(Teil 1)

Die meisten Leser haben auf den Mitmach-Karten den Wunsch nach einem Maschinensprachkursus geäußert. Voilà, hier ist er. Dieser Kurs kann kein Buch über Maschinensprache ersetzen, er wird Ihnen jedoch helfen, diese Sprache wesentlich leichter zu verstehen. Sowohl der 6502-Prozessor im VC 20 als

Assembler ist keine Alchimie

auch der 6510-Prozessor im C 64 werden behandelt.

Vermutlich hat es Ihnen auch schon ab und zu in den Fingern gejuckt, wenn Sie von Wunderdingen gelesen haben, die man per Maschinensprache mit dem Computer machen kann. Vielleicht haben Sie sogar schon mal nichtsahnend angefangen einzutippen, was Sie als Assemblerlisting sahen. Doch schon nach dem ersten »C000 LDA # $00« und RETURN weigerte sich der Computer mit einem lapidaren »SYNTAX ERROR«. Wieso, werden Sie sich gefragt haben, das ist doch nun die Sprache unserer Maschine, nämlich Maschinensprache, was habe ich falsch gemacht?

Dann sind Sie sicherlich mal auf diese merkwürdigen Basic-Programme gestoßen, in denen ein langer Wurm von DATA-Zeilen mit einem kleinen FOR..NEXT..POKE-Kopf vorne und einem SYS-Schwanz hinten enthalten ist, und die man Basic-Lader nennt. Sie haben fleißig Zahlen eingetippt — das ganze hoffentlich sofort abgespeichert-, vorschriftsmäßig mit dem SYS-Befehl gestartet und auf einen scheintoten Computer geschaut, der nur noch durch Aus- und Einschalten wiederzubeleben war. Wenn Sie dann nach langer Fehlersuche den irrtümlich eingetippten Punkt durch ein Komma ersetzen, (oft finden Sie auch keinen Fehler, denn bei langen DATA-Sequenzen schlägt der Druckfehlerteufel mit Vorliebe zu), werden Sie sich gefragt haben, warum in aller Welt dieses kleine Mißgeschick den ganzen Computer abstürzen läßt. Sie merken vermutlich schon, daß mir das alles und noch mehr (worüber ich schamhaft schweige) passiert ist. Die Konsequenz war, daß ich los ging, um ein schlaues Buch zu erwerben. Aber merkwürdig, damals tauchte der Begriff »Maschinensprache« in keinem Titel auf. Irgendwann begriff ich, daß Assembler und Maschinensprache irgend etwas miteinander zu tun haben.

Aber da fing das ganze Elend erst richtig an: Da gab es 6502-, Z80-, 8080-, 8085-, 6800-Assembler, da waren irgendwelche Schaltpläne, anscheinend, wie man wo was hinlötet- für mich als Nichtelektroniker eine Art moderner Kunst-, da war von CPU, Bussen, negativen Flanken, Zweiphasentakten die Rede.

Ich habe mich furchtbar geärgert über die Geheimsprache, die es dem Uneingeweihten verwehrt, etwas zu verstehen. Seither hat sich einiges verändert. Die Geheimnisse sind keine mehr und ich werde Ihnen in dieser Serie ohne verschlüsselte Sprache die magischen Zirkel der Assembler-Alchimisten offenbaren. Heute gibt es auch Bücher über »Maschinensprache auf dem Commodore 64« und es sei Ihnen angeraten, ruhig auch das eine oder andere durchzuarbeiten. Sie werden allerdings feststellen, daß die meisten davon gerade dort aufhören, wo es anfängt, spannend zu werden: Bei der Benutzung von Routinen des Betriebssystems und des Interpreters. Deswegen soll der Schwerpunkt dieser Serie anders liegen:

Wir werden das notwendige Grundwissen über die Hardware nur ganz knapp behandeln, dann das Vokabular des 65xx-Assembler kennenlernen. Den Hauptteil der Serie verbringen wir aber mit Dingen, über die es kaum Literatur gibt, nämlich wie man für eine Unzahl von Programmieraufgaben nicht nochmal das Rad erfinden muß, weil es schon längst in unserem Computer existiert.

Bevor wir loslegen, will ich Ihnen noch etwas Literatur empfehlen:
a) Wenn wir über Speicheraufbau, das binäre und das hexadezimale Zahlensystem reden, werde ich voraussetzen, daß Sie die Serie »Reise durch das Wunderland der Grafik« gelesen haben, die in dieser Zeitschrift in den Folgen 1 und 2 (Ausgaben 4/84 und 5/84) diese Themen grundlegend behandelt hat.
b) Als Nachschlagebuch sehr wertvoll ist das Buch von Rodney Zaks: Programmierung des 6502. Das ist gewissermaßen ein Klassiker.
c) Später wird Ihnen dieses Buch fast unentbehrlich vorkommen: R. Babel, M. Krause, A. Dripke: Systemhandbuch zum Commodore 64 (und VC 20), München 1983

Weitere Literaturempfehlungen werde ich Ihnen von Fall zu Fall geben. Gerade zu unserem Computer erscheint fast jeden Monat ein neues Buch und es ist nicht einfach, die Spreu vom Weizen zu trennen.

Einige Begriffsklärungen

Zunächst einmal muß ich Sie enttäuschen: Ich glaube kaum, daß Sie mit Ihrem Computer je einmal in Maschinensprache verkehren werden! Maschinensprache, das ist die einzige, die der Computer direkt versteht, das sind vorhandene oder nicht vorhandene Stromimpulse oder Magnetisierungszustände, die bei unserem Computer durch 8-Bit-Binärzahlen auszudrücken sind. Was wir mit unserem Computer reden werden ist Assembler. Mit dem Computer sprechen soll heißen: Mit dem Gehirn unseres Computers, dem Prozessor, oft auch CPU (von Central Processing Unit = Zentraler Arbeitsbaustein) genannt, verkehren, also ihm Befehle zu geben. Solche CPUs werden bei verschiedenen Firmen hergestellt, sind daher unterschiedlich aufgebaut und auch unterschiedlich ansprechbar. Ein weit verbreiteter Prozessortyp ist der 6502, der das Gehirn des C 64 und auch des VC 20 ist. Genau genommen ist das Gehirn des C 64 allerdings der 6510, ein dem 6502 fast identischer Prozessor. Auf den kleinen Unterschied werden wir noch zu sprechen kommen. Beide (6502 und 6510) sind in 6502-Assembler zu programmieren und wenn wir diese Sprache sprechen, sind für uns alle 6502-Computer zugänglich: Commodore, Apple, Atari und einige an-

dere. Nun wissen Sie aber immer noch nicht, was Assembler eigentlich ist. Das englische Wort »assemble« heißt auf deutsch etwa montieren, zusammenstellen. Es handelt sich also um eine Programmiersprache und weil sie sehr eng am Computer orientiert ist, spricht man von einer »maschinenorientierten« Programmsprache im Gegensatz zu »problemorientierten« Programmsprachen wie Basic, Pascal, Cobol etc., die — so sollte es jedenfalls sein — auf jedem Computertyp gleich aussehen.

Ein Assembler ist aber noch etwas anderes, nämlich ein Software-Instrument, das einen in Assembler geschriebenen Befehl in die Maschinensprache übersetzt. Man spricht vom Vorgang des Assemblierens. Das umgekehrte leistet ein Disassembler, welcher uns Maschinensprache durch Rückübersetzung lesen hilft. Um die Verwirrung noch etwas zu steigern, sage ich Ihnen auch noch, was ein Monitor ist. In diesem Zusammenhang ist kein Bildschirmgerät damit gemeint, sondern ebenfalls ein Software-Instrument, das den Einblick in die Register und Speicher des Computers gewährt.

Damit Sie nun den Überblick noch völlig verlieren, sei abschließend zu diesem Sprachenwirrwarr noch erzählt, daß Software-Pakete, die sowohl Assembler als auch Disassembler als auch Monitor enthalten und noch eine Menge anderer brauchbarer Dinge, oft als »Assembler« angeboten werden. Das ist ein alter Trick der Alchimisten, verschiedenen Dingen den gleichen Namen zu geben!

Basic contra Assembler

Um das Nachfolgende deutlich zu machen, schalten Sie bitte Ihren Computer an und tippen die beiden folgenden Programme ein, die beide genau dasselbe tun: Das obere Viertel unseres Bildschirmes mit dem Buchstaben A zu füllen (beim VC 20 ist es die obere Hälfte). Zunächst einmal in Basic:

```
10 FOR I=1024+255 TO 1024 STEP-1
20 POKE I, 1:POKE I+54272,14
30 NEXT I
```

Für den VC 20 (Grundversion und 3-KByte-Erweiterung) ist zu setzen: Anstelle von 1024 jetzt 7680, statt 54272 jetzt 30208 und statt 14 die 6. Wenn Sie mehr als die 6,5 KByte im VC 20 haben, dann setzen Sie statt 1024 jetzt 4096, statt 54272 jetzt 34304 und ebenfalls statt 14 die 6. Das Programm braucht 55 Byte + 7 Byte für die Variable I, macht zusammen 62 Byte Speicherplatz. Es geht ganz schnell und wenn Sie es schaffen, können Sie ja mal mitstoppen, wie lange es von RUN bis READY braucht: Zirka 4 Sekunden.

Jetzt dasselbe in Assembler. Weil wir aber noch nicht soweit sind, erst mal als Basiclader, der uns das Programm in den Speicher bringt (wir kommen dazu gleich noch). Geben Sie also NEW ein und dann:

```
10 FOR I=7000 TO 7000+16
20 READ A:POKE I,A:NEXT I :END
30 DATA 160,255,162,14,169,1,153,255,
3,138,153,255,215,136,208,244,96
```

Beim VC 20 geben Sie bitte statt der 14 (Zeile 30,4.Zahl) eine 6 ein. Starten Sie den Basiclader mit RUN und nach dem READY geben Sie NEW und CLR ein: wir brauchen ihn den Basiclader nicht mehr. Ab Speicherstelle 7000 steht jetzt unser Assemblerprogramm als Maschinencode. Daß es wirklich dasselbe tut wie das Basicprogramm erfahren Sie durch SYS 7000. Da hatten Sie vermutlich gar keine Zeit mehr, auf die Stoppuhr zu drücken! (5,4 Millisekunden etwa dauert das ohne die Zeit, die der Basicinterpreter für den Befehl SYS benötigt). Außerdem braucht das Programm 17 Byte Speicherplatz.

Genau das ist es, was die Assemblerprogrammierung so reizvoll macht: Der Speicher faßt mehr an Programm und die Ausführung des Programmes geht fast 1000mal so schnell! Dazu kommen natürlich noch einige andere Kriterien, denn viele Probleme sind zum Beispiel in Basic nicht lösbar, sondern nur mit dem vielseitigeren Assembler.

Unser Computer ist darauf vorbereitet, daß wir ihn in Basic ansprechen. Er enthält im Normalfall sofort nach dem Einschalten ein stets präsentes Übersetzungsprogramm, den Interpreter, welcher unsere Basicanweisungen für ihn verständlich interpretiert. Auch das ist ein Unterschied zu Assemblerprogrammen: Ist ein solches Programm erst einmal assembliert (also als Maschinensprache im Speicher vorhanden), braucht man kein Übersetzungsprogramm mehr. Basicprogramme dagegen müssen bei jedem Durchlauf von vorne bis hinten ständig übersetzt werden, sie laufen nicht ohne vorhandenen Interpreter. Wie so ein Interpreter im Prinzip arbeitet und was ihn von einem sogenannten Compiler unterscheidet, sollten Sie mal in der April-Ausgabe der Zeitschrift 64'er im Artikel ab Seite 110 von M. Törk über seinen Strubs-Precompiler nachlesen.

Dort sehen Sie dann auch, daß ein Compiler zwar ein Basicprogramm enorm beschleunigen kann, aber bei weitem nicht an die Geschwindigkeit reiner Assemblerprogramme heranreicht, vom Speicherplatzbedarf ganz zu schweigen.

Wie sag ich's meinem Computer?

Leider haben weder der C 64 noch der VC 20 einen Assembler implementiert. (Sie merken, daß jetzt von dem Software-Paket die Rede ist!). Es gibt einen etwas mühseligen Weg, dieses Handicap zu umgehen: Den Basiclader. Wie ist also der Weg, mit einem solchen Lader eigene Maschinenprogramme in den Computer zu bekommen?

a) Erstellen des Assemblerprogrammes. Das zu lernen ist die Hauptaufgabe in der Serie. Das Ergebnis wird eine Kette von Befehlen sein, zu denen zum Beispiel der Befehl RTS gehört.

b) Jedem Befehl in Assembler entspricht in Maschinensprache ein Binärcode in einer Speicherstelle. Diese Codes sind in Listen nachschlagbar: RTS entspricht dem Binärcode 0110 0000.

c) Der Code muß in eine Speicherstelle eingegeben werden. Das geschieht von Basic aus mit dem POKE-Befehl. Weil aber Basic keine Binärzahlen kennt, muß der Code ins Dezimalsystem umgerechnet werden. Glücklicherweise sind in den Tabellen meist schon die Codes als Dezimal- oder wenigstens als Hexadezimalzahlen enthalten. RTS ist dezimal 96 (oder hex.60, das auch $ 60 geschrieben werden kann). Man POKEt nun an die richtige Adresse den Wert 96, also zum Beispiel POKE 7016,96

d) Auf diese Weise wird Byte für Byte in der Programmabfolge verfahren. Das reine POKEn geschieht dann eben in der Form wie im oben gezeigten Basiclader. Mühsam, mühsam! Auch kann man leider nur mit dem PEEK-Kommando nachsehen, was denn nun im Speicher steht (PEEK (7016) gibt uns den Wert 96, entsprechend RTS).

Ein anderer Weg ist der Kauf eines Assembler-Moduls oder einer entsprechenden Programm-Kassette oder Diskette. Sehr preiswert ist es sicherlich, in Fachzeitschriften Umschau zu halten nach abge-

druckten Assemblern. Sehr gut finde ich beispielsweise den von U. Roller in Chip Nummer 1 (1984), aber auch im 64'er wird während des Verlaufs dieser Serie ein gut verwendbares Programm vorgestellt.

Assembler (das Software-Paket) gibt es in den unterschiedlichsten Ausführungen. Es gibt beispielsweise Direkt-Assembler, die jede Programmzeile sofort nach dem RETURN assemblieren, aber auch 2-Pass-Assembler, bei denen das erst nach Abschluß des Programms insgesamt durch einen Befehl (zum Beispiel ASSEMBLE) geschieht. Bei einigen kann man (ähnlich wie bei Basic mit REM) Kommentare anfügen, bestimmten Programmstellen Namen geben (sogenannte LABELS), ganze Programmabschnittte mit einem Merknamen aufrufen (sogenannte MAKROS) und so weiter. Was Sie für sich bevorzugen, bleibt Ihnen natürlich überlassen. Die in dieser Serie geschriebenen Programme werden auf diese schönen Erleichterungen verzichten, es wird sozusagen der nackte Assembler verwendet und solange in dieser Zeitschrift noch kein Software-Paket dafür veröffentlicht ist, werde ich jedes Programm, das wir zusammen entwickeln, als Basiclader angeben. Was Sie aber außer dem reinen Assembler noch brauchen, ist ein Disassembler und ein Monitor (ich habe schon erklärt, welchen ich meine), damit wir unseren Computer (fast) immer im Griff haben.

Wie funktioniert unser Computer?

Weil das Programmieren in Assembler einen viel engeren Kontakt zu technischen Einzelheiten unseres Computers erfordert, ist es notwendig, ein wenig über diese Innereien und ihre Funktion zu wissen. Sehen Sie sich dazu bitte das Bild 1 an.

Da sehen wir zunächst unseren Mikroprozessor, der meist eine Menge Funktionen in sich vereinigt (dazu kommen wir noch). Im Prinzip ist das unsere CPU (Zentraler Arbeitsbaustein). Der Prozessor steht über eine Reihe von Leitungen mit dem Rest des Computers in Verbindung. Diese Leitungen werden im Fachjargon BUSSE genannt. Da ist zunächst einmal der sogenannte Adreßbus, auf dem 16-Bit-Adressen transportiert werden, die der Prozessor erzeugt, und die die Herkunft oder auch das Ziel von Daten festlegen, die über den Datenbus laufen. Dieser kann 8-Bit-Daten transportieren, und zwar schreibend oder lesend, also zum Beispiel vom Prozessor zum RAM (schreibend), vom RAM zum Prozessor (lesend) und so weiter. Außerdem gibt es da noch einen Steuerbus, der verschiedene Synchronisationsaufgaben durchführen hilft. Links vom Prozessor ist ein Taktgeber angedeutet. Damit nichts durcheinander kommt, läuft alles im Computer sozusagen im Gleichschritt. Diese Uhr ist gewissermaßen der Trommler, den Sie vielleicht von den alten Ruder-Galeeren kennen. Dann sehen Sie rechts einen ROM-Bereich, also einen Nur-Lese-Speicher (Read Only Memory). Daß man hier nur herauslesen kann, ist durch den Pfeil zum Datenbus gekennzeichnet. Doppelpfeile finden wir aber beim RAM (Random Access Memory), einem Speicher für beliebigen Zugriff, also lesend und schreibend, und bei den Ein- und Ausgabebausteinen, die den Kontakt des Computers mit der übrigen Welt erlauben, also auch mit uns. Dieses Aufbauprinzip finden wir bei allen 8-Bit-Computern.

Das Innenleben eines Mikroprozessors

Um es gleich nochmal zu sagen: Was hier erzählt wird, ist nicht dazu geeignet, Elektronik-Freaks den totalen Durchblick zu geben. Wenn Sie das aber gerne möchten, dann sehen Sie sich zum Beispiel die Blockschaltbilder an im »Programmer's Reference Guide« für den Commodore 64 auf Seite 404 oder im »MOS-Hardware-Handbuch« auf Seite 34. Auch Rodney Zaks in dem anfangs schon erwähnten Buch »Programmierung des 6502« ist zu empfehlen. Er hat sich viel Mühe gegeben, sich verständlich auszudrücken. Mir kommt es nur auf den allgemeinen Überblick an. Den sollen Sie bekommen, wenn wir uns jetzt zusammen Bild 2 betrachten.

Da sehen Sie zunächst als Herzstück des Prozessors, die ALU (Arithmetik Logical Unit), also den arithmetisch-logischen Baustein. Die ALU hat die Fähigkeit, Rechenoperationen auszuführen mit Daten, die sie über den Datenbus und normalerweise vom Akkumulator erhält. Das Ergebnis wird ebenfalls im Akkumulator abgelegt (daher auch der Name: von akkumulieren, etwa ansammeln). Der Akkumulator ist das Register, das uns als Programmierer am häufigsten beschäftigen wird. Er ist die Sammel- aber auch die Verteilerstelle für fast alle Daten, die wir hin- und herschieben wollen. Sowohl der Akku (so werde ich, mit der Hoffnung auf Ihr wohlwollendes Verständnis, künftig bezeichnen) als auch alle anderen Register, das heißt, die höchste Zahl, die darin bearbeitet werden kann, ist 255 (binär 1111 1111). Nahezu ebenso oft wie den Akku werden wir die beiden sogenannten Index-Register X und Y benutzen. Warum man sie Index-Register nennt, sehen Sie noch im Verlauf der Serie. Als nächstes zum Prozessor-Statusflaggen-Register (hier P genannt). Man findet hier angezeigt, ob eine Rechenoperation ein negatives Ergebnis hatte oder ob eine Null aufgetaucht ist oder ob ein Übertrag stattgefunden hat. Auch dieses Register wird uns noch häufig begegnen. Das Stapelregister, auch Stackpointer (Stapelzeiger) genannt, gibt uns Auskunft über den Füllungsgrad eines 256 Byte großen speziellen Speichers, der vom Prozessor direkt verwaltet wird. Auch damit werden wir noch oft zu tun haben. Schließlich kommen wir zur vorhin erwähnten Ausnahme, zum Programmzähler (PCL, PCH). Das ist ein 16-Bit-Register, das sich aus zwei 8-Bit-Registern (PCL für das LSB und PCH für das MSB) zusammensetzt und daher alle 65535 Speicherplätze ansprechen kann. Hier ist immer die Adresse des nächsten abzuarbeitenden Befehls enthalten.

Ich will an dieser Stelle nicht in die Einzelheiten der Befehlsabarbeitung einsteigen (das können Sie auch bei Rodney Zaks nachlesen, wenn Sie's genau wissen wollen). Es soll nur gesagt sein, daß sich die Verarbeitung in drei Schritte unterteilen läßt:

a) den nächsten Befehl holen
b) den Befehl decodieren
c) den Befehl ausführen

Zu c) ist noch zu sagen, daß es Befehle gibt, die der Prozessor ohne weitere Angaben ausführen kann. Für andere müssen erst noch weitere Daten aus dem Speicher geholt oder dort abgelegt werden. Deswegen brauchen die Befehle unterschiedliche Zeiten zur Ausführung. Die Zeit wird als Anzahl von sogenannten Taktzyklen in den Befehlstabellen angegeben. Unser Computer hat eine Taktfrequenz von rund 1 MHz, was bedeutet, daß ein Taktzyklus etwa eine Mikrosekunde (10^{-6} Sekunden) dauert. Auf diese Weise wurde die Zeitdauer für unser kleines Demonstrationsprogramm zu ►

Anfang berechnet. Auch das werden Sie noch lernen.

Der Speicher unseres Computers: Eine Straße mit 65536 Hausnummern

Diese Serie ist für den VC 20 und den C 64 geschrieben. Den Speicheraufbau des Commodore 64 finden Sie in der April-Ausgabe dieser Zeitschrift ab Seite 119. Deswegen soll hier nur der des VC 20 gezeigt werden. Man muß beim VC 20 zwei Konfigurationen unterscheiden — sehr zum Leidwesen der Benutzer. In Bild 3 ist die Aufteilung gezeigt, die in der Grund- und der um 3 KByte erweiterten Version vorliegt.

In Bild 4 sehen Sie die Speicheraufteilung, die bei mehr als 6,5 KByte eingestecktem Speicher gültig ist.

Wenn Sie die VC 20 Speicherar-

Tabelle 1. Basic-Start- und -Endadressen beim VC 20 mit verschiedenem Speicherausbau

Grundversion	:Basic-Start	4096	Basic-Ende	7679
+3-K-Erweiterung	: —"—	1024,	—"—	7679
+8-K-Erweiterung	: —"—	4608,	—"—	16383
+16-K-Erweiterung	: —"—	4608,	—"—	24575
+24-K-Erweiterung	: —"—	4608,	—"—	32767

Bild 1. Aufbauprinzip eines 8-Bit-Computers

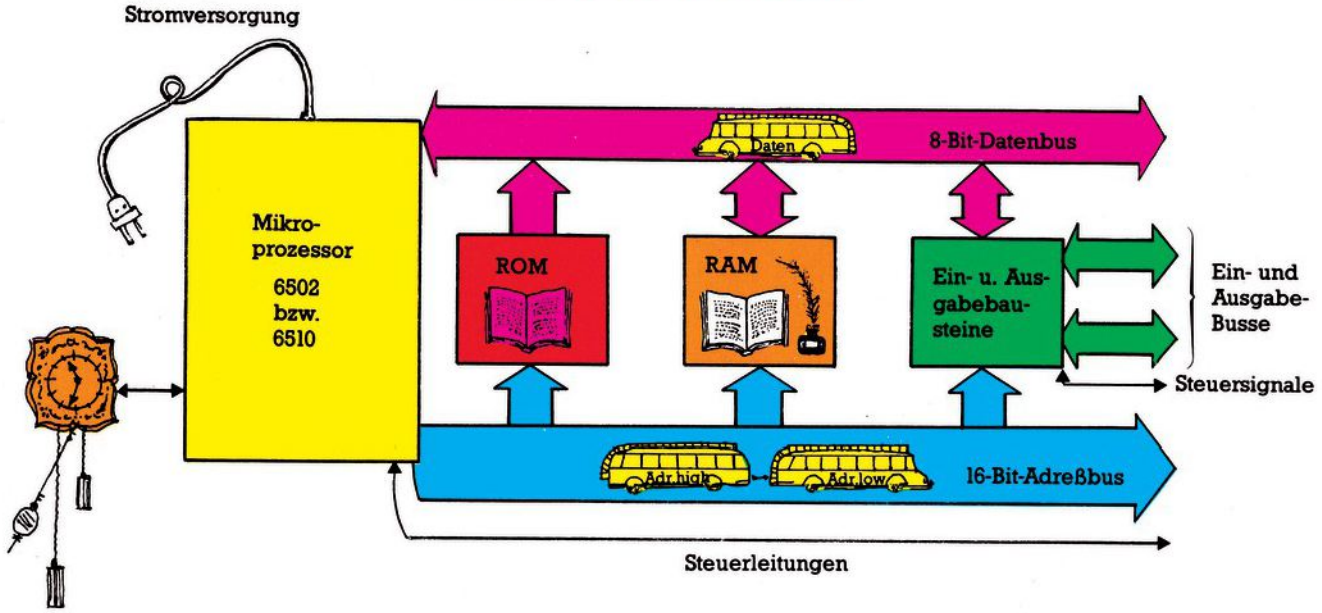


Bild 2. Aufbauschema eines 6510-Prozessors

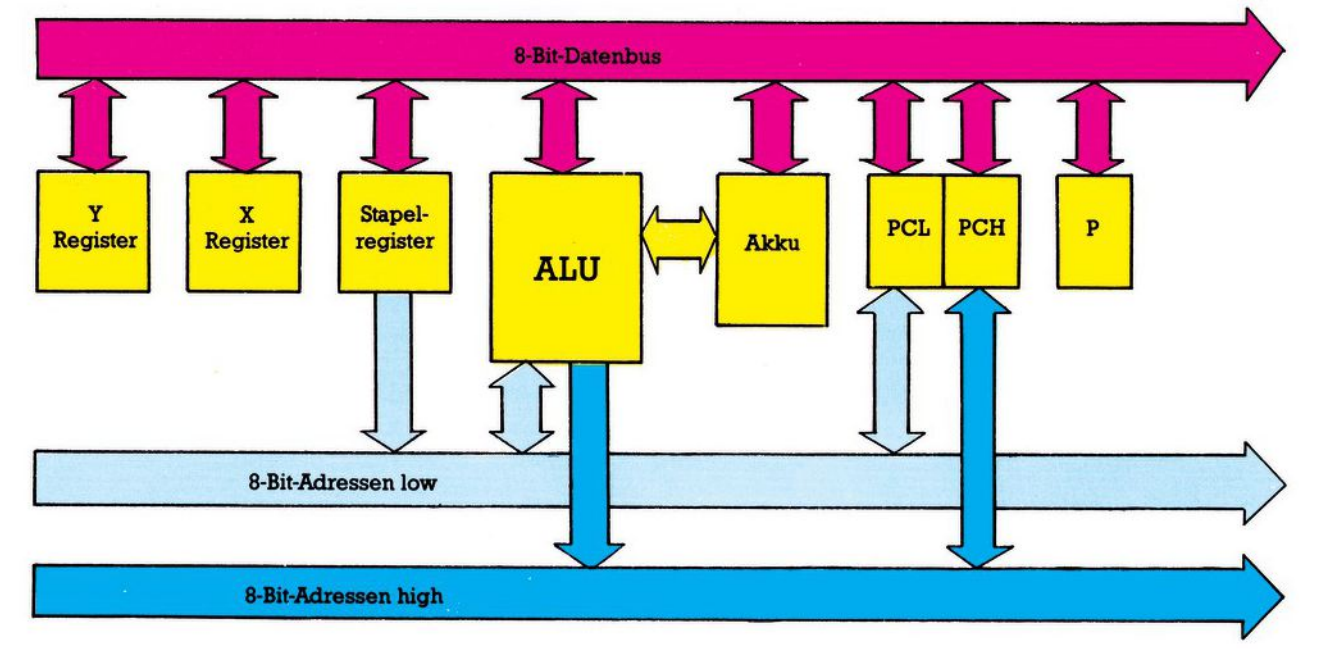


Bild 3. VC 20-Speicher (Grundversion oder mit 3-KByte-Erweiterung)

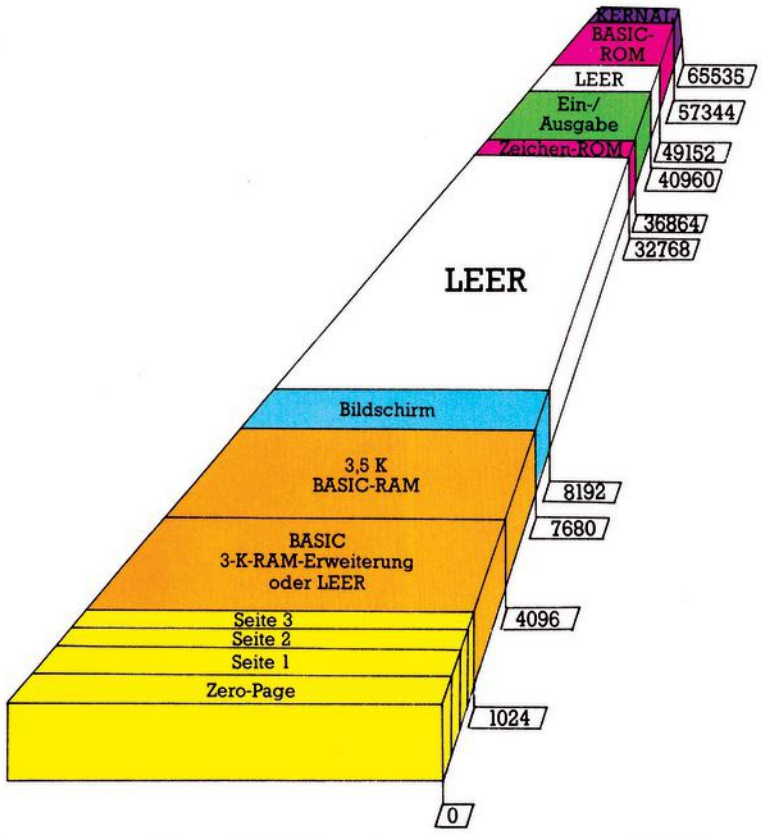
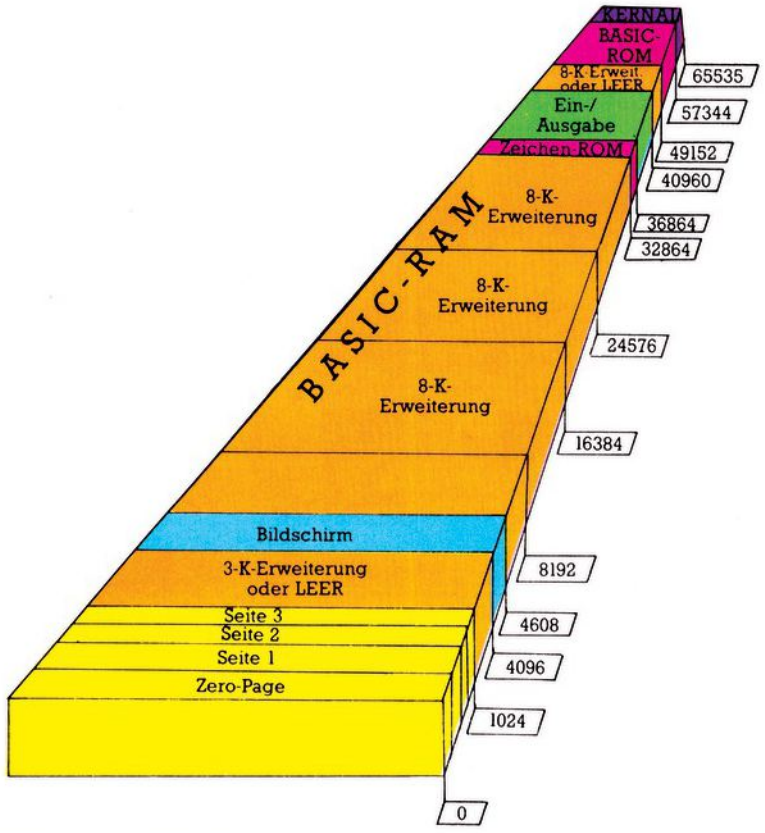


Bild 4. VC 20-Speicher (Version mit mehr als 6,5 KByte Speicherplatz)



chitekturen mit der des C 64 vergleichen, werden Sie eine Reihe von Unterschieden feststellen. Genau besehen gibt es an den wichtigen Punkten aber eine Menge Gemeinsamkeiten! Der VC 20 kennt nur Speicher-Häuser mit Erdgeschoß, im Gegensatz zum C 64, wo manche Bereiche sogar zwei Etagen haben (soll heißen: mehrfach belegt sind). Durch die Eigenart des C 64 aber, im Normalfall das Basic-ROM, die Ein- und Ausgabebausteine und das Betriebssystem eingeschaltet zu haben, kann man ihn eigentlich genauso behandeln wie einen VC 20, bei dem die genannten ROM-Bausteine, — und zwar das Basic-ROM —, um 8 KByte verschoben sind. Die Unterschiede der ROM-Inhalte sind fast vernachlässigbar. Wir werden im Einzelfall darauf zu sprechen kommen. Bei den Ein- und Ausgabebausteinen liegen allerdings größere Unterschiede.

Die Seiten 0 bis 3 (eine Seite oder auch page enthält 256 Byte und man zählt oft auch in diesen Seiten, wenn vom Speicher die Rede ist), sind sich ebenfalls sehr ähnlich und die wenigen Unterschiede werden uns ebenfalls noch beschäftigen. Der Bildschirm liegt bei der Grundversion und der mit der 3-KByte-Erweiterung von 7680 bis 8191, in der Version mit mehr als 6,5 KByte von 4096 bis 4607 und beim C 64 von 1024 bis 2047. Der Bildschirmfarbspeicher liegt — bei gleicher Reihenfolge — von 37888 bis 38399, beziehungsweise von 38400 bis 38911 und schließlich von 55296 bis 56295. Der Basic-RAM-Bereich beginnt beim C 64 im Normalfall bei 2048 und endet bei 40959. Beim VC 20 ist das natürlich wieder von der jeweiligen Erweiterung abhängig (Tabelle 1).

Dies gilt — wie Sie leicht auch aus Bild 4 sehen können — auch dann, wenn zu den 8 KByte/16 KByte/24-KByte-Erweiterungen noch die 3-KByte-Erweiterung und die 'KByte-Erweiterung im hohen Speicherbereich (40960 bis 49151) verwendet werden. Diese letztgenannten Adressenbereiche sind dann gut als geschützte RAM-Bereiche für Maschinensprache zu verwenden, ebenso wie beim C 64 der Speicherabschnitt von 49152 bis 53247.

Damit schließen wir zunächst mal den Hardware-Überblick ab. In der nächsten Folge stelle ich Ihnen dann die ersten Assembler-Befehle vor, und wir beginnen zu programmieren. Sie gehören nun zum 1. Grad der Assembler-Alchimisten.

(Heimo Ponnath/aa)