ganz kurz vor der READY. Meldung etwas auf blitzen.

Jetzt benutzen wir den C64 um unser Programm zu starten, es ist ja noch im Speicher.

RUN führt in diesem Fall zu einer Fehlermeldung, denn wir haben ja kein BASIC Programm eingegeben.

Wir arbeiten mit Assembler und haben jetzt direktere Möglichkeiten: Tippt folgendes ein.

```
SYS 2049
```

Der C64 meldet sich wie immer mit READY.

Aber er hat uns einen Gruß hinterlassen: in der oberen linken Ecke des Bildschirm sehen wir ein PIK Zeichen!

Unser Programm wurde erfolgreich ausgeführt.

Was ist passiert?

Nehmen wir den Windows Taschenrechner und schalten (zum Beispiel mit der Maus, erfreulicherweise geht aber auch Cursor und Space) auf Dezimal.

Jetzt tippen wir 2049 ein und sehen: 801 hexadezimal!

Mit dem BASIC Befehl SYS setzen wir den „Program Counter“ des C64 auf eine Speicherstelle und der läuft einfach los.

Der Programmzähler läuft sozusagen immer mit, damit der C64 weiß, wo im Speicher er sich gerade befindet. Nach jedem abgearbeitetem Befehl wird er um die entsprechende Länge des Befehls erhöht.

Was haben wir eigentlich programmiert?

Das Programm startet mit dem Befehl LDA – „LoaD Accumulator“ , lade etwas in den Akku. Gemeint ist hier keine wiederaufladbare Batterie, sondern die für uns wichtigste Speicherzelle in unserem 8 Bit-Computer.

Technisch gesehen rechnet der Computer in der ALU, der „Arithmetic Logic Unit“. Aber die spricht nicht mit uns. Unser Kontaktmann ist der Akku. In diesem können wir hauptsächlich Zahlen „erzeugen“, rechnen und eine Menge anderer toller Sachen machen.

Wir beginnen damit, den Wert 65 „in den Akku zu laden“, hexadezimal $41. Sehr wichtig ist hier das Rautezeichen / Doppelkreuz / Hashtag Zeichen, „Gartenzaun“, egal, wie man es nennen will.

Es bedeutet „direkt“ und macht den Unterschied zwischen einer konkreten Zahl und einer Speicheradresse. Und es ist ein Quell großer Freude, wenn man es vergisst und keine Ahnung hat, warum nicht das passiert, was passieren soll.

Der nächste Befehl ist STA $0400. „STore Accumulator“ in der Speicherzelle 1024.

Also speichere den Inhalt des Akkus in der Adresse 1024 (Hex $0400).

Diese Adresse ist sozusagen mit dem VIC-II Chip im C64 verknüpft. In diesem Augenblick bedeutet genau diese Adresse das oberste linke Buchstabenfeld des Monitors.

Es wird einfach weiter gezählt, 1025 ist das Feld daneben, 1024 + 40 = 1064 ist das erste Feld in der 2. Reihe, usw.

Der C64 kann 25 Reihen mit jeweils 40 Spalten ausgeben. Um das ganze noch etwas verwirrender zu machen, kann (und muss man sogar oft) in Assembler diese Adresse auch ändern, die Spalten auf 38 verringern oder die Zeilen auf 24. Aber das brauchen wir jetzt nicht.

Am Ende springen wir mit RTS – „ReTurn from Subroutine“ zum Aufrufer zurück. In diesem Fall also der BASIC Prompt. In BASIC oder vielen anderen Programmiersprachen wäre es das RETURN.

Die Kassette „C64 BASIC-Kurs.“ (Bild: Gerrit Ludwig)

Das geht doch auch alles in BASIC

„Okay, toll. Aber das geht doch auch alles viel einfacher“ könnten wir uns jetzt denken.

Und genau hier kommt der Teil, der mich so an Assembler fasziniert.

Die gleiche Ausgabe wie eben bekommen wir, wenn wir

```
PRINT CHR$(147) CHR$(97)
```

eingeben. Bildschirm löschen und ein PIK Zeichen schreiben.

Und jetzt komme ich auf das erste Beispiel in dem vorhin erwähnten Assembler-Buch zu sprechen.

Erfreulicherweise war nämlich meine Mutter die Bibliothekarin und wurde eines schlimmen Tages beauftragt, die relativ große Bibliothek in ein relativ kleines Zimmerchen umzuräumen und dabei den Großteil der teilweise sehr alten und teuren Bücher weg zuschmeißen. Was dann unter großem Protest und mit Hilfe weiterer Kollegen leider auch erfolgte. Deshalb habe ich ein paar Bücher und die 64er Sammlung an mich genommen. Meine Mutter natürlich auch, aber das volle Haus ist ein anderes Thema.

Wir geben an unserem C64 Emulator sicherheitshalber NEW ein.

Und danach folgendes BASIC Programm.

```
100 X = 0
110 A = X
120 POKE 1024+X, A
130 A = 1
140 POKE 55296+X, A
150 X = X + 1
160 IF X <> 256 THEN 110
170 END
```

Das Programm funktioniert ähnlich wie das Beispiel von vorhin: es schreibt nacheinander – allerdings statt nur einem – 256 Zeichen auf den Bildschirm. Diese werden dann noch über eine andere Speicherzelle in weißer Schrift dargestellt.

Wenn wir das Programm mit RUN ausführen, können wir dabei zusehen, wie ganz gemütlich Zeichen für Zeichen auf dem Bildschirm erscheint und sich danach der C64 wie gewohnt mit READY. meldet.

Und jetzt kommt der entscheidende Moment. Wir übertragen das Programm nach Assembler:

Also wieder den Monitor öffnen (Alt-M) und folgendes eingeben.

```
a c000 ldx #$00
```

Wir fangen diesmal bei $C000 an und schreiben 2 Byte an Befehlen, somit meldet sich

der Monitor mit der nächsten freien Speicherstelle $C002 und wir tippen einfach weiter.

```
txa
sta $0400,x
lda #$01
sta $d800,x
inx
bne $c002
rts
```

Ihr ahnt es schon, wenn wir fertig sind, können wir das Monitorfenster schließen, so dass VICE wieder zurück kommt. Wenn wir mit dem Taschenrechner nachsehen, was $C000 in dezimal ist, so kommen wir auf 49152.

Deshalb geben wir jetzt im C64

```
SYS 49152
```

ein und … die 256 Zeichen des C64 stehen alle sofort auf dem Bildschirm!

Wer nochmal den Vergleich sehen will, das BASIC Programm ist weiterhin im Speicher.

Wir können es mit

```
LIST
```

nochmal ansehen und mit RUN wieder ausführen.

Es bleibt aber dabei: Mit BASIC braucht das Programm laut dem Buch ca. 7 Sekunden, bis alle Zeichen da sind.

Mit Assembler sind sie quasi sofort da!

Wenn das mal nicht deutlich macht, wo der Unterschied zwischen BASIC und Assembler bei einem langsamen 8 Bit Rechner liegt (!)

Die beiden Programme sind so geschrieben, dass sie zeilenweise miteinander verglichen werden können, es passiert in etwa das gleiche im BASIC- als auch im Assemblerprogramm.

Da dies ja kein Assembler-Kurs ist, gehe ich jetzt nicht mehr so detailliert darauf ein.

Ich hoffe natürlich, dass ich damit bei ein paar Lesenden das Interesse auf Assembler geweckt habe und ihr euch vielleicht ein bisschen auf den Seiten mit den Assembler Kursen umguckt.

Ein selbstgemachter Dreher um den Tonkopf meiner Datasette zu justieren. Damals normal, heute etwas skurril. Habe allerdings dazu auch schon witzige YT Videos gesehen. (Bild: Gerrit Ludwig)

Aber was ist denn jetzt mit den Spielen?

Genau, da war ja noch was. Ein paar Zeichen aus dem C64 Zeichensatz machen ja noch kein „Giana Sisters“. Der Witz ist hier übrigens, dass Giana im Spiel falsch geschrieben wurde.

Aber wir sind der Sache schon viel näher, als ihr denkt.

Ich habe ja schon sehr oft von „PETSCII“ gesprochen. Die Kennenden wissen: das sind die lustigen Zeichen, die mit Hilfe der Commodore und der Shift Tasten ausgegeben werden können.

Der Name PETSCII setzt sich zusammen aus dem legendären „PET 2001“ Computer von Commodore, der bereits 1977 erschien und einer Form der ASCII Tabelle, die den Buchstaben und Tasten auf Computern Werte zuordnet.

Für uns interessant ist zunächst der „PET“.

Dieser hatte nicht wie der C64 2 * 256 Zeichen, sondern nur 128. Und diese waren fest im ROM verankert und konnten nicht wie beim C64 verändert werden. Auch sonst hatte er keine Möglichkeit für Grafik. Deshalb hat man ihm ein paar Sonderzeichen spendiert, um wenigstens so etwas wie Kästchen oder stilisierte Menüs zeigen zu können. Manch einer erinnert sich vielleicht noch an alte DOS-Programme, die etwas ähnliches gemacht haben.

Diese Zeichen hat der C64 immer noch, und dazu hat er nochmal weitere 384 bekommen. Da ist sowohl ein neuer Groß- und Kleinschrift Modus dabei (Umzuschalten mit Commodore und Shift), wie auch ein paar neue grafische Zeichen.

So, den entscheidenden Punkt habe ich jetzt „en passant“ mitgegeben: beim C64 lässt sich der Bildschirmzeichensatz verändern! Und das sogar in Farbe!

Nur verhält es sich leider so, dass der normale Zeichensatz auch beim C64 im ROM (wir wissen: Nur-Lese-Speicher) steht und von dort aus gelesen wird. Sollen Zeichen verändert werden, muss der Zeichensatz von einer Stelle im RAM (frei veränderbarer Speicher) gelesen werden. Da dies nicht in ein paar Zeilen Beispiel-Code abgehandelt werden kann, verweise ich auch hier wieder auf die tolle Webseite, wo das super erklärt wird.

Wie ich vorhin schon erwähnt habe, kann der C64 im Normalfall 25*40 Zeichen, darstellen. Jedes Zeichen ist 8 Bit breit und 8 Bit hoch. Aha! Wir benötigen also 8*8 Bit = 64 = 8 Byte im Speicher für ein einziges Zeichen.

Wir haben für ein „Charset“, also einen Zeichensatz, 256 verschiedene Zeichen. Somit brauchen wir 8 * 256 = 2048 Byte für einen kompletten Zeichensatz.

Malen wir damit den ganzen Bildschirm voll, brauchen wir 25*40 = 1.000 Byte. Das ist nicht allzu viel, denn wir haben ja noch weitere 63 kByte übrig.

Gut, wir brauchen nochmal die gleiche Menge an Farbinformationen, die auch in einem Speicherbereich für jedes einzelne Zeichen zu finden sind. Das haben wir vorhin kurz gesehen, aber ich bin nicht weiter darauf ein gegangen: in einer ganz anderen „Ecke“ des C64 Speichers stehen Farbwerte für jedes sichtbare Zeichen. Dieser Speicherbereich bleibt auch fest und kann nicht verschoben werden.

Mit der Darstellung durch bunte Zeichen brauchen wir pro Bildschirm also knapp 2 kByte wenn wir den Speicher für den Zeichensatz nicht mitzählen.

Nun hat aber der C64 auch einen Bitmap Modus. Das Wort kennen wir vielleicht von Windows–Grafiken, die relativ viel Speicher benötigen. Genauso ist es hier auch.

Der C64 hat ja 8 * 40 = 320 Pixel in der Breite und 8 * 25 = 200 Pixel in der Höhe. Wollen wir die alle im Speicher behalten, brauchen wir 320 * 200 = 64.000 Bit. Teilen wir das durch 8 kommen wir auf 8.000 Byte, also etwa 8kByte. Wollen wir das noch ein bisschen bunt, brauchen wir nochmal 1 weiteres kByte. (Genauer nachzulesen bei retro-programming.de)

Noch ein kurzes BASIC Beispiel

Wenn ich euch nochmal einladen darf, ein weiteres mal mit BASIC rumzuspielen, verspreche ich euch noch ein Aha-Erlebnis.

Gut wäre es, wenn auf dem VICE Bildschirm ein bisschen Text steht, aber der Cursor sollte nicht ganz unten sein.

Jetzt gebt folgendes ein.

```
POKE 53270,9
```

Euch sollte auffallen, dass sich der ganze Text einen Pixel nach rechts verschoben hat.

Geht mit den Cursor Tasten wieder nach oben und ändert die 9 in 10. Wieder bewegt sich alles einen weiteren Pixel nach rechts!

Wir können das problemlos bis 15 machen, ab 16 schalten wir den Multicolor Modus ein und es wird etwas unübersichtlich.

Wer an die Stelle zurück findet, kann beispielsweise wieder auf 8 stellen und alles ist wieder in Ordnung.

Na, ist das keine Magie?

Mein erster Scrollversuch – mit einem „geheimnisvollen“ podcastenden Biker… (das echte C64 Programm läuft natürlich flüssiger, aber selbst so eine kleine gif Animation braucht heute leider schon das zigfache an Speicher).

Was wir jetzt gesehen haben, ist das berühmte Hardware-Scrolling. Wir haben hier die Möglichkeit, in nur einer Speicherstelle den gesamten Text-Bildschirm zu steuern, ohne dass wir dafür umständlich programmieren und den Prozessor belasten müssen (wie bei anderen Homecomputern der Zeit). Noch interessanter dabei ist, dass hier tatsächlich auf jedes Bit geachtet werden muss. Wer will, kann z.B. statt einer Zahl zwischen 8 und 15 einfach mal 3 ein geben.

Oha, was ist passiert? Wir haben auf den 38 Zeichen-Modus umgeschaltet und das Bild verschoben.

Um den Normalzustand wieder herzustellen, poken wir wieder die 8 in den Speicher.

Zur Erinnerung, die 8 sieht in binär so aus.

```
%00001000
```

Und genau dieses Bit in der Adresse $D016 schaltet zwischen dem 38- und dem 40 Zeichen Modus um. Um ganz sicher zu gehen, könnt ihr nochmal die 0 poken.

Das Problem ist nur, dass BASIC weder Binär- noch Hexadezimalzahlen kennt. Wenn ihr euch jetzt an die binäre Darstellung erinnert, wisst ihr auch, dass die Bits 0 bis 2 für die Anzahl der zu verschiebenden Pixel zuständig sind: 0 bis 7.

Mittels des Hardware-Scrolling können Programmierende ein Bild im Textmodus beim C64 also um bis zu 7 Pixel verschieben.

Danach bleibt uns nichts anderes übrig, als jedes Zeichen einzeln auf eine neue Position zu setzen.

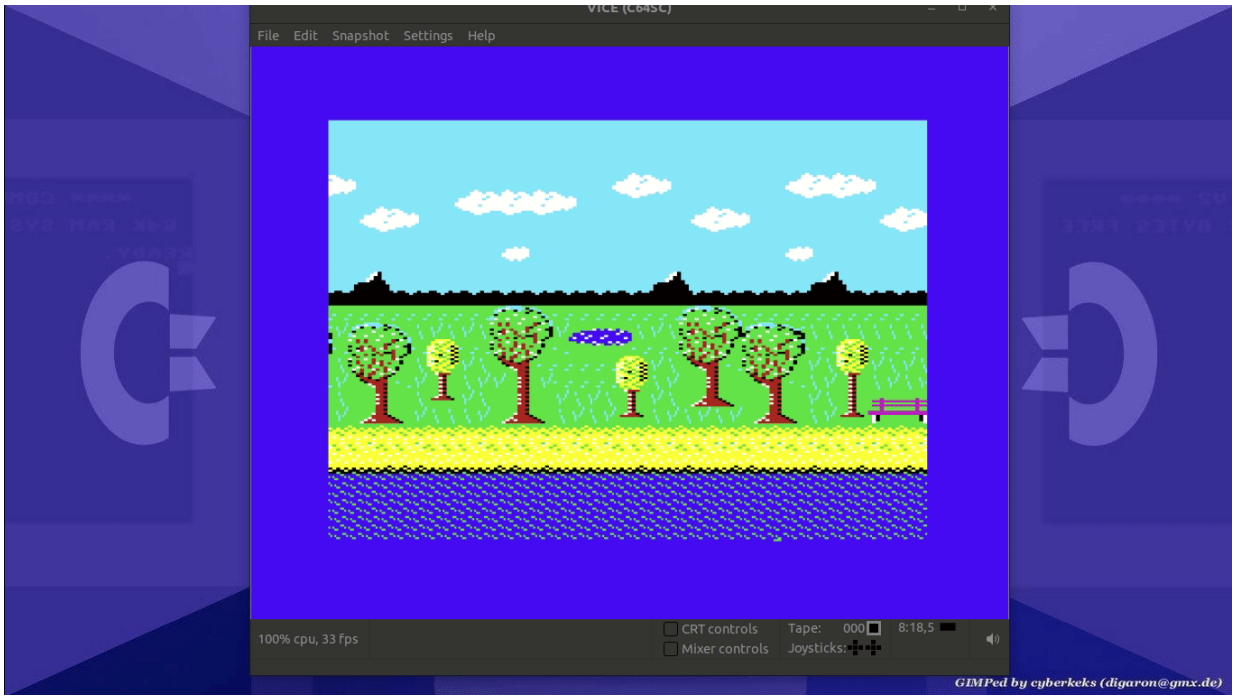
Also beispielsweise von Spalte 39 auf Spalte 38, von 38 auf 37, von 37 auf 36 usw. bis auf Spalte 1 (wenn wir zum Beispiel nach rechts laufen und das Scrolling entsprechend von 7 bis 0 geht).

Da dies im 40 Spalten Modus komische Effekte mit sich bringt, hat man den 38 Spalten Modus eingebaut. Damit verschwindet das Bild sauber im Rahmen und alles wirkt fließend.

Ist die erste Reihe verschoben, geht es mit der nächsten Reihe weiter. Aber wenn ihr euch das Beispiel von vorhin vorstellt, geht das mit Assembler ziemlich flott.

So flott, dass man das problemlos so schnell erledigen kann, dass ein flüssiger Eindruck entsteht.

Und BÄM! – wir haben einen 2D Scroller.



Mein missglückter Versuch eines Parallax Scrollers mit 3 Ebenen: Der Vordergrund scrollt mit 2 Pixeln, die Berge mit einem und die Wolken sollen eigentlich nur jedes 2. Frame ein Pixel scrollen. Hier sieht man sehr gut, dass das Umkopieren der Zeichen nach dem Hardware Scrolling zeitlich nicht richtig funktioniert. Wirkt leider auch durch die gif Kompression noch etwas Schlimmer als im Original.

Viele Action-Spiele auf dem C64 sind tatsächlich nichts weiter als „Texte“, die auf dem

Bildschirm hin und her bewegt werden. Bedient man sich jetzt noch ein paar weiterer Tricks, können schöne Effekte erzielt werden, indem bestimmte Bereiche auf dem Bildschirm in unterschiedlichen Geschwindigkeiten gescrollt werden – der sogenannte Parallax-Effekt.

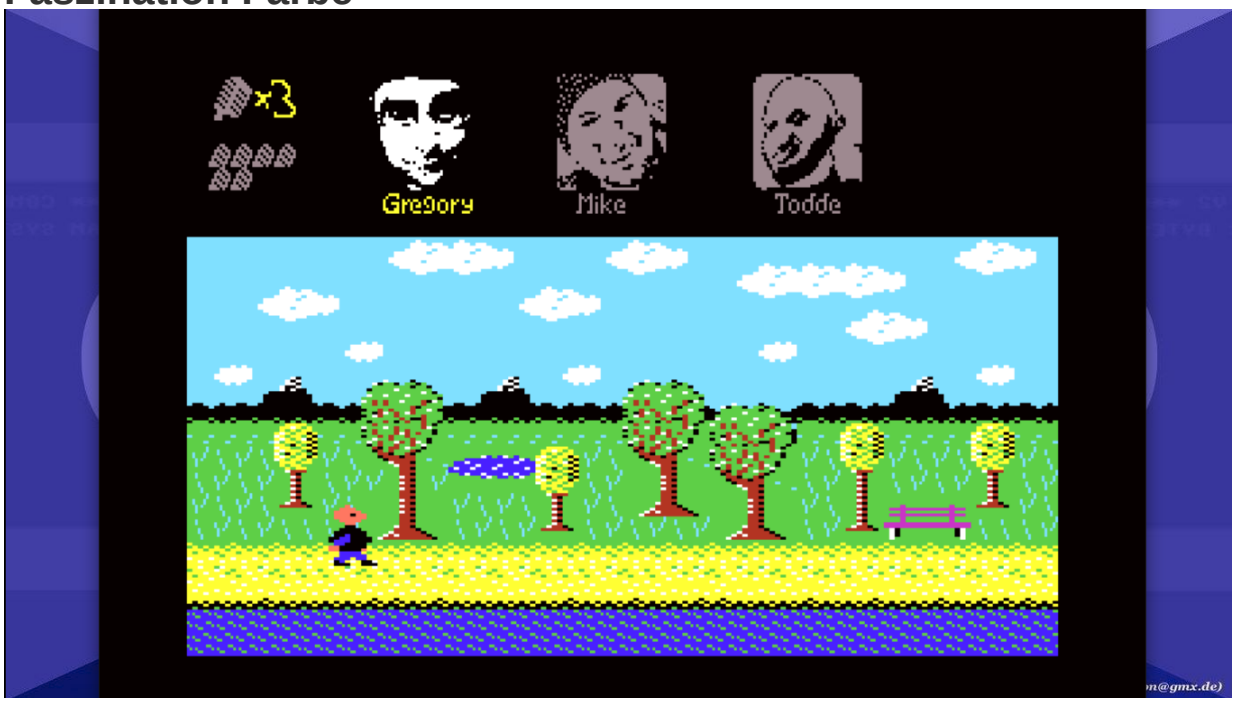
Zeilen, die den Bildteil darstellen, der weiter weg ist werden dabei langsamer gescrollt als die Teile im Bildvordergrund. Wie im echten Leben im Bahn oder im Auto zu beobachten ist.

Am Amiga gab es mal ein Spiel, das für diesen Effekt besonders viele einzelne Zeilen im Bild verwendete und dadurch einen tollen 3D Effekt erzielte: Elfmania.

Aber auch am C64 sind mittlerweile unglaubliche Effekte möglich. Wer davon etwas sehen will, sollte einfach mal auf Twitter nach c64 parallax suchen. Es gibt sehr viele Homebrew Coder, die die alte Hardware ausreizen, wie es sich die Entwickler niemals gedacht hätten. Ganz zu schweigen natürlich von Demos, die auf Youtube bewundert werden können. Aber das ist ein anderes Thema.

Hier noch ein gutes Beispiel in Blogform.

Faszination Farbe



Die aktuellste Version des Scrollers. Hier habe ich im oberen Bildbereich eine hohe Auflösung mit einem gesonderten Zeichensatz ausprobiert. Im unteren Teil wird das Bild von einem weiteren Zeichensatz mit multicolor Low – Res aufgebaut und das Spiel dargestellt. Irgendwann kommen Micha mit seinem Bike und Todde dazu, suchen Mikrophone und lösen zusammen knifflige Aufgaben. (Bild: Gerrit Ludwig)

Tja, mit diesem Wissen habe ich mir tatsächlich einen kleinen Teil der früheren Magie genommen. Ich kann natürlich nicht behaupten, dass ich jetzt die Frau in rot nicht mehr sehe, und statt dessen nur noch den Code der Matrix ?

Aber ich denke jetzt viel mehr über die Spiele nach, die ich sehe. Am Einfachsten geht das natürlich bei diesen alten 8 Bittern, die mit den sogenannten Tiles („Fliesen“,

„Kacheln“) arbeiten. Das ist im Prinzip nur ein anderer Name für die besprochenen Buchstaben-Zeichen.

Auch ohne dieses Wissen haben viele sicherlich schon bemerkt, dass diese Wolken bei Super Mario (natürlich in 8 und 16 Bit) irgendwie alle gleich aussehen. Bei den Bodenelementen ist es noch offensichtlicher. Und wer aufpasst, kann auch mal darauf achten, dass die Büsche auch nur der obere Teil einer Wolke in einer anderen Farbe sind.

Die Bilder werden aus ein paar „Zeichen“ zusammengesetzt und ergeben ein Großes Ganzes.

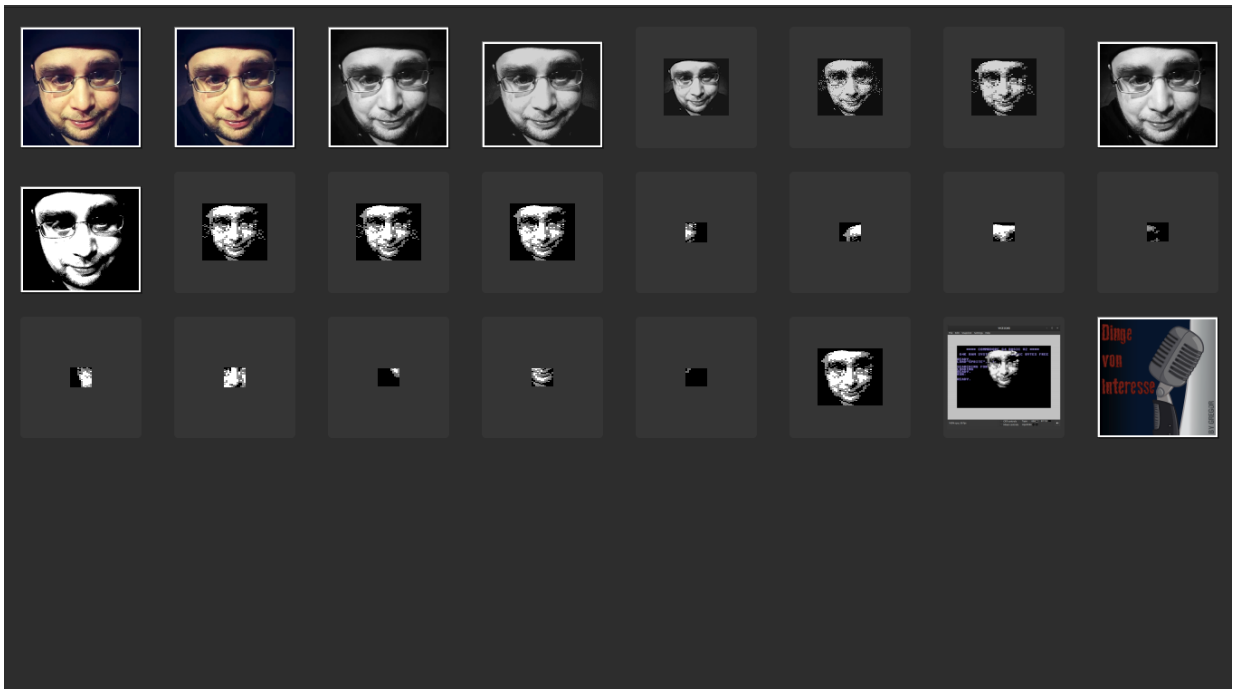
Aber deswegen sind sie lange nicht weniger schön.

Als kleinen Nachtrag möchte ich noch ein paar Gedanken zur unterschiedlichen Farbdarstellung anhängen.

Hier kann ich mich nur wiederholen, „wie die meisten schon wissen werden“ … war es bei den alten 8- und 16-Bit Computern gängig, dass sie entweder viele Farben in niedriger Auflösung, oder immer weniger Farben in relativ hoher Auflösung darstellen konnten. Wobei sich „relativ“ natürlich nur auf unsere heutige Sicht bezieht.

Die frühen Videospiele – und auch noch Homecomputer – waren ja primär für die Verwendung am Fernseher gedacht. Und es war selbst in den auslaufenden 70er und beginnenden 80er Jahren noch relativ teuer, ein Farbgerät zu besitzen. Geschweige denn mehr als eines!

Also wurden die Geräte so entwickelt, dass sie sowohl in schwarz/weiß, als auch in Farbe bedienbar waren. Es gab sogar Telespiele, die bunte Folien für den schwarz/weiß Fernseher hatten, um das Bild etwas farbiger gestalten zu können. Es könnte aber auch sein, dass diese Geräte einfach noch keine Farbe darstellen konnten, da bin ich mir jetzt auch nicht mehr so sicher.



Manchmal braucht man gar kein „Master Control Program“. Es reicht ein Gerrit, um @Onkel8028 (Gregor) auf mein Spiel-Grid zu beamen. (Bild: Gerrit Ludwig)

Der gemeine amerikanische NTSC Fernseher hatte in den 80er Jahren grob gesehen (bin kein Fernsehtechniker) etwa 480 Zeilen, lief dafür aber etwas flotter mit etwa 60Hz, der deutsche PAL TV bis maximal 575 Zeilen, leider nur mit 50 Hz. Dadurch wurden die Spiele im PAL Raum langsamer und der Ton tiefer. Wer mit dem PAL Fernsehen aufgewachsen ist und irgendwann mal die ersten BluRays seiner Lieblingsfilme in 24p gesehen hat, weiß vielleicht, woran ich denke, auch wenn das nicht direkt das selbe ist.

Beim Amiga gab es sehr oft Spiele, die sogar nur das NTSC Format hatten und dementsprechend einen riesigen Balken am unteren Bild zeigten.

Wer die Daten des C64 im Kopf hat, weiß, er kann 320*200 Bildpunkte darstellen.

Das ist allerdings so nicht ganz richtig, denn das bezieht sich auf den inneren Teil des Bildes. Außen rum ist ein aus heutiger Sicht (gesehen auf einem großen, hochauflösendem PC Monitor im Emulator) riesiger Rahmen, der zunächst zu nichts nutze erscheint.

Dies war früher eine Art „Geschenk“ der Ingenieure: Die Bildröhren waren nicht alle gleich in den Fernsehern verbaut. Manche zeigten mehr Bild, manche weniger, manche waren nicht mittig, usw.

Deshalb hat der wichtige Teil des C64 Bildes einen großen Spielraum außen rum bekommen, auf dem nichts passiert. Der Vorteil eines analogen Monitor war es z.B., dass man die Bildbreite- und Höhe einstellen konnte, so dass ich z.B. bei meinem Wiedereinstieg in den C64 fast schon geschockt war. Ich hatte den Rahmen auf meinem damaligen Monitor offenbar recht klein bekommen und hatte mein Bild somit viel größer in Erinnerung.

Bildröhren haben die älteren unter uns noch im Physikunterricht kennen gelernt. Ob das heute noch Programm ist, weiß ich nicht.

Aus der Sicht von Programmierenden fängt leider auch der Nullpunkt für beispielsweise Sprites in der linken oberen Ecke des Bildes an, also IM und natürlich unter dem Rahmen. Dieser ist ganze 50 Pixel hoch und 24 Pixel breit.

Könnten wir diesen Bereich frei nutzen, hätten wir schon eine Auflösung von (320+2*24)*(200+2*50) = 368 * 300 (!).

Im Normalfall geht das nicht, allerdings gibt es auch hier einige moderne Spiele und Anwendungs-Programme, die mit Tricks zumindest Teile des Rahmen ein bisschen nutzen können.

Tja, und dann ist ja da noch das Thema Farbe.

Da ich so lange mit dem C64 innig verbunden bin, hatte ich das einfach akzeptiert. Dass es aber auch ganz anders geht, habe ich mir erst kürzlich so richtig klar gemacht.

Wie ich schon angedeutet habe, ist der 6502 Prozessor ja zu dieser Zeit in vielen Computern eingesetzt worden. Er soll teilweise sogar heute noch in ähnlicher Form produziert werden.

Was den Charakter der Geräte im Endeffekt so richtig prägt, hängt allerdings auch ganz eng an den verwendeten Grafik- und Soundchips.

Beim C64 ist es der berühmte „Video Interface Controller“ getaufte VIC-II. Der Vorgänger hatte seine Einführung bereits im VC-20 bekommen, der Nachfolger VIC-III wurde schon für den nicht erschienenen C65 designt. Vollendet wird dieses Unterfangen heute mit extrem enthusiastischen Entwicklern, die vor Jahren anfingen, alte Prototypen des C65 aufzukaufen und nach zubauen.

Sie arbeiten jetzt am sog. Mega-65, der mit einem VIC-IV getauftem Grafikchip arbeitet und eine möglichst 100%ige Abwärtskompatibilität zu C65 und C64 aufweisen soll.

Sowohl für Sprites auch für die farbige Darstellung von Charakteren (also Buchstaben) hat man sich darauf geeinigt, eine mögliche Farbe direkt „im Pixel“ zu kodieren.

Dafür werden allerdings 2 Bits benötigt, denn wir wissen ja: 1 Bit hat nur 2 Zustände, 2 Bits können allerdings 4 Zustände abbilden: 0 bis 3.

Daher kommt die Einschränkung auf 4 mögliche Farben (Inkl. Transparenz-Farbe) pro Buchstabe bzw. „4*8“ Pixeln – also 4 doppelte Pixel in der Breite.

Um es noch einen Ticken komplizierter zu machen hat man – wie üblich ausgehend vom Normalfall – für den gesamten Bildschirm 2 Farben, 1 Transparenzfarbe und für jeden Buchstaben eine weitere Farbe. Wie auch übrigens im Hi-Res Modus für jeden Buchstaben eine andere Farbe möglich ist.

Diese Halbierung der Auflösung in der Breite auf 160 „Doppelpixel“ hat den Vorteil, dass nicht mehr Speicher benötigt wird als im Hi-Res Modus.

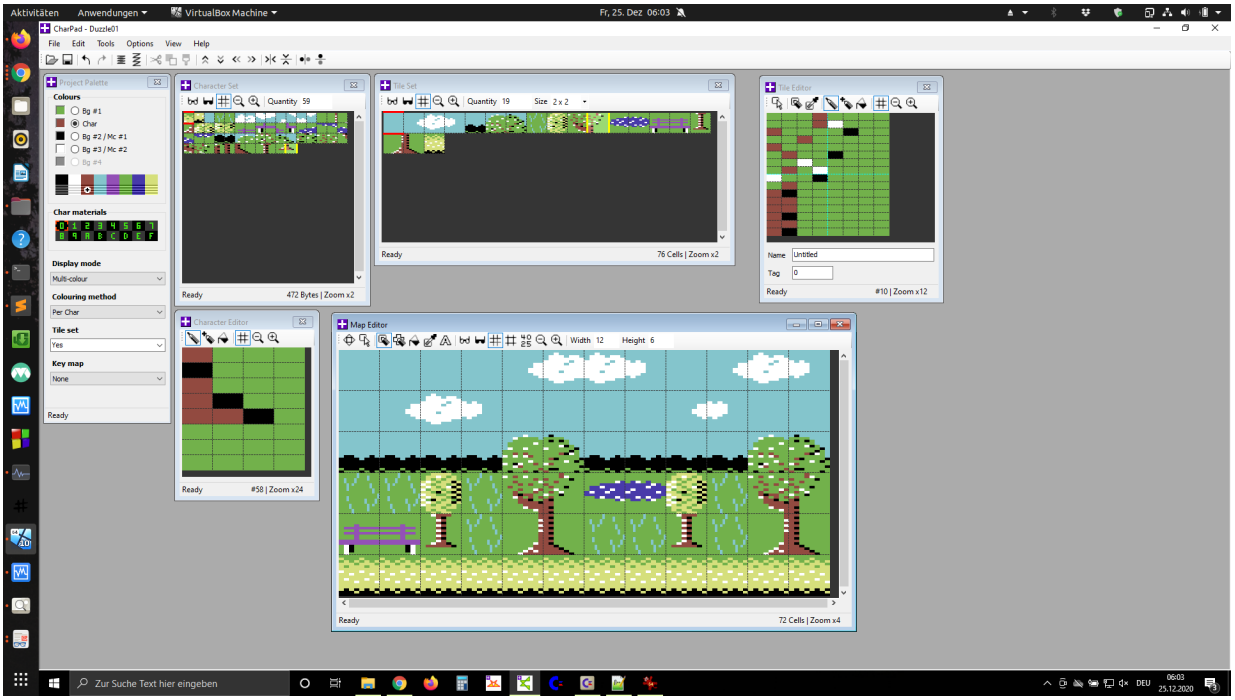
Eine besondere Einschränkung, die mir früher auch nicht unbedingt so negativ aufgefallen ist wie heute, ist dabei folgendes:

Wir haben festgestellt, dass sehr viele scrollende Spiele im Textmodus arbeiten, da dieser das Hardware-Scrolling unterstützt und wenig Speicher braucht.

Weiterhin haben wir gesehen, dass wir nur 4 Farben pro Zeichen benutzen können.

Eine Besonderheit des Multicolor (mc) Textmodus ist es, dass die Zeichen sowohl in mc, als auch einfarbig, aber dafür eben hochauflösend dargestellt werden können.

Das gemeine daran ist, dass der Umschalter dafür in Bit Nr. 4 sitzt und die Bits Nr. 3 bis 1 die Farbe enthalten. 3 Bits können aber nur 8 Zustände haben, also stehen, wenn der MC Textmodus für einen ganzen Bildschirm aktiviert ist, für jeden Modus nur 8 Farben zur Verfügung.



Meine ersten Tiles mit Charpad. Hier bestehend aus jeweils 2*2 Charakteren. (Bild: Gerrit Ludwig)

Das ist ein wichtiger Punkt wenn man unbedarft anfängt und mit fantastischen Tools wie zum Beispiel Charpad einen Hintergrund für ein Spiel designen will. Es gibt nur 8 Farben (!).

Leider fällt mir das jetzt auch bei vielen alten Spielen aus den frühen 80er Jahren auf, die ich als Kind sehr gerne gespielt habe. Die haben oft tatsächlich nur 4 Farben auf dem ganzen Bildschirm.

Das es auch anders geht, zeigt der Grafikchip des NES. Das NES hat quasi fast den gleichen Hauptprozessor wie der C64, der etwas schneller getaktet ist und dem – ähnlich wie beim Prozessor des Atari 2600 – in paar weniger benötigte Funktionen fehlen.

Der Grafikchip des NES wird PPU genannt, für „Picture Processing Unit“.

Der Vorteil dabei ist, dass das NES in genau der gleichen Assembler Sprache programmiert werden kann wie eben die anderen Geräte. Nur die Behandlung von Grafik und Sound ist anders.

Die PPU des NES hat eine feste Grafik von 32 * 30 Tiles, die auch hier wieder mit 8 * 8 Pixeln auflösen. Also 256 * 240 Pixel. Mit fest meine ich, es gibt gar keinen freien Bitmap Modus, es wird immer mit Tiles gearbeitet.

Es kommt allerdings ohne Rahmen aus und hat „echte quadratische“ Pixel – in mehr Farben.

Irgendwie zumindest.

Es hat insgesamt 54 verschiedene Farben (der C64 ja nur 16). Um diese verwenden zu können, müssen 8 Paletten angelegt werden – 4 für den Hintergrund und 4 für Sprites.

Allerdings benötigt jede Palette die Transparenzfarbe, so dass nur 3 Farben übrig bleiben – ähnlich wie beim C64. Jedes 8 * 8 Pixel – Tile kann eine der 4 Paletten haben.

Wir kommen also auf 3 * 4 = 12 Farben für den Hintergrund und nochmals 12 Farben für die Sprites – jetzt mal ohne die Transparenz mit zu zählen

Ein Sprite besteht auch nur aus 8 * 8 Pixeln, es dürfen aber insgesamt 64 am Bildschirm sein, davon jedoch nur wiederum 8 in einer Zeile. Puuh, ganz schön verwirrend.

Das erklärt allerdings, warum man in manchen NES Spielen besonders bei Schüssen die Grafik manchmal flackern sieht – sie besteht dort aus Sprites, von denen ein paar verschwinden, wieder auftauchen und dann wieder verschwinden …

Jede Farbe in einem Tile bekommt nun einen Wert, ähnlich wie am C64. Dieser Wert wird wiederum auf die Palette gemappt.

Also Pixel Nr 1 bekommt die Farbe 3, Pixel 2 die Farbe 5, usw.

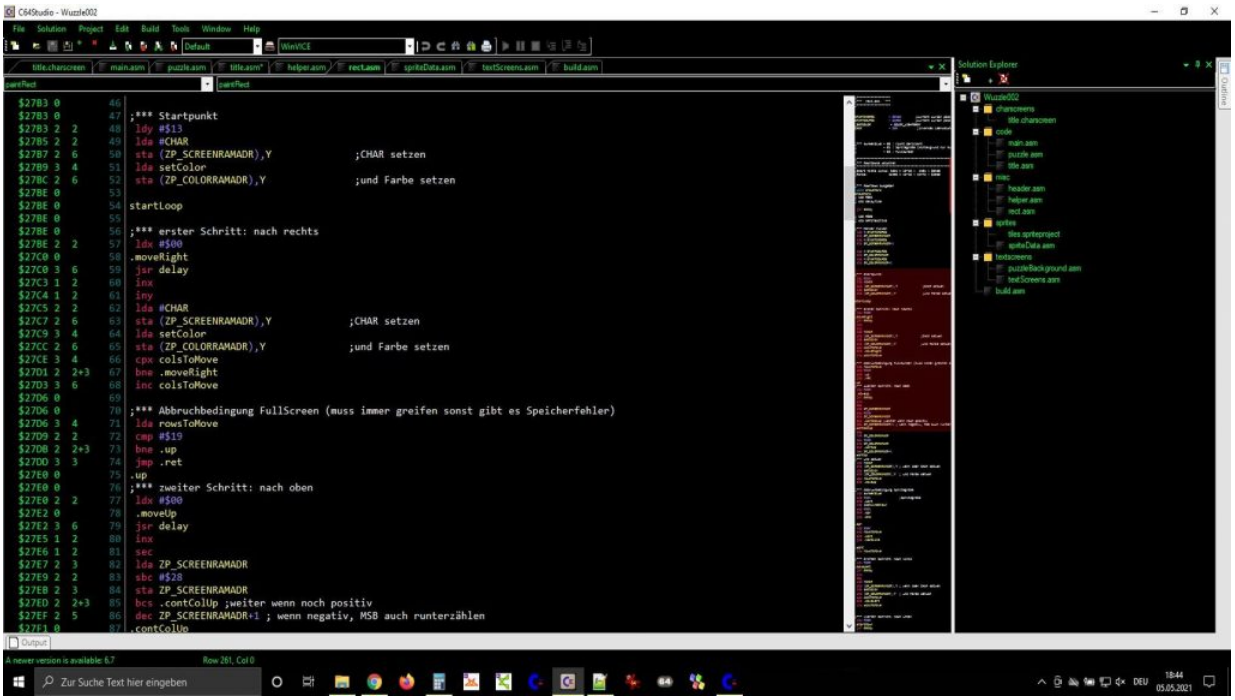
In der Palette steht dann 3 ist gelb, 5 ist blau, etc.

So benötigt man für jeden Bildschirm eine Tabelle mit den Informationen, welches Tile an welcher Position steht, und welche Palette es benutzt. Beim NES wird das „Nametable“ genannt, die Informationen sind wie üblich in Hex kodiert.

Und jetzt kommt doch noch ein bisschen Magie.

Während beim C64 jede Speicherzelle einzeln angesprochen und u. U. verändert wird, um einen Bildschirm zu füllen, wird bei der Programmierung der PPU nur diese nametable (natürlich auch Zahl für Zahl) und die Paletten in jeweils eine einzige Speicherstelle kopiert und das ganze Bild ist da.

Da ich beim NES noch nicht viel weiter bin, kann ich auch noch nicht sagen, wie hier einzelne Tiles modifiziert werden. Aber dass es geht, und uns trotzdem verzaubert, sehen wir ja an den tollen Spielen.



6502 Assembler Code im C64 Studio. (Bild: Gerrit Ludwig)

Ende

So, ich hoffe, ich konnte einen kleinen Einblick geben, warum mich die einfache 8 Bit-Assemblerprogrammierung gerade für den C64 so sehr fasziniert. Falls jemand mit gemacht hat, freut mich das umso mehr, und falls die Downloadzahlen meines kleinen Puzzlespieles etwas nach oben gehen noch mehr.

Ich veröffentliche auf Twitter hin und wieder Screenshots und Animationen meines aktuellen Spieleprojekts unter dem vorläufigen Hashtag #DuzzleWuzzle (Für das erste Spiel unter #PuzzleWuzzle). Dort findet sich alles wieder, was ich hier beschrieben habe. Einen richtigen Namen habe ich zwar schon im Sinn, aber um diesen öffentlich präsentieren zu können, muss ich mir erst ein bisschen mehr Mal-Geschick anlernen.

Hinweis: das wunderschöne Titelbild von Giana und Maria wurde freundlicherweise von dem großartigen PixelPoldi von Pixelbeschallung.at gestaltet – vielen Dank dafür!

Und außerdem tausend Dank an @Onkel8028 (Gregor), @Elraider31 (Micha) und Todde @ToddesNerdcast. Lasst euch nicht vom Grue fressen, wir freuen uns auf jeden eurer neuen Podcasts!

Dieser Beitrag wurde publiziert am Donnerstag, dem 1. Juli 2021 um 09:45 Uhr in der Kategorie: Videospielgeschichten. Kommentare können über den Kommentar (RSS) Feed verfolgt werden. Du kannst zum Ende springen und ein Kommentar abgeben. Pingen ist momentan nicht erlaubt.

Über Videospielgeschichten

Videospielgeschichten ist eine offene Plattform für Hobbyautoren und Journalisten. Die Webseite wurde 2009 gegründet, um es jedem Menschen, unabhängig von seiner Profession, zu ermöglichen, persönlich, authentisch und unabhängig über Videospiele zu schreiben

https://www.videospielgeschichten.de