Inhaltsverzeichnis Seite 1

Danksagung / Hinweis		Seite 2
Teil 1 – Die Grundelemente erklärt	(Ausgabe 11/90)	Seite 3
1. Einleitung		Seite 3
2. Die Hardware		Seite 3
3. Was ist ein Interrupt		Seite 4
4. Der Timer-Interrupt		Seite 5
Das Interrupt-Control-Register (ICR)		Seite 9
Teil 2 – Interruptablauf erklärt	(Ausgabe 12/90)	Seite 11
1. Der Prozessor und ein Interrupt		Seite 11
Teil 3 – Programmierung der CIA mittels System-IRQ	(Ausgabe 01/91)	Seite 16
Teil 4 – NMI Interrupts erklärt	(Ausgabe 02/91)	Seite 24
Teil 5 – Timerprogrammierung und -kopplung	(Ausgabe 03/91)	Seite 31
Teil 6 – Programmierung der Echtzeituhren	(Ausgabe 04/91)	Seite 37
Teil 7 – Ein- und Ausgabeprogrammierung	(Ausgabe 05/91)	Seite 45
Teil 8 – Joystickabfrage	(Ausgabe 06/91)	Seite 48
Teil 9 – Programmierung einer „echten“ Mausabfrage	(Ausgabe 08/91)	Seite 52
Teil 10 – Mausabfrage über Userport	(Ausgabe 10/91)	Seite 58
Sourcecode-Listings		Seite 67
MSGOUT-ROM.SRC / MSGOUT-RAM.SRC	(01/91 – Teil 3)	Seite 67
EVAL.SRC	(03/91 – Teil 5)	Seite 68
CLOCK.SRC	(04/91 – Teil 6)	Seite 69
WAITSHIFT.SRC	(05/91 – Teil 7)	Seite 72
AMIGA-MAUS1.SRC	(08/91 – Teil 9)	Seite 73
AMIGA-MAUS2.SRC	(09/91 – Teil 10)	Seite 75
AMIGA-MAUS3.SRC	(09/91 – Teil 10)	Seite 77

Danksagung

Die Veröffentlichung der Magic Disk Kurse als PDF-Datei erfolgt mit schriftlicher Genehmigung der COMPUTEC MEDIA GmbH. Es dürfen keine kommerziellen Absichten verfolgt werden!

Mein Dank gilt ebenfalls Mirco Geldermann. Auf seiner Webseite www.magicdisk64.de ist es möglich die Diskettenimages der Magic Disk C64 Ausgaben offiziell herunterzuladen.

Hinweis

Ein komplettes ROM-Listing des C64 mit deutschen Kommentaren ist unter folgenden Link zu finden: http://www.pagetable.com/c64rom/c64rom_de.html.

Teil 1 – Magic Disk 11/90

1. Einleitung:
Herzlich willkommen zu unserer neuen Kurs-Serie. Nachdem Sie mein Kollege IVO HERZEG die letzten Monate ja eingehend in die Materie der Raster-Interrupts eingeführt hat, will ich Ihnen nun ein weiterführendes Thema anbieten: die CIA-Bausteine des C 64.
Diese steuern (wie im Raster-IRQ- Kurs schon angedeutet) die übrigen Interruptfunktionen unseres Rechners und sind für den Kontakt mit der Außenwelt des 64ers verantwortlich. Ohne sie könnten wir ihn in keinster Weise bedienen - die Bedienung der Tastatur und des Joysticks oder die Datenspeicherung auf Massenmedien wie Kassette oder Diskette wäre gar nicht möglich. Sie sehen also, daß in den nächsten Monaten einige ganz interessante Themen auf Sie zukommen werden.
Ich möchte mich zunächst einmal um die Interruptprogrammierung kümmern und in fortführenden Folgen Anleitungen zur Bedienung der Ein-/ Ausgabeeinheiten des 64ers geben. Wir werden dann den Joystick einmal genauer unter die Lupe nehmen und auch den Anschluß einer Maus durchführen, ganz abgesehen von den vielfältigen Möglichkeiten die uns der Userport bietet, um Hardware-Erweiterungen zu bedienen.
Im übrigen sollte ich noch darauf hinweisen, daß Sie zum vollen Verständnis dieses Kurses doch schon tiefergreifende Kenntnisse von der Programmierung in Maschinensprache haben sollten, sowie in der Handhabung eines Maschinensprache-Assemblers und eines Speichermonitors.
Nichts desto trotz können auch BASIC- Programmierer einiges hier lernen, was eventuell auch von BASIC aus genutzt werden kann, jedoch mit Sicherheit nicht in der komplexen und vielfältigen Art und Weise, wie dies von Maschinensprache aus möglich ist.

2. Die Hardware:
Zunächst jedoch einmal eine kleine Beschreibung, mit was für Bausteinen wir es überhaupt zu tun haben. Die beiden CIAs des C64 sind zwei unscheinbare 40-polige Microchips mit vielfältigen Möglichkeiten. Man könnte sie quasi als "Manager" unseres Computersystems bezeichnen, die die Verbindung zwischen den einzelnen Ein- und Ausgabeeinheiten herstellen und deren Zusammenwirken erst richtig möglich machen.
Beide CIAs sind baugleich und können somit also problemlos miteinander vertauscht werden (was oft bei einer Prüfung auf Funktionsstörungen schon zu einer eindeutigen Analyse führen kann - trotzdem sei von einer Nachahmung ohne Vorkenntnisse abgeraten). Sie tragen die Bezeichnung MOS 6526 und befinden sich in der Ecke links oben auf der Mutterplatine unseres Rechners.
Soviel zur hardwaremäßigen Einordnung dieser kleinen Helfer, jedoch möchten wir uns hier ja mit der softwaremäßigen Bedienung befassen, weshalb ich nun also zu den für uns interessanten Fähigkeiten komme. Die CIAs beinhalten jeweils:

- Zwei 16- Bit-Timer, mit denen man hervorragend besonders zeitkritische Programm-Probleme lösen kann.
- Eine 24- Stunden Echtzeituhr, die die Zeit im Gegensatz zur interruptgesteuerten BASIC-Uhr TI$ extrem genau geht.
- Zwei freiprogrammierbare Datenports, mit denen auch komplexe Datenübertragungen über den Userport möglich werden.
  Wir unterscheiden die beiden CIAs im Folgenden mit CIA1 und CIA2 . Sie werden vom Betriebssystem für unterschiedliche Aufgaben genutzt, die nun ebenfalls beschrieben werden sollen:
  - CIA1 ist mit der Tastatur und den beiden Joystickports verbunden und ist somit für die Eingabe über Tastatur, Joysticks, Maus oder Paddles zuständig.
    Des Weiteren wird von ihm der Systeminterrupt gesteuert, der zyklische Aufgaben, wie das Empfangen von Tastencodes oder das Weiterzählen der BASIC-Uhr TI$

erledigt (dazu später mehr).

- CIA2 ist für die komplette Daten Ein-/ Ausgabe zuständig. Er steuert den IEC-Bus, mit dem bis zu 4 Diskettenlaufwerke und maximal 2 Drucker angesteuert werden können. Des Weiteren ist er am Kasettenport angeschlossen und seine Datenleitungen sind am Userport herausgeführt, wodurch auch PC-Standard- Schnittstellen wie RS-232 (seriell) oder CENTRONICS (parallel) softwaremäßig emuliert werden können.

Das Wichtigste, was wir zur Interruptprogrammierung wissen müssen ist, daß der CIA1 mit der IRQ-Leitung und der CIA2 mit der NMI-Leitung des Prozessors verbunden ist. Je nach Aufgabengebiet einer Interruptroutine müssen wir also unterscheiden, von welchem CIA die Interrupts ausgelöst werden. In der Speicherkonfiguration des 64ers sind die beiden Chips getrennt an zwei verschiedenen Basisadressen eingebunden. Ihre Register sind jedoch aufgrund der Baugleichheit für die gleichen Funktionen zuständig, weshalb wir auch nur EINE Registertabelle benötigen. Es kommt halt nur drauf an, welchen der beiden Zwillinge wir ansprechen wollen. Die Basisadresse für CIA1 ist $DC00 (= dez.56320), für CIA2 $DD00(= dez.56576). Wollen wir also Timer A (dazu später) von CIA1 mit dem Grundwert 16384 initialisieren, so müssen wir die Register 4 und 5 ab $ DC00 ($DC04 und $DC05) mit dem LO/ HI-Byte von 16384 beschreiben, bei Timer A von CIA2 ebenfalls Register 4 und 5, jedoch diesmal ab Basisadresse $DD00($DD04 und $DD05).

3. Was ist ein Interrupt?

Nun zu einigen grundlegenden Informationen zu Interrupts. Ich benutzte dieses Wort die ganze Zeit schon, ohne zu erklären was es überhaupt bedeutet (obwohl Sie sich darin vielleicht schon durch den Raster-IRQ- Kurs auskennen).
Interrupt ist englisch und heißt wörtlich übersetzt "Unterbrechung". Der Prozessor des C64 besitzt, wie jeder andere Prozessor auch, sogenannte Interrupt-Eingänge. Beim Prozessortyp 6510(wie er in unserem Rechner Verwendung findet) sind dies insgesamt drei Leitungen, womit er zwischen drei verschiedenen Interrupts (rein hardwaremäßig - softwaremäßig sind es sogar noch mehr) unterscheiden kann. Diese sind IRQ, NMI und RESET. Diese drei Leitungen können nun extern, von anderen Bausteinen, wie zum Beispiel (und vor allem) von den CIAs angesprochen werden um dem Prozessor das Eintreten eines bestimmten Ereignisses zu signalisieren. Der Prozessor bemerkt dies und kann nun durch ganz bestimmte Maßnahmen auf die Bearbeitung eines Ereignisses eingehen.
Der Clou an der Sache ist, daß der Prozessor so nicht ständig auf das Eintreten eines Ereignisses warten muß und deshalb beispielsweise nicht ständig in einer Endlosschleife prüfen muß, ob in irgendeiner Speicherzelle irgendwann einmal ein bestimmter Wert steht, sondern er bekommt diese Arbeit von den CIAs abgenommen, die ihn schlichtweg nur noch darauf aufmerksam machen, daß er nun seine Achtung etwas anderem schenken sollte - dem Interruptereignis. So kann er also auch gerade mit ganz anderen Dingen beschäftigt sein - nämlich mit der Abarbeitung eines Programms - und trotzdem zwischendurch ganz gezielten Aufgaben nachgehen.
In der Praxis sieht das so aus, daß er seine momentane Arbeit - das Hauptprogramm - dann UNTERBRICHT und in ein Jobprogramm zur Bearbeitung des Interrupts springt. Ich möchte hier zur Verdeutlichung einmal ein Beispiel aus dem Alltag bringen.
Ich, Uli Basters, sitze hier an meinem Rechner und bin gerade dabei, den ersten Teil des CIA-Kurses zu schreiben. Plötzlich klingelt das Telefon. Bevor ich aufstehe um zum Telefon zu gehen speichere ich schnell noch das bisher geschriebene ab und merke mir vor, daß ich nach dem Telefonat unbedingt weiterschreiben werde. Am anderen Ende ist mein Kollege Ralf Zwanziger, der mir den nächsten Redaktionsschluß durchgibt. Nachdem ich aufgehängt habe erinnere ich mich an mein Vorhaben, gehe wieder zurück zum Rechner, lade den Text wieder ein und setze meine Arbeit fort.
Diesen Vorgang kann man sehr gut mit den Tätigkeiten des Prozessors beim Eintreten eines Interrupts vergleichen. Eine Interruptleitung signalisiert ihm, daß ein Interruptereignis

eingetreten ist (das Telefon klingelt). Schnell merkt er sich noch die wichtigsten Daten, nämlich den Inhalt der Prozessorregister (Akku, X und Y-Register), den Prozessorstatus (Speicherung des Textes) und den Inhalt des Programmzählers (das Vormerken weiterzuarbeiten). Danach springt er auf eine Jobroutine, die er für das Eintreffen eines Interrupts parat hat und arbeitet diese dann ab (ich führe ein Telefonat) . Ist er am Ende dieser Routine angelangt, so holt er sich Programmzähler, Status- und Prozessorregister wieder ins Gedächtnis zurück (Erinnerung weiterzuarbeiten und wieder einladen des Textes) und setzt seine alte Arbeit wieder fort.
Diesen ganzen Vorgang erledigt er mit einer derart affenartigen Geschwindigkeit, daß wir meinen er würde beides gleichzeitig tun. Das wäre dann auch der nächste Vorteil der ganzen Geschichte.
Durch Interrupts ist man also in der Lage mehrere Dinge, ganz unabhängig voneinander, scheinbar gleichzeitig zu erledigen - so auch das Betriebssystem, das, während es in der Hauptschleife auf Tasteneingaben wartet, über einen Interrupt den Cursor weiterhin blinken läßt. Die Möglichkeiten hier sind sehr vielfältig, wie wir noch bemerken werden.

4. Der Timer-Interrupt:
Soviel zu den Vorgängen innerhalb unseres Rechners. Nun möchte ich mich ein wenig mit den Unterschieden zwischen den drei Interruptarten beschäftigen.
Wir haben also insgesamt drei verschiedene Unterbrechungen. Eine davon, nämlich der IRQ wird Ihnen vielleicht, wenn auch unbewußt, vielleicht schon nur zu gut bekannt sein. Er wird vom CIA1 ausgelöst und vom Betriebssystem für interne, zyklische Aufgaben verwendet. Deshalb ist er auch ein gutes Beispiel für uns um in dieser Materie einzusteigen, da uns das Betriebssystem schon eine komplette IRQ-Routine zur Verfügung stellt - den System-IRQ. Der System-IRQ nutzt die einfachste und zugleich auch vielseitigste Funktion des CIAs um Interrupts auszulösen - den Timerinterrupt. Bevor ich mich jetzt jedoch in unverständlichen Erklärungen verliere erst einmal eine Registerbeschreibung eines CIA-Registers. Hierzu habe ich Ihnen eine Grafik vorbereitet, die eine Kurzbeschreibung der Register liefert; ausgedruckt sollte sie Ihnen immer parat liegen, da wir sie in Zukunft häufiger benutzen werden.

WR	Name	Beschreibung
00	PRA	Datenregister Port A
01	PRB	Datenregister Port B
02	DDRA	Datenrichtungsregister Port A
03	DDRB	Datenrichtungsregister Port B
04	TA LO	LOW-Byte für Timer A
05	TA HI	HIGH-Byte für Timer A
06	TB LO	LOW-Byte für Timer B
07	TB HI	HIGH-Byte für Timer B
08	TOD 10´	Hardwareuhr: 1/10 Sekunden
09	TOD SEC	Hardwareuhr: Sekunden
10	TOD MIN	Hardwareuhr: Minuten
11	TOD HR	Hardwareuhr: Stunden
12	SDR	Serielles Schieberegister
13	ICR	Interrupt Control Register
14	CRA	Control Register für Timer A
15	CRB	Control Register für Timer B

Wie man leicht erkennen kann sind zur Timerprogrammierung sechs Register notwendig:

TA-LO (Reg.4)
TA-HI (Reg.5)
TB-LO (Reg.6)
TB-HI (Reg.7)
CRA (Reg.14)
CRB (Reg.15)

Des Weiteren brauchen wir auch noch das Interrupt-Control-Register (Reg.13). Ohne dieses Register läuft interruptmäßig überhaupt nichts mit dem CIA. Der System-IRQ benutzt nun Timer A, der für ihn den Auslöser darstellt. Deshalb können wir die Register TB-LO, TB-HI und CRB vorläufig einmal ausschließen und uns nur den Registern für Timer A zuwenden. Zunächst möchte ich jedoch den Begriff "Timer" definieren. Ein Timer, so wie er pro CIA ja zweimal vorhanden ist, ist nichts anderes als ein spezielles Zählregister, das ständig, nach ganz bestimmten Regeln von einem Maximalwert in Einerschritten heruntergezählt wird. Ist der Timer bei Null angelangt, so wird ein Interrupt ausgelöst.
Den schon angesprochenen Maximalwert müssen wir, wie Sie sicher schon vermuten in LO/ HI-Byte aufgespaltet in TA-LO und TA-HI schreiben. Diese Timerregister haben quasi ein "Doppelleben". Sie bestehen nämlich zum Einen aus einem Speicherregister, dem sogenannten "LATCH" und einem Zählregister oder auch "COUNTER" . Schreiben wir nun einen Wert in die Timerregister, so wird dieser zunächst im Latch abgelegt und gleichzeitig in den Counter geladen. Starten wir anschließend den Timer, so beginnt der CIA damit, den Counter herabzuzählen. Hierzu kann man verschiedene Signalquellen wählen, die ein Herabzählen veranlassen. Diese Signalquellen, können im Control-Register für Timer A festgelegt werden. Jedes der acht Bits in diesem Register steuert eine ganz bestimmte Funktion. Des Weiteren kann von hier auch der Timer gestartet und gestoppt werden, sowie einige bestimmte Zählmodi eingestellt werden. Zur Erläuterung habe ich Ihnen einmal eine Tabelle erstellt:

Bit0 (START/STOP)	Durch Löschen dieses Bits wird der Timer gestoppt. Setzt man es, so wird begonnen, den Counter herabzuzählen.
Bit1 (PB ON/ OFF)	Dieses Bit eignet sich besonders für Hardware-Erweiterungen. Ist es gesetzt, so wird beim Unterlauf des Timers ein Signal auf die Portleitung PB6 (Bit 6 an Port B) gelegt. Ist es gelöscht, so werden keine Signale ausgegeben.
Bit2 (TOGGLE-PULSE)	Dieses Bit arbeitet nur in Zusammenhang mit "PB ON/ OFF". Ist dieses gesetzt, so gelten folgende Bestimmungen für "TOGGLE-PULSE":

- Wenn gelöscht, so werden wie bei "PB-ON/ OFF" beschrieben Signale auf die Leitung PB 7 gelegt. Diese sind übrigens genau einen Prozessortaktzyklus lang.
- Wenn gesetzt, so wird der Zustand von PB6 bei jedem Unterlauf des Counters in den jeweils anderen Zustand gekippt, das heißt von Gesetzt auf Gelöscht und umgekehrt. Damit läßt sich also ganz einfach ein Rechtecksignal erzeugen, wobei die Pulsbreite von der Laufzeit des Counters abhängt.

  Wie Sie sehen, sind die Bits 1 und 2 für ganz spezifische Aufgaben verwendbar und haben leider sehr wenig mit Interrupts zu tun, zumal kein Interruptsignal an den Prozessor gesandt wird. Für manche Harwareerweiterungen sind Sie jedoch bestimmt sinnvoll einzusetzen, da man so sehr einfach beliebige Taktfrequenzen für irgendwelche Schaltungen am User-Port des C64 erzeugen kann, da die Leitung PB6 an selbigem herausgeführt ist. Hierzu jedoch in einem späteren Kursteil mehr.

Bit3 (ONE-SHOT/CONTINOUS)	Ist dieses Bit gelöscht, so befindet sich der Timer im CONTINOUS-Modus. Das heißt, daß der Counter bis 0 heruntergezählt, wieder mit dem Wert im Latch geladen wird und von neuem beginnt zu zählen. Bei gesetztem Bit ist der ONE-SHOT-Modus aktiv - es wird bis 0 gezählt und neu geladen, jedoch stoppt der Timer jetzt automatisch, solange bis man ihn wieder mit Bit0 des Controlregisters in Gang setzt.
Bit4 (FORCE-LOAD)	Wird dieses Bit gesetzt, so wird der Counter, egal ob der Timer im Moment läuft oder nicht, mit dem Wert im Latch neu geladen.
Bit5 (IN MODE)	Hier wird die Quelle des "Timer-Triggers" festgelegt. Der Timer-Trigger ist die Einrichtung, die den CIA dazu veranlaßt den Counter einmal herunter zuzählen. Ist dieses Bit gelöscht (was bei uns eigentlich immer der Fall ist), so wird der Systemtakt als Trigger herangezogen (das werden wir im nächsten Abschnitt ganz genau behandeln). Ist es gelöscht, so ist die CNT-Leitung des CIA der Trigger. Diese ist an den Userport herausgeführt, so daß somit auch Hardware-Erweiterungen in der Lage sind die Interrupts im 64 er extern zu steuern.
Bit6 (SP-DIRECTION)	Dieses Bit hat etwas mit Register 12 der CIA zu tun (SDR). Dieses Register ist für die serielle Datenübertragung sehr nützlich. Hier kann ein 8- Bit-Wert gespeichert werden, der zyklisch aus dem Register heraus, an den Pin SP des CIA (mit dem Userport verbunden) gerollt wird, beziehungsweise ein Wert kann über den Pin SP hereingerollt werden (auch dazu in einem späteren Kursteil mehr). Bit6 von CRA steuert nun die Datenrichtung von SDR. Ist es gelöscht, so ist SDR auf Eingang geschaltet (Bits werden hereingerollt), ist es gesetzt, so wird SDR als Ausgang benutzt (Bits werden herausgerollt).
Bit7 (POWER FREQUENCY)	Dieses Bit wird benötigt um den Triggerfrequenz für die Echtzeituhr des CIAs zu bestimmen. Je nach dem, in welchem Land wir unseren 64 er angeschlossen haben, beträgt die Netzfrequenz des Wechselstroms aus der Steckdose nämlich 50 oder 60 Hertz (man spricht hier vom sogenannten "technischen Strom") . Diese Frequenz wird nun benutzt um die Zeiteinheiten der Echtzeituhr festzustellen. Ist dieses Bit nun gesetzt, so gilt eine Frequenz von 50 Hz als Trigger, ist es gelöscht, eine von 60 Hz. Da wir in der Bundesrepublik Deutschland ja die von 50 Hz haben, sollte bei Uhr-Betrieb dieses Bit also immer gesetzt sein, da andernfalls unsere Uhr schnell "nach-gehen" könnte.

Na das ist doch schon eine ganze Menge an Information. Doch keine Angst, die Bits 1 und 2, die etwas komplizierter erscheinen mögen, wollen wir vorläufig erst einmal wegfallen lassen. Von ihnen wird, ebenso wie von Bit 6 und 7, in einer späteren Folge dieses Kurses mehr die Rede sein.

Nun zum Systemtakt, der - in aller Regel als Timer-Trigger dient. Er stellt wohl einen der wichtigsten Grundbausteine in unserem (wie auch in jedem anderen) Rechner dar. Ein Rechensystem, so wie es von einem Computer verkörpert wird, braucht schlichtweg IMMER einen Grundtakt, den alle miteinander verknüpften Bausteine benutzen können um synchron miteinander zu arbeiten. Er dient sozusagen als " Zeitmaß" für die Geschäftswelt in einem Rechner. Ein Taktzyklus stellt die elementare Zeiteinheit innerhalb eines Rechner dar, an den sich alle Bausteine halten müssen um mit ihren Signalen nicht zu kollidieren. Solche Takte

werden von Quarz-Bausteinen erzeugt, die, je nach chemischer Zusammensetzung, eine ganz bestimmte Eigenfrequenz haben, die extrem genau ist (wie von Quartz-Uhren her ja allgemein bekannt).
Ein Systemtakt ist von Rechner zu Rechner verschieden. Beim AMIGA beträgt er zum Beispiel 7 .16 MHz (Megahertz), beim ATARI ST 8 MHz, bei PCs mittlerweile zwischen 4.77 und 33 MHz (und mehr). Je höher ein Rechner "getaktet" ist, desto mehr Befehle kann er pro Sekunde abarbeiten und desto schneller ist er; weshalb die Taktfrequenz häufig auch als ein Maß für die Rechengeschwindigkeit eines Rechners herangezogen wird.
Der C64 schneidet hier, aufgrund seiner schon etwas älteren Entwicklung und dem Fakt, daß er halt einfach nur ein Homecomputer (der für jeden erschwinglich sein soll) ist, relativ schlecht ab. Mit etwa 1 MHz ist er vergleichsmäßig langsam, was jedoch immer noch affenschnell ist! Um genau zu sein sind es 985248.4 Hz (also knapp ein Mhz). So zumindest bei der europäischen Version, die Sie ja alle haben sollten. Die amerikanischen 64 er sind sogar noch ein wenig schneller (nämlich 1022727.1 Hz), was für uns jedoch unerheblich ist.
Doch was bedeutet diese Zahl nun eigentlich für uns. Sie bedeutet schlichtweg, daß wir pro Sekunde genau 985248.4 Taktzyklen haben; und die brauchen wir ja als Timer-Trigger. Damit wird es einfach, die Dauer zwischen zwei Counter-Unterläufen zu berechnen. Angenommen, Sie wollten (aus welchem Grund auch immer), daß jede 1/16- Sekunde ein Interrupt ausgelöst würde. Demnach müßten Sie den Systemtakt einfach durch 16 dividieren und das Ergebnis in LO/ HI-Byte aufgespalten in die Register TA-LO und TA-HI schreiben. Konkret wäre das:

```
985248.4 / 16 = 61578.025 (dez.)
                $F08A (hex.)
          LO:   $8A = 138
          HI:   $F0 = 240
```

Schreiben Sie diese beiden Werte nun in die Register 4 und 5 des CIA1, so wird der System-interrupt, der ja durch Timer A der CIA1 gesteuert wird, eindeutig verlangsamt (normalerweise tritt er nämlich jede 1/60- Sekunde auf). Erkennen können Sie dies am langsameren Cursor-blinken, da auch das vom Systeminterrupt erledigt wird. Probieren Sie es doch einfach einmal, also:

```
POKE 56320+4,138:POKE 56320+5,240        (56320 ist die Basisadresse von CIA1!)
```

So. Nun wissen Sie also, wie man den Timer A eines CIA programmiert. Da dieser für IRQs jedoch schon vom Betriebssystem benutzt wird, und es da möglicherweise Timing-Probleme gibt, wenn wir eine eigene IRQ-Routine schreiben wollen, die zwar parallel zum Betriebssystem-IRQ läuft, aber öfter oder weniger oft als dieser auftreten soll, so gibt es für uns auch die Möglichkeit auf Timer B auszuweichen. Dieser ist absolut analog zu bedienen, nur, daß wir die Timerwerte diesmal in die Register 6 und 7 der CIA schreiben müssen (TB-LO und TB-HI). Des Weiteren wird er vom Control-Register-Timer-B (CRB) gesteuert - Register 15 der CIA also. Bis auf kleinere Unterschiede, ist der Aufbau der Register CRA und CRB identisch. Hier die Unterschiede:

1. Grundsätzlich sind die Funktionen der Bits 0 bis 4 gleich, jedoch mit dem Unterschied, daß sich die Bits 1 und 2 nicht mehr auf PB6 sondern auf PB7 beziehen.
2. Bit 5 und 6 steuern den IN-MODE von Timer B (bei CRA war das NUR Bit 5). Hierzu ergeben sich 4 verschiedene Timer-Trigger-Modi:

Bit 5	6	Funktion
0	0	Zähle Systemtakt
0	1	Zähle CNT-Flanken
1	0	Zähle Unterläufe von Timer A
1	1	Zähle Unterläufe von Timer A, wenn CNT=1

Somit kann Timer B bei Steuerung durch Hardware bestens eingesetzt werden, da mehr externe Steuermöglichkeiten (durch CNT) vorhanden sind.

3. Bit 7 steuert die Alarmfunktion der Echtzeituhr. Ist es gesetzt, so werden die Werte, die man in die Register der Echtzeituhr schreibt (dazu auch in einem späteren Kursteil) als Alarmzeit genommen. Ist es gelöscht, so kann die normale Uhrzeit eingestellt werden. Da die beiden Register CRA und CRB ebenfalls eine sehr wichtige Funktion bei den vielseitigsten Anwendungen erfüllen, habe ich Ihnen einmal eine grafische Übersicht angefertigt:

Bit	**CRA**	**CRB**
0	0 = Stop Timer;	0 = Stop Timer
	1 = Start Timer	1 = Start Timer
1	1 = Zeigt einen Timer Unterlauf an Port B in Bit 6 an	1 = Zeigt einen Timer Unterlauf an Port B in Bit 7 an
2	0 = Bei Timer Unterlauf wird an Port B das Bit 6 invertiert	0 = Bei Timer Unterlauf wird an Port B das Bit 7 invertiert
	1 = Bei Timer-Unterlauf wird an Port B das Bit 6 für einen Systemtaktzyklus High	1 = Bei Timer-Unterlauf wird an Port B das Bit 7 für einen Systemtaktzyklus High
3	0 = Timer-Neustart nach Unterlauf (Latch wird neu geladen)	0 = Timer-Neustart nach Unterlauf (Latch wird neu geladen)
	1 = Timer stoppt nach Unterlauf	1 = Timer stoppt nach Unterlauf
4	1 = Einmalig Latch in den Timer laden	1 = Einmalig Latch in den Timer laden
5	0 = Timer wird mit der Systemfrequenz getaktet,	Bit 5..6: Timer wird getaktet %00 = mit der Systemfrequenz %01 = von positiver Flanke am CNT-Pin %10 = vom Unterlauf des Timer A %11 = vom Unterlauf des Timer A, wenn CNT-Pin High ist
	1 = Timer wird von positiven Flanken am CNT-Pin getaktet	
6	Richtung des seriellen Schieberegisters, 0 = SP-Pin ist Eingang (lesen)	
	1 = SP-Pin ist Ausgang (schreiben)	
7	Echtzeituhr, 0 = 60 Hz	0 = Schreiben in die TOD-Register setzt die Uhrzeit
	1 = 50 Hz	1 = Schreiben in die TOD-Register setzt die Alarmzeit

In der Magic-Disk Ausgabe scheint sich ein Fehler eingeschlichen zu haben, da hier die Grafik des ICR Registers gezeigt wurde.

Das Interrupt-Control-Register (ICR)

So. Nun wissen wir also, wie man die Timer der CIAs steuert. Interrupts haben wir nun aber noch lange nicht! Die Timer dienen ja lediglich als Interrupt-Quellen. Wir benötigen noch ein weiteres Register des CIA um ihm zu sagen, daß beim Unterlauf eines Timers auch die IRQ-Leitung (beim CIA1, bzw. NMI-Leitung beim CIA2) des Prozessors zu aktivieren ist, um einen Interrupt zu signalisieren. Dieses Register ist Register 13 eines CIAs, das Interrupt Control Register (ICR) . Es ist wohl das wichtigste Register im CIA überhaupt, denn ohne es könnten wir mit ihm überhaupt nichts anfangen!

Bevor wir also einen Timer zur Interrupterzeugung starten, sollten wir also immer im ICR auch angeben, daß dieser Timer Interrupts erzeugen soll. Darüber hinaus gibt es auch noch eine ganze Menge anderer Interruptquellen, die dieses Register steuert. Ich gebe Ihnen hier einmal eine tabellarische Übersicht der einzelnen Bits vom ICR. Die Bits müssen gesetzt sein, um einen Interrupt auszulösen, wenn das entsprechende Ereignis eintritt:

Bit0 Löse einen Interrupt aus, wenn Timer A unterläuft.
Bit1 Löse einen Interrupt aus, wenn Timer B unterläuft.
Bit2 Löse einen Interrupt aus, wenn die Alarmzeit der Echtzeituhr mit der aktuellen Zeit übereinstimmt.

Bit3 Löse einen Interrupt aus, wenn das SDR (Serial Data Register) voll, bzw. leer ist (abhängig von der entsprechenden Betriebsart - heraus- oder hereinrollen).
Bit4 Löse einen Interrupt aus, wenn am Pin FLAG (am Userport herausgeführt) ein Signal anliegt.
Bit5 Unbelegt.
Bit6 Unbelegt.
Bit7 Doppelfunktion (siehe unten).

Wie Sie sehen, ist es ganz einfach, bestimmte Interruptquellen zu wählen. Sogar von externer Hardware können DIREKT Interrupts ausgelöst werden.
Nun jedoch noch zu der Sonderfunktion von Bit 7. Man muß beim ICR nämlich unterscheiden, ob man nun in das Register schreibt, oder ob man es ausliest. Es hat nämlich, wie die Timerregister auch, eine Doppelfunktion. Man unterscheidet zwischen einem Latch-Register, in dem die Interrupt-Maske (INT-MASK) gespeichert wird und den Interrupt-Daten (INT-DATA).
Schreiben wir in das ICR, so wird der geschriebene Wert zwar zwischengespeichert, jedoch können wir ihn nicht lesen. Denn wenn wir lesen, gibt uns das ICR augenblickliche Informationen, ob und welches Interruptereignis eingetreten ist, nicht aber, welchen Wert wir vorher hineingeschrieben haben.
An diese Doppelfunktion von Registern sollten Sie sich gewöhnen, denn wir werden noch öfter damit zu tun haben.
Lesen wir nun aus dem ICR Daten aus, so zeigt uns Bit 7 an, ob eines der zugelassenen Interruptereignisse eingetreten ist, das heißt, daß Bit 7 immer dann gesetzt ist, wenn mindestens ein Bit von INT-MASK mit einem Bit von INT-DATA übereinstimmt. Dadurch haben wir eine einfache Kontrolle, ob ein Interrupt auch tatsächlich vom CIA ausgelöst wurde, oder nicht doch von was anderem (wie zum Beispiel vom VIC, der ja die Raster-Interrupts auslöst) .
Durch eine einfache Abfrage mit dem Assembler-Befehl "BMI" (Branch on MInus), der ja den Zustand von Bit 7 überprüft, können wir schnell feststellen, von wo der Interrupt nun kommt.
Schreiben wir in das ICR, so ist Bit 7 nochmal doppeldeutig:
Ist es nämlich gelöscht, so wird jedes weitere 1-Bit sein korrespondierendes Maskenbit in INT-MASK löschen. Die anderen Bits bleiben unberührt davon. Es wird also quasi ein AND mit dem Wert den wir schreiben und dem Wert der in INT-MASK steht vollzogen.
Ist Bit 7 jedoch gesetzt, so wird jedes weitere 1-Bit sein korrespondierendes Masken-Bit setzen. Die anderen bleiben ebenfalls davon unberührt. Diesmal wird also ein OR mit den beiden Werten vollzogen. Damit können wir also problemlos ganz gezielt Bits im ICR setzen und löschen, ohne dabei aus Versehen andere Bits zu verändern.
Auch hier will ich Ihnen eine grafische Übersicht liefern, damit Sie die Bittabelle vom ICR immer auf Papier parat haben können.

Bit	Funktion
0	Unterlauft Von Timer A
1	Unterlauft Von Timer B
2	Uhrzeit und Alarmzeit sind gleich
3	SDR ist Voll / Leer (je Betriebsart)
4	Am Pin „FLAG“ liegt ein Signal an
5	unbenutzt
6	unbenutzt
7	Lesend: mindestens 1 Maskenbit= Datenbit Schreibend: 0 = lösche korrespondierendes Maskenbit 1 = setze korrespondierendes Maskenbit

Das war's dann mal fürs erste. Ich hoffe, ich habe Ihnen nun mit den (zugegeben) überaus trockenen Grundlagen der Interrupt-Programmierung, nicht das Interesse am Thema dieses Kurses genommen.
Keine Panik, im nächsten Monat werden wir uns dann einmal um die praktische Anwendung kümmern. Ich zeige Ihnen dann einmal den kompletten Ablauf des Systeminterrupts, auf dem wir dann unsere ersten Schritte in Sachen Interrupt aufbauen werden. Bis dahin Servus,

Ihr Uli Basters (ub)

Teil 2 – Magic Disk 12/90

Herzlich willkommen zur zweiten Runde unseres CIA-Kurses. Nachdem ich Sie letzten Monat ja lange genug mit der trockenen Theorie von der Bedienung der Timer der CIAs gelangweilt habe, will ich jetzt einmal den Grundstein zur Praxis legen. Ich möchte mich diesmal mit dem eigentlichen Ablauf eines Interrupts beschäftigen und Ihnen einen solchen auch haarfitzelgenau anhand des Systeminterrupts erklären.

1. Der Prozessor und ein Interrupt.
Wie ich schon im letzten Teil erwähnte, wird der Systeminterrupt vom Betriebssystem des 64 ers zur Erledigung verschiedener zyklischer Aufgaben genutzt.
Er wird 60 mal in der Sekunde aufgerufen und von Timer A der CIA1 erzeugt. Demnach haben wir also einen IRQ (wir erinnern uns: CIA1 erzeugt Impulse an der IRQ-Leitung des Prozessors, CIA2 an der NMI-Leitung). Die Timerregister TALO und TAHI beinhalten die Werte 37 und 64. Der Timer zählt also von 16421(= HI*256+ LO) bis 0 herunter und löst dann einen Interrupt aus. Das ist auch ganz logisch, denn wenn wir ja 60 Interrupts in der Sekunde haben wollen, dann müssen wir ja den Systemtakt durch 60 dividieren. Das Ergebnis hiervon ist 16420.8, aufgerundet 16421! Ich hoffe Sie verstehen jetzt, warum ich mich zunächst um die Timer selbst gekümmert hatte, da Sie nun auch besseren Einblick in den Systeminterrupt haben. Wenn Sie einmal ein bisschen rumprobieren möchten, bitte:
Schreiben Sie doch einfach einmal mittels POKE einen höheren oder niedrigeren Wert als 64 in TAHI (Reg.5 von CIA1).
Das Ergebnis sehen Sie dann am Cursorblinken, was ebenfalls vom Systeminterrupt erledigt wird. Der Cursor sollte nun entweder schneller (bei niedrigerem Wert) oder langsamer (bei höherem Wert) blinken. Übertreiben Sie die Werte jedoch nicht übermäßig, da auch die Tastaturabfrage vom Systeminterrupt erledigt wird, und Sie so entweder einen Turbo-Cursor haben, mit dem bei eingeschaltetem Key-Repeat (das ist bei meinem Floppy-Speeder-Betriebssystem nämlich der Fall, und ich bin eben beim Ausprobieren natürlich prompt wieder darauf hereingefallen...) keine vernünftigen Eingaben gemacht werden können, ebenso wie bei einem ultra langsamen Cursor, wo es noch bis morgen dauern würde, einen rücksetzenden POKE-Befehl einzugeben.
Dieser Trick wird übrigens oft benutzt, um Programme schneller zu machen. Je öfter nämlich ein Interrupt pro Sekunde auftritt, desto weniger Zeit hat der Prozessor, das momentan laufende Hauptprogramm abzuarbeiten. Setzt man jedoch die Zahl der Interrupts pro Sekunde herunter, oder schaltet man ihn sogar ganz ab (indem man den Timer einfach anhält, oder mittels des Assemblerbefehls SEI Interrupts ganz sperrt), so läuft das Hauptprogramm logischerweise schneller ab. Dies nur ein Tip am Rande.
Was geht nun eigentlich in unserem 64 er vor, wenn ein Interrupt abgearbeitet werden soll? Nun, zunächst einmal haben wir da einen Unterlauf von Timer A der CIA1. Diese legt sodann auch gleich ein Signal an die IRQ-Leitung des Prozessors an und markiert die Interruptquelle "Timer A" in ihrem ICR.
Folgende Prozesse gehen nun im Prozessor vor:

1. Der Prozessor überprüft nun nach jedem abgearbeiteten Befehl den Zustand der Unterbrechungsleitungen (IRQ, NMI) . Ist eine davon gesetzt, und ist der zugehörige

Interrupt auch freigegeben, so beginnt er damit, die Unterbrechung zu bearbeiten. Hierzu müssen zunächst die wichtigsten Daten auf den Stapel gerettet werden. Das sind in der hier gezeigten Reihenfolge:

- HI-BYTE des Programmzählers
- LO-BYTE des Programmzählers
- Prozessorstatusregister

Der Programmzähler ist ein Prozessorinternes Register, das anzeigt, an welcher Speicheradresse, der nächste zu bearbeitende Befehl liegt. Das Prozessorstatusregister beinhaltet die Flaggen, die dem Prozessor bei Entscheidungen weiterhelfen (ich verweise da auf die letzten Kursteile des Assemblerkurses meines Kollegen RALF TRABHARDT).

2. Der Prozessor setzt sich selbst das Interrupt-Flag und verhindert so, daß er durch weitere Interrupts gestört wird.
3. Ganz am Ende des Speichers (in den Adressen $ FFFA-$ FFFF) sind 6 Bytes für das Interrupt-Handling reserviert. Dort stehen insgesamt 3 Vektoren, die dem Prozessor zeigen, bei welcher Unterbrechung er wohin springen muß, um einen Interrupt zu bearbeiten. Diese Vektoren heißen im Fachjargon übrigens auch Hardware-Vektoren. Hier einmal eine Auflistung:

Interrupt	**Vektor**	**Adresse**
NMI	$FFFA/$FFFB	$FE43 (65091)
RESET	$FFFC/$FFFD	$FCE2 (64738)
IRQ,BRK	$FFFE/$FFFF	$FF48 (65352)

Da wir ja einen IRQ behandeln, holt sich der Prozessor jetzt also die Adresse, auf die der Vektor in $FFFE/$ FFFF zeigt in den Programmzähler und beginnt so damit eine Jobroutine für den IRQ abzuarbeiten. Diese Jobroutine wollen wir uns jetzt einmal genauer ansehen. Vorher jedoch noch eine kleine Erläuterung. Wie Sie ja sehen, wird der Vektor für den IRQ auch als Vektor für BRK-Unterbrechungen benutzt. Der BRK-Befehl sollte Ihnen ja vielleicht bekannt sein (wenn Sie sich mit Maschinensprache auskennen). Er hat ans ich ja keinen direkten Verwendungszweck, jedoch wird er von vielen Monitor-Programmen zum Debuggen benutzt. Wie Sie ebenfalls sehen können, hat er gleiche Priorität wie ein IRQ, und man kann Ihn also auch als eigenen Interrupt ansehen. Vielmehr ist der BRK-Befehl die Unterbrechungsquelle für BRK-Interrupts.
Wie man mit ihm umgeht, will ich Ihnen später zeigen.
Werfen wir nun jedoch einmal einen Blick auf die Jobroutine ab $FF48. Diese befindet sich natürlich im Betriebssystem-ROM, weshalb ich Ihnen hier auch einen Auszug daraus liefern möchte (ich habe dies in Form einer Grafik getan, damit ich ausführlichere Kommentare zu den einzelnen Programmschritten geben kann).

Betriebssystemroutine zur Bearbeitung eines IRQs / BRKs

```
$FF48     PHA            Akku auf Stapel retten
$FF49     TXA            X-Register in Akku schieben
$FF4A     PHA            und auf den Stapel retten
$FF4B     TYA            Y-Register in Akku schieben
$FF4C     PHA            und auf Stapel retten

$FF4D     TSX            Stackpointer als Zähler ins X-Registers
$FF4E     $LDA $104,X    Akku mit aktuellen Prozessorstatus laden
$FF51     AND #$10       BRK-Flag isolieren
$FF53     BEQ FF58       Wenn Akku = 0, dann ist BRK-Flag gelöscht, also IRQ
$FF55     JMP (0316)     BRK-Routine aufrufen (wenn BRK-Flag gesetzt)
```

```
$FF58        JMP (0314)        IRQ-Routine aufrufen
```

Dies ist also zunächst einmal eine Vorbereitungsroutine für die prioritätsgleichen Interrupts IQ und BRK. Bei beiden sollten AKKU, X- und Y-Register zunächst gerettet werden, wenn wir nicht wollen, daß der Prozessor nach Abarbeiten des Interrupts mit falschen Werten in diesen Registern fortfährt, da so das Hauptprogramm ja unter anderen Bedingungen ablaufen würde. Dies geschieht durch die Transferbefehle im ersten Teil des Listings. Als nächstes muß die kleine Routine herausfinden, ob sie jetzt durch einen IRQ oder einen BRK aufgerufen wurde. Normalerweise ist ja bei einem BRL die BRK-Flag im Prozessorregister gesetzt. Das Dummer hierbei sit jedoch, daß die alten Flaggenwerte mittlerweile schon wieder durch die vorangegangenen Befehle verändert wurden. Um nun also an den alten Prozessorstatus zu kommen, bedient sich das Programm einem kleinen Trick. Es holt sich zunächst einmal den Stackpointer in das X-Register. Wie Sie wissen sollten, werden die Werte auf dem Stackpointer abgelegt. Da der Speicherbereich des Stacks von $0100 bis $01FF geht, wird also der erste Wert , der überhaupt auf den Stapel gelegt wird zunächst in Speicherzelle $01FF landen. Der Stackpointer wird anschließend dekrementiert und zeigt nun auf $01FE. Dies setzt sich bei jedem Wert so fort.
Die Interruptvorbereitungsroutine nutzt nun den Fakt, daß der Stackpointer auch automatisch als Zeiger für den zuletzt geschriebenen Wert dient. Vorausgesetzt, man benutzt die X-indizierte-absolute Adressierung, mit der Anfangsadresse des Stacks plus 1 als Indexbasis. Mit dem Stackpointer im X-Register würde also mit dem Befehl:

```
LDA $0101,X
```

immer der zuletzt auf den Stack geschriebene Wert in den AKKU geladen werden. Würden wir die normale Basisadresse (ohne +1) benutzen, so würden wir auf die Stackadresse zugreifen, da hier noch nichts wissenswertes steht
Da unsere Routine nun weiß, daß der Prozessorstatus zum Zeitpunkt des Interruptauslösens mittlerweile an viertletzter Stelle auf dem Stack steht (wir hatten nachdem der Prozessor den Status gerettet hatte ja auch noch selbst AKKU, X- und Y- Register gerettet), nimmt sie also auch die Stackbasis +1 nochmal plus 3 (für die 4 geretteten Werte), um an den alten Prozessorstatus heranzukommen. Deshalb ist die Indexbasis diesmal $0104. Dasselbe wie:

```
LDA $0104,X
```

hätten wir auch mit der Befehlsfolge:

```
INX
INX
INX
LDA $0101,X
```

erreicht, jedoch ist diese Methode umständlicher und benötigt 3 Bytes mehr. Sie soll hier sowieso nur als erläuterndes Beispiel dienen.
Nachdem wir nun den alten Prozessorstatus im AKKU haben, müssen wir nur noch zusehen, daß wir die BRK-Flagge herausisolieren. Diese befindet sich mit einer $10 (dezimal 16), also dem Wert von Bit 4 verUNDen müssen, um alle anderen Bits zu löschen. War die BRK-Flagge nun gesetzt, so ist der Akkuinhalt ungleich 0 (nämlich $10). War sie gelöscht, so ist er gleich 0. Der anschließende BEQ-Verzweigungsbefehl prüft dies und verzweigt entsprechend auf die zwei indirekte JMP-Befehle, die dann entweder Jobroutine für einen BRK oder die für einen IRQ aufrufen.
Hier ist nun der Punkt, an dem wir später dann in das Interruptsystem des Betriebssystems eingreifen können. Denn die benutzten Vektoren der JMP-Befehle, befinden sich im RAM und sind somit veränderlich. Ändern wir also den Inhalt dieser Vektoren, so können wir das Betriebssystem dazu veranlassen, unsere eigenen Interruptroutinen anzuspringen. Dazu später mehr.

Kommen wir nun zum System-IRQ. War der Auslöser des Interrupts kein BRK-Befehl, sondern wie in unserem Fall die CIA1, dann springt die Jobroutine von oben ja über den IRQ-Vektor in den Adresse $0314 / $0315, Diese Adressen beinhalten normalerweise die Werte $31 und $EA. Demnach befindet sich die Betriebssystemroutine für den IRQ ab Adresse $EA31. Dort werden jetzt diverse Unterroutinen abgearbeitet, die folgende Funktionen haben:

- Basic-Uhr TI$ weiterzählen und RUN/STOP-Taste abfragen
- Steuerung des Motors der Datasette (bei gedrückter Rekordertaste Motor einschalten, bei ungedrückter Rekordertaste ausschalten)
- Tastatur abfragen. Wenn eine Taste gedrückt ist, dann werden der Tastencode und der ASCII-Code der entsprechenden Taste zwischengespeichert. Dazu in einem späteren Kursteil mehr, denn die Tastatur ist ebenfalls an den CIAs angeschlossen.
- Zum Schluß werden die alten Prozessorregister wieder vom Stack geholt und der Interrupt beendet.

Schauen wir uns die IRQ-Routine doch einmal als ROM-Listing an:

Adresse	Befehl	Kommentar
$EA31	JSR $FFEA	Run/Stop-Taste abfragen und TI$ erhöhen
$EA34	LDA $CC	Flag zur Anzeige des Cursorblinkens laden
$EA36	BNE $EA61	Cursor soll im Moment nicht blinken, deshalb überspringen
$EA38	DEC $CD	Blinkzähler erniedrigen (alle 20 iRQs – 1x Blinken)
$EA3A	BNE $EA61	Noch nicht Null, also überspringen
$EA3C	LDA #$14	Blinkzähler wieder auf 20 (#$14) setzen
$EA3E	STA $CD	und speichern
$EA40	LDY $D3	Cursorspalte holen
$EA42	LSR $CF	Carry Bit wird 1, wenn der Blinkschalter 1 ist
$EA44	LDX $0287	Für später noch die Farbe unter dem Cursor holen.
$EA47	LDA ($D1),Y	Ebenso die den Zeichen-Kode
$EA49	BCS $EA5C	Blinkschalter war ein, dann weiter
$EA4B	INC $CF	Blinkschalter war 0 also auf 1
$EA4D	STA $CE	Zeichen unter Cursor merken
$EA4F	JSR $EA24	Zeiger in Farb-RAM berechnen, der auf Cursor zeigt
$EA52	LDA ($F3),Y	Zeiger steht jetzt in $F3/$F4 - Farb-Code holen
$EA54	STA $0287	und merken
$EA57	LDX $0286	Farb-Code unter Cursor in X-Register holen
$EA5A	LDA $CE	gemerktes Zeichen unter Cursor holen
$EA5C	EOR #$80	Zeichen invertieren
$EA5E	JSR $EA1C	Zeichen und Farbe setzen
$EA61	LDA $01	Prozessorport laden
$EA63	AND #$10	Bit für Rekorder-Taste isolieren
$EA65	BEQ $EA71	gedrückt, dann Sprung zum Motor einschalten
$EA67	LDY #$00	Wert 0 für keine Taste gedrückt
$EA69	STY $C0	Rekorder-Flag setzen
$EA6B	LDA $01	Prozessorport laden
$EA6D	ORA #$20	Rekoder-Motor ausschalten
$EA6F	BNE $EA79	unbedingter Sprung
$EA71	LDA $C0	lade Rekorder-Flag
$EA73	BNE $EA7B	verzweige, wenn Motor schon läuft
$EA75	LDA $01	Prozessorport laden
$EA77	AND #$1F	Rekorder-Motor einschalten
$EA79	STA $01	und wieder speichern
$EA7B	JSR $EA87	Tastaturabfrage
$EA7E	LDA $DC0D	ICR von CIA1 durch Lesen löschen ind IRQs freigeben
$EA81	PLA	AKKU aus dem Stapel holen
$EA82	TAY	und in Y-Register schieben
$EA83	PLA	AKKU aus dem Stapel holen
$EA84	TAX	und in X-Register schieben
$EA85	PLA	und Rückkehr vom Interrupt

```
$EA86         RTI
```

Die eigentliche IRQ-Routine, von $EA31 bis $EA7E habe ich Ihnen nur der Vollständigkeit halber hier aufgeführt. Um die einzelnen Operationen richtig zu verstehen, ist es sinnvoll, sich mit der Bedeutung der Zeropageadressen auszukennen, die vom Betriebssystem ja fast alle als schelle Zwischenspeicher benutzt werden. Dies soll jedoch nicht hier unser Thema sein; vielleicht werden wir ja einmal einen eigenen Kurs diesbezüglich starten.
Wichtig für uns ist der letzte Abschnitt der IRQ-Routine. Hier sehen Sie nämlich, wie ein IRQ wieder beendet wird. Die Grafik (ROM-Listing) spricht da für sich selbst:

- Zunächst müssen wir durch Lesen des ICR von CIA1 das IRQ-Flag wieder löschen, so daß neue IRQs durch Timerunterlauf auch wieder erkannt werden (das wissen Sie ja schon aus dem letzten Kursteil).
- In umgekehrter Reihenfolge werden nun die alten Werte von X-, Y-Register und AKKU wieder vom Stapel zurückgeholt.
- Abschließend wird mit dem RTI-Befehl der Interrupt beendet.

Was passiert nun eigentlich im Prozessor, wenn er auf den RTI-Befehl trifft?
Nun, zunächst einmal holt er sich den alten Prozessorstatus wieder vom Stapel zurück. Als nächstes folgen LO- und HI-Bytes des alten Programmzählers (in der hier genannten Reihenfolge). Jetzt sollte der Zustand , der vor der Unterbrechung herrschte, wiederhergestellt sein:

- Softwaremäßig haben wir die drei Prozessorregister zurückgesetzt.
- Der Programmzähler zeigt wieder auf die Adresse, an der der Prozessor vorher aufhörte.
- Das alte Prozessorstatusregister ist wieder an seinem Platz. Hierdurch ist auch die Interrupt-Flagge automatisch wieder gelöscht, da diese ja dem Zustand vor dem Interrupt entspricht; die I-Flagge MUSS gelöscht gewesen sein, sonst wäre ja gar kein Interrupt zustande gekommen! Somit sind also auch wieder neue Interrupts möglich.

Kommen wir nun zu der Routine, die das Betriebssystem für einen BRK-Interrupt bereithält. Wir erinnern uns, nachdem ein Interrupt aufgetreten war, hatte ja eine kleine Routine ab $FF48 dafür gesorgt, daß eine Unterscheidung getroffen wurde, von wo der Interrupt nun eigentlich kam. Im Falle eines IRQs ist sie dann über den Sprungvektor in $0314/$0315 auf die IRQ-Routine des Systeminterrupts gesprungen. Wurde jedoch der Interrupt von einem BRK-Befehl verursacht, so wird über den Vektor in $0316/$0317 gesprungen. Diese beiden Speicherzellen beinhalten die Werte $66 und $FE. Sie zeigen somit auf die Adresse $FE66, wo dann auch die Jobroutine steht, die einen BRK-Interrupt bearbeitet.
Eigentlich ist BRK ja kein richtiger Interrupt (zumindest wird er so nicht vom Betriebssystem genutzt), da seine einzige Interruptquelle ja ein BRK-Befehl selbst ist. Doch das ist auch der Sinn des Befehls. Alle Prozessoren haben einen solchen Befehl eingebaut, der dem eingefleischten Programmierer zu „debuggen“, also dem Entfernen von Parasiten (sprich Fehlern) in einem Programm dienen soll. Durch BRK, können wir nämlich mit Hilfe eines Assembler-Monitors einfach innerhalb eines Assemblerprogramms sogenannte BREAK-Points (BRK) setzen, bei denen automatisch abgebrochen wird, und bei vorher aktiviertem Monitor, in diesen gesprungen wird. Dieser kann uns nun den Inhalt der Prozessorregister genau anzeigen, einschließlich Prozessorstatus und Programmzähler zum Zeitpunkt des BREAKs, da die letztgenannten ja auf den Stapel gerettet wurden.
Ohne Monitor wird nun aber auf die systemeigene Routine verzweigt, die einem Warmstart des Rechners entspricht. Vielleicht haben Sie ja einmal erlebt, was passiert, wenn der Prozessor des C64 auf einen solchen Befehl trifft. Das Ergebnis ist ähnlich dem, das wir erhalten, wenn wir die Tastenkombination RUN/STOP-Restore drücken (dazu will Ihnen auch in einem folgenden Kursteil mehr erzählen):

- Der Bildschirm wird, ebenso wie die Zeichenfarbe, wieder auf die Einschaltfarben

gesetzt.
- Eventuell eingeschaltete Sprites, oder HIRES-Grafikmodi werden angeschaltet und die alte Textausgabe erscheint wieder.
- Sollte der SID gerade dabei gewesen sein, irgendetwas an Musik oder Tönen von sich zu geben, so verstummt er.
- Der Bildschirm wird gelöscht und in der zweiten Zeile erscheint das altbekannte „READY.“, mit einem blinkenden Cursor darunter.

Dies alles wird von einer winzig kleinen Routine bewältigt, die so kleine ist, daß ich Ihnen hierfür nicht extra eine Grafik aufführen muß. Das ist sie:

```
$FE66      JSR $FD15       Standard-Vektoren für Interrupt und I/O setzen
$FE69      JSR $FDA3       I/O initialisieren
$FE6C      JSR $E518       Bildschirmreset
$FE6F      JSR ($A002)     zum BASIC-Warmstart
```

Vier Befehle – das ist alles? Was passiert hier nun?

- Also zunächst einmal wird die Unterroutine ab $FD15 aufgerufen. Diese setzt 16 Vektoren im RAM wieder in ihren Ursprungszustand zurück, indem sie einfach die 32 (=16x2) normalen Werte aus dem ROM in das RAM ab $0314 schreibt. Sie sehen also, daß auch der IRQ- und BRK-Vektor zurückgestellt werden (dort befinden sich ebenso einige Vektoren für die Ein- / Ausgabe, von dene später noch die Rede sein wird).
- Als nächstes kommt eine Routine ab $FDA3 dran. Sie setzt die Werte aller Bausteine zurück. Der VIC und der SID werden so initialisiert und ebenso die beiden CIAs. Dadurch wird auch der Systeminterrupt vorbereitet (Timer mit dem richtigen Wert laden, Interruptquelle festlegen etc.).
- Jetzt kommt noch die Routine ab $E518 zum Zuge. Sie löscht ganz einfach den Bildschirm und füllt das Farb-RAM mit der aktuellen Zeichenfarbe wieder auf (diese Routine wird auch benutzt, wenn die SHIFT-CLR/HOME Tastenkombination drücken.
- Zum Schluß müssen wir noch in einen definierten Zustand kommen. Dies wird mit dem indirekten Sprung über $A002 getan. Dort, am Anfang des Basic-ROMs steht der Vektor, der auf eine Routine zeigt (bei $E37B), die das BASIC des C64 startet und in die Eingabewartescheife verzweigt.

Somit wurde also ein kleiner RESET durchgeführt, was wir auch einem „Warmstart“ nennen (im Gegensatz zum Kaltstart, der den „richtigen“ RESET bezeichnet, weil er also noch „kalt“ ist, aktiv wird).

So. Dies soll für heute genügen. Nächsten Monat werden wir uns dann einmal in den System-IRQ „einklinken“ und eigene IRQs parallel zu ihm laufen lassen. Ebenso werdne wir ihn auch einmal ausschalten und einen reinen User-IRQ programmieren. Bis dahin wünsche ich Ihnen wie immer allzeit „Gut Hack“.

(ub)

Teil 3 – Magic Disk 01/91

Hallo zusammen zum dritten Teil des CIA-Kurses. Diesen Monat geht's ran an die Buletten, wir wollen uns endlich einmal um die konkrete Programmierung der CIAs und somit von Interrupts kümmern! Letztes Mal hatten wir ja den System-IRQ behandelt, dessen Funktionsaufbau wir heute brauchen werden. Ich werde Ihnen anhand eines Beispielprogramms einmal zeigen, wie wir den System-IRQ für uns benutzen können. Also los geht's…
Wie Sie nun ja wissen, läuft im C64 im Normalfall ja schon ein IRQ, der Systeminterrupt nämlich. Er wird 60 Mal pro Sekunde aufgerufen und arbeitet eine Jobroutine im Betriebssystem-ROM ab, die gewisse interne Aufgaben (die wir im letzten Monat ja schon besprochen hatten) abarbeitet. Wollen wir einen eigenen IRQ schreiben, so ist die einfachste

Methode hierfür ein "Einklinken" in den System-IRQ. Es hat den Vorteil, daß wir uns (wenn es sich um zyklisch wiederkehrende Aufgaben handelt) nicht noch umständlich um das Programmieren eines CIA-Timers kümmern müssen, sondern einfach die vorgegebene Timerprogrammierung übernehmen.
Sie erinnern sich ja bestimmt noch daran, daß beim Auftreten eines IRQ-Ereignisses, der Prozessor über einen Vektor in $FFFE/$FFFF auf eine kleine Jobroutine verzweigt, die feststellt, ob der Interrupt, der aufgetreten ist, ein IRQ oder ein BRK-Interrupt war. Diese Routine verzweigte dann wiederum über zwei Vektoren im RAM auf verschiedene Jobroutinen für die beiden Interrupts.
Diese waren:

$0314/$0315 (dez. 788/789) für den IRQ
$0316/$0317 (dez. 790/791) für den BRK

Wollen wir also, daß der Prozessor jetzt auf eine eigene Routine verzweigt, dann müssen wir einfach den entsprechenden Vektor hier dahingehend verändern (im Fachjargon spricht man auch von "verbiegen"), daß er anschließend auf unsere eigene IRQ bzw. BRK-Routine zeigt. Ich habe Ihnen, wie schon erwähnt, einmal ein kleines Beispiel vorbereitet, das dies verdeutlichen soll. Das Problem das ich lösen wollte, war folgendes:
Stellen Sie sich vor, Sie programmierten gerade eine Anwendung und Sie wollten, daß Ihr Programm von Zeit zu Zeit Fehler- oder Benutzungshinweise auf dem Bildschirm ausgibt. Damit das ganze auch noch optisch gut ins Auge fällt, wäre es angebracht, daß diese Mitteilung längere Zeit aufblinkt, so daß sie dem Benutzer buchstäblich "ins Gesicht springt". Dies ist eine Aufgabe, die sich hervorragend über einen IRQ lösen läßt. Es hat sogar zusätzlich noch den Vorteil, daß das Hauptprogramm vollkommen unberührt von dem wäre, was da angezeigt werden soll. Es genügt also, eine Interrupt-Routine zu aktivieren, die dann so ganz nebenher zum Beispiel die Mitteilung "Das Programm rechnet!" ausgibt, während das Hauptprogramm tatsächlich gerade mit irgendeiner Berechnung beschäftigt ist. Wollen wir uns ansehen, wie man eine solche Routine nun realisiert. Zunächst einmal brauchen wir natürlich eine eigene IRQ-Routine. Sie soll nachher die Nachricht auf dem Bildschirm ausgeben. Ich habe mich da einmal auf die letzte Bildschirmzeile festgelegt. Das ist ein Randbereich, den man gut nutzen kann. Zur eigentlichen Textausgabe brauchen wir zwei kleine Unterroutinen - eine, die den Text schreibt, und eine die ihn wieder löscht, damit wir somit ein Blinken erzeugen. Der Einfachheit halber, habe ich mich dazu entschieden, den auszugebenden Text im Bildschirmcode irgendwo im Speicher abzulegen. Dann genügt es nämlich (im Gegensatz zum ASCII-Code), den Text einfach in den Bildschirmspeicher einzukopieren, was durch eine kleine, aber feine Schleife sehr schnell erledigt wird.
Der Aufbau dieser Routine verlangt es mitunter auch, daß das letzte Zeichen des Textes den binären Wert 0 hat. Im Bildschirmcode ist dies der Klammeraffe ("@"). Zum Löschen der Mitteilungszeile genügt es, diese mit dem Bildschirmcode für "SPACE" aufzufüllen. Das wäre gleich dem Vorgang, wenn Sie mit dem Cursor in die unterste Zeile des Bildschirms fahren und nun 40 Mal die SPACE-Taste drücken würden. Der Bildschirmcode für das Zeichen SPACE ist 32($20).
Hier nun also die beiden Routinen, die diese Aufgaben übernehmen. DOMSG gibt den Bildschirmtext aus, und BLANK löscht die Mitteilungszeile. Zu DOMSG sei noch zu sagen, daß die Anfangsadresse des auszugebenden Textes, vorher schon von der Interrupt-Initialisierungsroutine in die beiden Adressen nach dem LDA-Befehl geschrieben wurde. Das Programm hat sich also selbst verändert. Dies ist (für uns) die einfachste und sinnvollste Lösung, die man in einem solchen Fall anwendet. Zu jener Initialisierungsroutine kommen wir später. Ich möchte übrigens darauf hinweisen, daß alle hier erwähnten Routinen und Programme mit dem HYPRA-ASS- Assembler aus der Computerzeitschrift "64' er" erstellt wurden.
Leser, die das Eingabeformat und die Bedienungsweise dieses Assemblers kennen, können sich also glücklich schätzen.

Trotzdem werde ich die auftauchenden Assembler-Besonderheiten hier erklären, damit Sie die Programme auch mit jedem anderen Assembler eingeben können. Sie sollten auf jeden Fall wissen, daß in diesem Assembler anstatt absoluter Sprungadressen sogenannte "Labels" benutzt werden. Das sind Sprungmarken, die einfacher zu handhaben sind, da man so nur auf einen Namen springen muß, dessen absolute Adresse der Assembler berechnet. Assemblerprogrammierer sollten sich aber sowieso mit Labels auskennen, da sie heutzutage in jedem Assembler Verwendung finden.

```
DOMSG     LDY #00        Y-Reg. als Zeiger initialisieren.
LOOP1     LDA $C000,Y    Zeichen holen (Anfangsadresse Text plus Y-Offset).
          BEQ L3         War das letzte Zeichen gleich 0 (= ), dann ENDE.
          STA $07C0,Y    Ansonsten Zeichen in Bildschirmspeicher schreiben.
          INY            Zähler erhöhen.
          BNE LOOP1      Unbedingter Sprung.

BLANK     LDY #39        Y-Reg. als Zeiger initialisieren (39+1=40 Zeichen füllen
          LDA #32        Bildschirmcode für SPACE in Akku holen.
          STA $07C0,Y    Akku in Bildschirmspeicher entleeren.
          DEY            Zähler erniedrigen.
          BPL LOOP2      Solange wiederh., bis 0 unterschritten wird (Y ist dann negativ).
          RTS            Und Tschüß!
```

Die Anfangsadresse der Mitteilungszeile (im Folgenden MSG-Zeile; MSG = Message = Mitteilung) ist logischerweise die Adresse des ersten Zeichens in der 25. und letzten Zeile des Bildschirms. Sie errechnet sich aus der Basisadresse des Bildschirmspeichers (normalerweise=1024) plus der Zeilenanzahl-1 multipliziert mit 40. Da unsere Zeilenanzahl 25 ist, lautet die Rechnung für uns:

1024+(25-1)*40=1024+24*40=1984 (=$07C0)

Was der Adresse in unseren beiden Routinen entspricht! Des Weiteren muß ich noch auf eine Besonderheit in DOMSG hinweisen. Die Schleife wird erst dann verlassen, wenn das zuletzt gelesene Zeichen 0 ist (wir Erinnern uns - das ist die Endmarkierung). Über BEQ springen wir dann auf den RTS-Befehl der BLANK-Routine. Der Branch-Befehl BNE am Ende der Routine ist ein sogenannter unbedingter Sprung. Die hier abgefragte Bedingung ist immer erfüllt, weil das vorher inkrementierte Y-Register nie die 0 erreichen wird, da die maximale Zeichenanzahl ja 40 ist.
Mehr wäre unsinnig, denn man sähe diese Zeichen ja gar nicht auf dem Bildschirm. Versierte Assembler-Programmierer kennen diese Art des Springens. Sie hat den Vorteil der Speicherersparnis (ein JMP-Befehl belegt immer 3 Bytes, ein Branch-Befehl, wie BNE nur 2) und ist zusätzlich ganz sinnvoll, wenn man Routinen relokatibel halten möchte. Verschiebt man ein Assembler-Programm im Speicher, so ist es nur dann auch an anderer Stelle lauffähig, wenn keine absoluten Zugriffe auf programminterne Adressen stattfinden. Der JMP-Befehl springt ja immer absolut und somit auf die alte Adresse. Der BNE-Befehl jedoch ist relativ adressiert. Er merkt sich nur um wieviele Bytes im Speicher er nach vorne, oder nach hinten springen muß.
Wird der Sprungbereich eines Branchbefehls nicht überschritten (+127 und -128 Bytes vom Befehl selbst entfernt), so ist der Einsatz von unbedingten Sprüngen sehr sinnvoll (insofern möglich, also wenn man über den Inhalt eines bestimmten Prozessorflags eine eindeutige Aussage machen kann).
Doch zurück zu unserer Message-Ausgabe-Routine. Außer den beiden Routinen zur Textausgabe, brauchen wir auch die IRQ-Routine selbst, die den Aufruf dieser beiden Routinen steuert.
Legen wir nun also einmal fest, daß der Message-Text jeweils einmal pro Sekunde blinken soll. Das heißt im Klartext, daß wir zunächst einmal den Text auf den Bildschirm schreiben müssen, wo er eine halbe Sekunde stehen bleibt und ihn anschließend wieder für eine halbe Sekunde

löschen. Dieser Vorgang wiederholt sich nun beliebig oft. Damit das Ganze aber auch irgendwann einmal ein Ende hat, müssen wir nach einer bestimmten Anzahl von Blinkzyklen die Interruptroutine auch wieder abschalten.
Da immer alle halbe Sekunde eine der beiden Ausgaben getätigt werden soll, müssen wir so logischerweise alle 30 Interrupts eine solche tätigen (wir wollen ja den Systeminterrupt benutzen, der wie gesagt 60 Mal pro Sekunde auftritt).
Für all diese Funktionen brauchen wir insgesamt 3 verschiedene Zwischenspeicher:

1. Eine Speicherzelle, die als Interruptzähler dienen soll. Sie zählt 30 Interrupts mit und veranlaßt dann die IRQ-Routine eine der beiden Ausgaben zu tätigen. Dieser Speicherzelle wollen wir den schlichten Namen COUNTER (="Zähler") geben.
2. Damit die IRQ-Routine auch immer weiß, ob sie nun Text ausgeben, oder Text löschen soll, brauchen wir auch noch eine Speicherzelle, in der vermerkt ist, welcher Ausgabemodus als nächstes benötigt wird. Ich nenne diese Speicherzelle einmal MSGMODE.
3. Zuletzt brauchen wir noch eine Speicherzelle, die mitzählt, wie oft der Text nun geblinkt hat. Ist eine ge- wisse Anzahl erreicht, so kann die IRQ-Routine in eine Routine verzweigen, die diese selbst beendet. Diese Adresse habe ich PULSE genannt.

Die drei soeben erwähnten Namen werden alle in dem nun folgenden Source-Listing für die IRQ-Routine verwendet, so daß Sie jetzt also über deren Verwendungszweck informiert sind. Ich habe diesen Namen natürlich auch schon absolute Adressen zugewiesen. HYPRA-ASS benutzt dafür den Pseudo-Opcode ".EQ" für "is EQual"(="ist gleich"). Schauen Sie sich hierzu auch einmal den Sourcecode zu MSGOUT an (auf der Vorderseite dieser MD als "MSGOUT-ROM. SRC").
Die Zwischenspeicher belegen in der oben angezeigten Reihenfolge die Adressen $FB, $FC, $FD. Diese sind allesamt aus der Zeropage, und (wie die fleißigen Assembler-Programmierer unter Ihnen sicher wissen) vom Betriebssystem nicht genutzt, weshalb wir sie für unsere Zwecke verwenden können.
Jetzt brauchen wir eine Initialisierungsroutine, die die neue IRQ-Routine in den System-IRQ einbindet und sie somit im System installiert. Das hier ist sie (Source-Listing auf dieser MD steht sie ganz am Anfang und heißt MS-GOUT):

```
SEI              Alle weiteren IRQs sperren (wegen, des Verbiegens der Vekoren)
STA PULSE        Im Akku steht die Blinkanzahl; also merken wir sie uns.
STX LOOP1+1      LO-Byte der Anfangsadresse des Textes wird in DOMSG eingesetzt
                 (LOOP1 ist ein Label davon, siehe oben).
STY LOOP1+2      Dasselbe mit dem HI-Byte.

LDX #<(IRQ)      LO-Byte der Anfangsadr. der neuen IRQ-Routine laden (siehe unten).
LDY #>(IRQ)      HI-Byte laden.
STX $0314        LO-Byte des Vektors auf unsere Routine ausrichten.
STY $0315        HI-Byte des Vektor auf unsere Routine ausrichten.

LDA #01          Initialisierungswert in Akku laden.
STA COUNTER      Zählregister damit initialisieren.
STA MSGMODE      Mode-Register damit initialisieren.
CLI              Alle Voreinstellungen getätigt; wir können den IRQ wieder freigeben.
RTS              Und Tschüß!
```

Die Routine ist in 3 Abschnitte gegliedert. Im ersten Abschnitt werden zunächst die Aufruf-Parameter gemerkt. Diese sind:

- Die Anzahl der Blinkvorgänge steht im Akku.
- Die Anfangsadresse des auszugebenden Textes steht in LO/ HI-Byte-Darstellung in X- und Y-Register.

Der SEI-Befehl am Anfang ist sehr wichtig.Wir müssen nämlich davon ausgehen, daß gerade dann ein Interrupt auftreten könnte, wenn das LO-Byte der neuen IRQ-Adresse schon gesetzt

ist, das HI-Byte jedoch noch nicht. Dann zeigt der Vektor irgendwo in den Speicher hinein. Tritt jetzt ein IRQ auf, dann springt der Prozessor in die Pampas und verabschiedet sich meistens danach. Um dem vorzubeugen, muß man einfach alle IRQs verhindern, was ja mit dem SEI-Befehl erzielt wird. Der Prozessor ignoriert jetzt die Tatsache, daß da die CIA1 gerade einen Interrupt meldet und wir können in Ruhe den Vektor neu setzen. Diese Aufgabe erledigt der zweite Teil unserer Routine. Die Anfangsadresse der neuen IRQ-Routine wird in X und Y-Register geholt und in die Speicheradressen unseres Zeigers geschrieben. Im dritten und letzten Teil initialisieren wir noch zwei der drei Variablen (PULSE wurde durch das Speichern am Anfang der Routine schon gesetzt). Den COUNTER laden wir mit 1, damit gleich beim nächsten Interrupt eine Ausgabe erfolgt (siehe auch unten). MSGOUT kann zwei verschiedene Zustände haben. Entweder es steht dort 0, dann soll bei der nächsten Ausgabe der Text gedruckt werden, oder wir haben dort eine 1, dann soll die MSG-Zeile gelöscht werden. Ich initialisiere hier mit 1, damit beim ersten Aufruf die Zeile erst einmal gelöscht wird. Würden wir zuerst den Text schreiben, könnte es uns passieren, daß in den verbleibenden Zeichen (vorausgesetzt der Text ist weniger als 40 Zeichen lang) noch alter "Zeichenmüll" im Bildschirmspeicher steht.
Kommen wir nun endlich zur Interruptrountine selbst. Hier einmal das Listing:

```
IRQ     DEC COUNTER       Zähler herunterzählen.
        BEQ L1            Wenn Zähler=0, dann verzweigen auf Ausgabe.
        JMP SYSIRQ        Sonst springen wir auf den System-IRQ.

L1      LDA MSGMODE       Welcher Ausgabemodus?
        BNE L2            Ungleich 0, also verzweigen auf "Zeile löschen"!

INC     MSGMODE           Schon mal den MSGMODE auf 1 schalten.
        JSR DOMSG         MSG ausgeben.
        JMP PRP           Und Ausgaberoutine verlassen.

L2      DEC MSGMODE       Schon mal den MSGMODE auf 0 schalten.
        JSR BLANK         Und MSG-Zeile löschen.

        DEC PULSE         Wenn Zeile gelöscht, ist ein Blinkvorgang abgeschlossen. Also
                          Zähler für Blinks erniedrigen.
        BPL PRP           Wenn noch nicht die 0 unterschritten wurde, Ausgaberoutine
                          verlassen.

        LDX #<(SYSIRQ)    Wenn ja, dann haben wir oft genug geblinkt...
        LDY #>(SYSIRQ)    ...also LO/HI-Byte der Adresse vom System-IRQ laden.
        STX $0314         Und den IRQ-Vektor
        STY $0315         ...wieder auf den System-IRQ setzen.
        JMP SYSIRQ        Eigenen IRQ mit Sprung auf System-IRQ beenden.

PRP     LDA #30           Akku laden...
        STA COUNTER       ...und den IRQ-Zähler neu initialisieren.
        JMP SYSIRQ        Auch hier beenden wir den eigenen IRQ mit einem Sprung auf
                          den System-IRQ.
```

Hier die Erklärung:
Nachdem wir die Initialisierungsroutine von vorhin aufgerufen haben, zeigt der IRQ-Vektor jetzt also auf die Routine "IRQ" . Die CIA1 signalisiert nun einen Timerunterlauf in Form eines Signals an den Prozessor. Dieser springt daraufhin auf die Jobroutine ab $FF47, wo die Prozessorregister auf den Stapel gerettet werden und über den IRQ-Vektor unsere Routine angesprungen wird.
Diese erniedrigt nun also den COUNTER und prüft, ob er schon 0 ist. Das ist der Fall, da wir den COUNTER ja mit 1 initialisiert hatten, und er soeben auf 0 abgezählt wurde. Das Programm verzweigt deshalb also auf das Label "L1".
Dort wird jetzt geprüft, welcher Ausgabemodus eingestellt ist. Da MSGMODE auf 1 steht gehts jetzt also gleich weiter zu "L2", wo zunächst einmal MSGMODE auf 0 gezählt wird. Durch einen

Aufruf von "BLANK" wird die MSG-Zeile gelöscht.
Dies heißt für uns auch, daß einmal geblinkt wurde. Also müssen wir jetzt den Zähler für die Anzahl der Blinks um 1 erniedrigen. Gehen wir einmal davon aus, daß wir die Initialisierungs-routine mit einer 10 im Akku aufgerufen hatten. Somit ist der Inhalt von PULSE jetzt, nach dem Herunterzählen 9. Das heißt, daß die 0 noch nicht unterschritten wurde und deshalb wird beim folgenden Branch-Befehl auch gleich auf das Label PRP verzweigt. Dort steht eine kleine Jobroutine, die unseren IRQ wieder beendet.
Der COUNTER wird hier mit 30 neu geladen und das Programm verzweigt anschließend auf den System-IRQ, der nun regulär abgearbeitet wird, und der den Interrupt wieder beendet, indem er die alten Prozessorregister zurückholt und den Prozessor mittels RTI wieder in das alte Programm, das bearbeitet wurde, als der Interrupt auftrat, zurückschickt.
Das Label SYSIRQ beinhaltet also die Sprungadresse des System-IRQs, wie Sie anhand des Source-Codes erkennen können.
Ich habe dort nämlich wieder mittels ".EQ" eine Zuweisung an diesen Labelnamen gemacht. Bei dem folgenden IRQ, zählt unsere Routine wieder den COUNTER um 1 herunter. Diesmal jedoch, ist diese Speicherzelle noch nicht 0, weshalb die Routine auch nicht in die Ausgaberoutine ab "L1" verzweigt, sondern gleich auf den System-IRQ springt. Dies geht nun 30 Interrupts lang so weiter, erst dann gibt es wieder eine 0 im COUNTER. Unsere Routine verzweigt jetzt wieder in die Ausgaberoutine. Dort wird wieder der Ausgabemodus geprüft, der diesmal jedoch 0, also " Text ausgeben" ist. Dort müssen wir jetzt MSGMODE dann auf 1 hoch zählen und dann mittels DOMSG unsere Mitteilung auf dem Bildschirm ausgeben.
Anschließend können wir den Interrupt wieder über den System-IRQ verlassen. Diese Vorgänge werden nun solange wiederholt, bis PULSE die 0 unterschreitet. Dann nämlich wird nicht auf PRP verzweigt, sondern es werden gleich die Befehle hinter dem BPL-Befehl abgearbeitet. Sie setzen den IRQ-Vektor wieder auf den System-IRQ zurück, so daß also unsere eigene Routine nicht mehr angesprungen wird. Ihre Aufgabe ist nun erfüllt. Diesmal brauchen wir das Interrupt-Flag übrigens nicht zu setzen, da innerhalb eines Interrupts dieses Flag ja schon durch den Prozessor gesetzt wurde (letzten Monat hatte ich das ja genauer erklärt). Auch jetzt verzweigen wir wieder auf den System-IRQ um unseren Interrupt zu beenden. Das wäre nun also eine Routine, die in den System-IRQ eingebunden ist. Das Betriebssystem springt sie direkt an, und sie selbst fährt nach ihrer eigenen Arbeit gleich mit dem System-IRQ fort. So daß dieser also auch weiterhin arbeitet. Der Vorteil ist schnell ersichtlich. Laden Sie doch einfach einmal das Programm "MSGOUT. CODE" auf der Vorderseite dieser MD. Es ist ein Assembler-Programm, das mit "SYS 4096*8", oder "SYS 32768" aufgerufen wird. Der MSG-Text "DAS IST EIN IRQ UEBERS BETRIEBS-SYSTEM!" blinkt nun in der untersten Bildschirmzeile. Währenddessen haben wir aber immer noch den Cursor auf dem Bildschirm, den wir auch weiterhin benutzen können. Würden wir unsere IRQ-Routine nicht über den System-IRQ wieder verlassen, wäre das auch nicht der Fall. Dadurch können Sie also während Ihren eigenen IRQs die Tastatur weiterhin verwenden! Kommen wir nun zu einem weiteren Problem. Angenommen, Sie wollten eine Mitteilung ausgeben, während zum Beispiel gerade eine Routine damit beschäftigt ist, im RAM unter dem ROM Daten zu verschieben. In dem Fall können Sie ja nicht mehr über den System-IRQ springen, da das Betriebssystem-ROM ja abgeschaltet wäre. Man kann dies tun, indem man einige Bits im Prozessorport verändert. Dieser wird durch die Speicherzelle $0001 repräsentiert. Dort steht normalerweise der Wert 55 (=$37), was für den Prozessor die Speicherkonfiguration:

- BASIC-ROM bei $ A000-$ BFFF eingeschaltet.
- I/ O-Bereich bei $ D000-$ DFFF (wo auch die Register der beiden CIAs liegen) eingeschalt
- Betriebssystem-ROM bei $ E000-$ FFFF eingeschaltet.

Wollen wir nun auf das RAM unter dem BASIC und dem Betriebssystem-ROM zugreifen, so kann man letztere mit dem Schreiben des Wertes 53 (=$35) in den Prozessorport abschalten. Das I/ O-ROM, das wir ja noch brauchen (wegen der CIA1), bleibt dabei eingeschaltet. Der System-IRQ ist somit nicht mehr für uns vorhanden und ebenso auch nicht die Jobroutine, die

über den IRQ-Vektor $0314/$0315 auf entsprechende IRQ-Routinen springt. In dem Fall müssen wir die Steuerung des Interrupts selbst bewältigen. Das heißt zunächst einmal, daß wir diesmal die Prozessorregister selbst retten müssen (was ja normalerweise die Jobroutine bei $FF47 macht - siehe Teil 2 des CIA-Kurses), und sie auch entsprechend wieder zurückholen müssen. Als IRQ-Vektor zählt jetzt auch nicht mehr der bei $0314/$0315, sondern wir benutzen den Hardware-Vektor direkt. Da das ROM dort ja abgeschaltet ist, können wir also problemlos die Speicherzellen $FFFE/$FFFF mit einem Vektor auf unsere IRQ-Routine beschreiben. Zur Demonstration habe ich Ihnen unsere MSGOUT-Routine einmal umgeschrieben, so daß sie auch ohne Betriebssystem auskommt. Der Source-Code hierzu ist ebenfalls auf dieser MD zu finden, unter dem Namen "MSGOUT-RAM. SRC". Im Prinzip brauchen wir nur ein paar Befehle zu der ROM-Version von MSGOUT hinzuzufügen, um die RAM-Version zu erhalten. Das wichtigste ist hierbei die Initialisierungsroutine, die ich Ihnen hier nun aufführen möchte:

```
                SEI                 Interrupts wie immer speren.
                STA PULSE           Blinkzähler merken.
                STX LOOP1+1         Anfangsadresse des…
                STY LOOP1+2         ... Textes merken.
                LDX #<( IRQ)        Anfangsadresse der neuen…
                LDY #>( IRQ)        ... IRQ-Routine laden.
                STX $FFFE           Und den Hardware-Vektor…
                STY $FFFF           ... darauf ausrichten.
                LDA #$35            Wert für "ROM aus" laden…
                STA $01             ... und ab in Prozessorport.
                LDA #01             Initialisierungswert laden.
                STA COUNTER         Zähler initialisieren.
                STA MSGMODE         Modus initialisieren.
                CLI                 IRQs wieder freigeben.
LOOP3           LDA $01             Prozessorport laden.
                CMP #$37            Vergleiche mit "ROM an".
                BNE LOOP3           Ungleich, also weiter prüfen.
                RTS                 Ansonsten Tschüß!
```

Viel hat sich hier ja nicht geändert. Den ersten Abschnitt kennen wir ja noch von der alten MSGOUT-Routine. Diesmal müssen wir jedoch noch aus einem zweiten Grund die Interrupts sperren. Indem wir nämlich später noch das Betriebssystem-ROM abschalten, nehmen wir dem Prozessor die Grundlage für IRQs. Zum Einen verschwindet somit nämlich der Hardware-Vektor des Betriebssystems, zum Anderen auch alle Jobroutinen für den System-IRQ. Der Prozessor springt dann irgendwo im undefinierten RAM rum und hängt sich dann unweigerlich auf. Also jetzt geht nix mehr ab mit IRQs!

Der zweite Abschnitt ist uns auch nicht so unbekannt. Diesmal setzen wir jedoch nicht den IRQ-Vektor $0314/$0315, sondern den Hardware-Vektor für IRQs bei $FFFE/$FFFF. Das können wir getrost auch bei eingeschaltetem ROM tun (wie das hier der Fall ist), denn die geschriebenen Daten landen auf jedem Fall im RAM, da der Prozessor ins ROM ja nicht schreiben kann. Weil er aber irgendwo hin muß mit seinen Daten, schickt er sie automatisch ins RAM. Nur der Lesezugriff kommt aus dem ROM!

Um auch dies zu ändern, verändern wir im dritten Abschnitt der Initialisierungsroutine dann auch noch den Prozessorport so, daß BASIC und Betriebssystem-ROM abgeschaltet werden. Im vierten Abschnitt werden jetzt noch die variablen Register unserer IRQ-Routine initialisiert. Hier hat sich nichts geändert.

Wichtig ist nun noch der letzte Abschnitt. Wir können nämlich unsere Initialisierungsroutine nicht einfach so verlassen - zumindest nicht in diesem Beispiel. Denn normalerweise, wenn Sie sich im Eingabemodus des 64ers befinden, wird eine Eingabeschleife des BASICs durchlaufen, die ständig auf Eingaben prüft und dann bei entsprechenden BA-SIC- Befehlen, diese aufruft. Wenn Sie also mit SYS unsere IRQ-Routine starten, dann wird die Initialisierungsroutine nach ihrer Arbeit wieder in die BASIC-Eingabeschleife zurückkehren wollen. Die ist jetzt jedoch nicht mehr verfügbar, weil wir ja das BASIC-ROM abgeschaltet haben. Auch hier springt der

Prozessor dann mitten ins leere RAM, verläuft sich dort und stürzt vor lauter Kummer einfach ab. Da ich die IRQ-Routine nun aber so programmiert habe, daß sie automatisch, wenn sie genug geblinkt hat, BASIC und Betriebssystem wieder einschaltet, können wir dies als Kennzeichen dafür nehmen, daß die Grundvoraussetzungen für ein Verlassen der Initialisierungsroutine wieder gegeben sind. Deshalb also, habe ich eine Warteschleife hier eingebaut, die immer nur prüft, ob die ROMs mittlerweile wieder da sind. Erst wenn dieser Fall eintritt, wird zurückgesprungen!
Soviel zur Initialisierung für eine Arbeit unter dem ROM. Kommen wir nun zur Interrupt-Routine selbst. Auch sie muß leicht modifiziert werden. Auch hier will ich einen kurzen Abriß der hinzugefügten Befehle geben:

```
IRQ       PHA            Akku retten
          TXA            X-Reg. in Akku schieben...
          PHA            ...und retten
          TYA            Y-Reg. in Akku schieben...
          PHA            ...und retten.
          ------------
          (etc…)
```

So fängt nun die neue IRQ-Routine an. Anschließend folgen genau die Befehle, die auch in MSGOUT-ROM verwendet wurden. Bis auf einen Unterschied: wenn es nämlich darum geht, den Interrupt wieder abzuschalten, weil wir oft genug geblinkt haben, lautet die Abschaltroutine folgendermaßen:

```
          LDA #$37       Alte Speicherkonfiguration
          STA $01        wieder einschalten.
          JMP SYSIRQ     Und IRQ beenden,
```

Hier wird einfach das ROM wieder eingeschaltet. Ein Zurückbiegen von Vektoren entfällt, da das ROM ja nun wieder da ist, und von nun an der System-IRQ wieder treu seine Dienste leistet, so, als wäre nichts geschehen. Nach dieser Änderung des Prozessorports ist auch die Bedingung der Warteschleife der Initialisierungsroutine erfüllt, womit diese sogleich wieder zum guten alten BASIC zurückspringt.
Eins muß ich jedoch noch hinzufügen. Wie sie ja noch wissen, verzweigt die ganze Routine ja noch öfter auf den System-IRQ, der dann ja gar nicht da ist! Demnach hätte ich diese Verzweigungen, die ich vorhin so leichtfertig übersprungen habe, ja erwähnen müssen! Nun, ich habe dieses Problem anders gelöst. Ich habe nämlich den ".EQ"-Pseudo-Opcode von "HYPRA-ASS", mit dem ich dem Label "SYSIRQ" die Adresse "$EA31" zuwies aus dem Source-Code entfernt, und dafür eine eigene SYSIRQ-Routine geschrieben. Der Name entspricht zwar nicht mehr dem, was vorher die Bedeutung war (SYStem-IRQ), aber so ging es halt am einfachsten. Diese neue Routine tut nun nichts anderes, als den Interrupt ordnungsgemäß zu beenden. Wie wir ja noch aus dem letzten CIA-Kurs wissen, tut dies der System-IRQ am Ende auch. Die entsprechenden Befehle hierzu stehen ab Adresse $ EA7E. Genau die habe ich nun in die neue "IRQ-Beenden"-Routine übernommen:

```
SYSIRQ    LDA $DC0D      ICR von CIA1 löschen.
          PLA            Altes Y-Reg. vom Stapel in Akku holen...
          TAY            ...und zurück in Y-Reg. schieben.
          PLA            Altes X-Reg. vom Stapel in Akku holen...
          TAX            ...und zurück in X-Reg. schieben.
          PLA            Alten Akkuinhalt vom Stapel holen.
          RTI            Und Interrupt beenden.
```

Die Bedeutung dieser Befehle sollte Ihnen ja noch bekannt sein. Zunächst müssen wir weitere IRQs durch Löschen des ICR-Registers der CIA1 wider ermöglichen (dadurch werden ja die Interrupt-Quellen-Flags gelöscht, wie wir aus Teil 1 dieses Kurses noch wissen). Dann holen wir uns in umgekehrter Reihenfolge die Prozessorregister wieder vom Stapel runter, bevor wir den

Interrupt mit RTI beenden.
So. Das war' s dann mal wieder für diesen Monat. Noch einen Hinweis zu den Programmen bezüglich dieses Kurses:

- Die beiden Source-Codes der MSGOUT-Routine können Sie übrigens auch lesen, wenn sie nicht den HYPRA-ASS besitzen. Laden Sie hierzu ein Source-Code- File einfach an den BASIC-Anfang (also mit ",8" am Ende) und geben Sie LIST ein. Jetzt wird der Text zwar nicht automatisch formatiert, so wie HYPRA-ASS das normalerweise tut, aber lesen kann man das ganze schon. Zur Anschauung genügt es zumindest.
- Das File "MSGOUT-CODE" beinhaltet beide Versionen von MSGOUT. Laden Sie es bitte absolut (also mit ",8,1") und starten Sie die einzelnen Routinen mit:
  - SYS 32768 für MSGOUT-ROM
  - SYS 32777 für MSGOUT-RAM

Ich will mich jetzt von Ihnen verabschieden. Nächsten Monat wollen wir uns dann einmal um die Kupplung von Timer A und Timer B einer CIA kümmern und auch noch den BRK-Interrupt behandeln. Bis dahin noch viel Spaß beim Herumprobieren mit IRQs.

(ub)

Teil 4 – Magic Disk 02/91

Herzlich Willkommen zum 4 . Teil unseres CIA-Kurses. Diesen Monat möchte ich dann doch vorgreifen und Ihnen zunächst einmal die NMI-Interrupts erklären. Dann können wir nämlich anhand eines einfachen Beispiels auch eine sehr nützliche Anwendungsweise von Timerkopplung behandeln.
Mit dem IRQ kennen Sie sich mittlerweile ja gut aus. Wir haben diese Interruptart in Zusammenhang mit dem Systeminterrupt ja schon eingehendst kennengelernt und auch schon eigene IRQ-Routinen geschrieben, die sowohl eigenständig, als auch im System-IRQ eingebunden arbeiteten.
Kommen wir nun also auch zu der anderen für uns wichtigen Interruptart, dem NMI. Zunächst: Was ist der Unterschied zwischen einem IRQ und einem NMI? Da haben wir zum einen schon einmal den Unterschied, daß beide Interruptarten von jeweils einem CIA angesteuert werden. Das hatte ich Ihnen ja schon zu einem früheren Zeitpunkt erläutert. CIA2 löst also NMIs aus, CIA1 IRQs. Doch es gibt da noch einen weitgehendst wichtigeren Punkt, in dem sich der NMI vom IRQ unterscheidet. Ich hatte Ihnen damals bei der Erklärung der Hardwareverbindungen der CIAs und des Prozessors untereinander ja schon erklärt, daß jede der CIAs nicht nur verschiedenartige Interrupts auslöst, sondern daß vielmehr der Prozessor über zwei verschiedene Eingänge verfügt, an denen der jeweilige Interrupt ausgelöst werden kann. Das bedeutet aber auch, daß er einen Unterschied zwischen beiden Interruptarten macht, und das ist ganz wichtig für uns zu wissen!
IRQ ist die Abkürzung für " Interrupt-ReQuest", was soviel bedeutet, wie "Anfrage auf eine Unterbrechung". Das Wort "Anfrage" möchte ich hier ganz deutlich herausstellen, denn wie Sie mittlerweile ja ebenfalls wissen sollten, können wir den Prozessor durch den SEI-Befehl dahingehend manipulieren, daß er Signale am IRQ-Eingang ignoriert. Im Fachjargon sagt man auch, man kann einen Interrupt " maskieren"- durch Setzen des Interruptflags können wir also softwaremäßig IRQs sperren und das ist dann auch der Punkt, bei dem der Unterschied zum NMI in Erscheinung tritt." NMI" ist nämlich ebenfalls eine Abkürzung und steht für "Non-Maskable- Interrupt", was mit "Nichtmaskierbare-Unterbrechung" den Nagel auf den Kopf trifft. Und schon hätten wir das Kind im Brunnen. NMIs sind softwaremäßig nicht sperrbar und das kann enorme Vorteile gegenüber dem IRQ haben!
Hier einmal ein einfaches Beispiel: die Routinen des Betriebssystems müssen in der Regel aus dem einen oder anderen Grund von Zeit zu Zeit IRQs verhindern.
Zum Einen aus Zeitersparnis und somit zur Geschwindigkeitssteigerung, zum Anderen bei

komplizierten Synchronisationsvorgängen mit der Peripherie des Computers, wobei auftretende Interrupts Zeitwerte verfälschen und somit stören könnten, benutzt das Betriebssystem nun ebenfalls den SEI-Befehl. Die Folge des Ganzen wird schnell klar: soll der IRQ nun ganz zeitkritische Arbeiten erledigen, so kommt er schnell aus dem Takt und ist somit oft viel zu ungenau. Glänzendes Beispiel ist die BASIC-Uhr TI$.
Sie wird nämlich über den System-IRQ gesteuert, der ja normalerweise 60 Mal pro Sekunde auftritt. Rein theoretisch braucht die Routine für TI$ also nur bis 60 zu zählen, um zu wissen, daß jetzt eine Sekunde verstrichen ist. Praktisch sieht es aber so aus, daß zum Beispiel die Ein-/ Ausgaberoutinen des Betriebssystems oft den IRQ unterbinden. Es genügt also, ein längeres Programm von Diskette zu laden um die TI$- Uhr extrem zu bremsen, so daß sie die eine oder andere Sekunde nachgeht. Aus den Sekunden werden Minuten, je mehr man lädt und irgendwann kann man die Zeitwerte der Uhr komplett vergessen: dadurch, daß IRQs zwischendurch nicht mehr auftreten können, aber die Zeit weiterhin unerbittlich verstreicht, zählt die TI$- Routine zwar weiterhin 60 IRQs, diese jedoch dauerten länger als eine Sekunde.
NMIs hingegen werden IMMER bearbeitet, sobald sie auftreten. Sogar dann, wenn sich der Prozessor gerade innerhalb eines IRQs befindet. Er rettet dann einfach die Daten des IRQs (Programmzeiger, Prozessorstatus etc.) und bearbeitet den NMI. Umgekehrt jedoch, kann kein IRQ während eines NMIs auftreten, da der Prozessor ja dann das Interruptflag ja schon von selbst gesetzt hat (sie erinnern sich...). Es sei denn wir lassen dies ausdrücklich zu, indem wir innerhalb der NMI-Routine das Flag durch CLI wieder löschen.
Sie sehen also, man muß immer Unterschiede machen, wofür ein Interrupt benötigt wird. Einfache Probleme lassen sich schnell mit dem IRQ bewältigen (und das ist bei den meisten der Fall), da er bei Bedarf auch sehr einfach abgeschaltet werden kann. Bei zeitkritischen Problemen benutzt man besser einen NMI. Er funktioniert genau und zuverlässig, wobei man allerdings in Kauf nehmen muß, daß man diesen nicht so einfach wieder verhindern kann.
Das ist nämlich der Grund warum man bei der Programmierung eines NMIs mehr Aufwand hat. Für ihn existiert, ebenso wie für den IRQ, auch ein Vektor, der verändert werden muß, wenn man die NMIs auf eigene Interruptroutinen umleiten will.
Wir hatten ja letzten Monat schon gelernt, daß man dabei sichergehen muß, daß während dieser Veränderung in gar keinem Fall ein Interrupt ausgelöst werden darf, da so schon während das LO-Byte, jedoch noch nicht das HI-Byte des Vektors verändert ist, der Rechner unkontrolliert in die Pampas springen könnte, was so unangenehme Folgen hätte, wie zum Beispiel einen Rechnerabsturz.
Aus diesem Grund müssen wir zusehen, daß alle eventuell in Frage kommenden NMI-Quellen so geschaltet sind, daß sie keinen Interrupt auslösen, während wir den NMI-Vektor verändern.
Im Normalfall ist dieses Problem eigentlich relativ einfach zu handhaben, denn das Betriebssystem benutzt den Timer-NMI ausschließlich nur bei Betrieb der RS232- Schnittstelle, also bei der seriellen Datenübertragung per Modem. In aller Regel können wir diesen Fall jedoch ausklammern und davon ausgehen, daß alle Funktionen der CIA2, die den NMI betreffen, funktionslos ihr Dasein fristen. Nur im Falle einer eigenen Benutzung sollten wir uns immer im Klaren darüber sein, was für eine Aufgabe der NMI gerade behandelt und wie sie gesteuert wird. Im Regelfall genügt es jedoch, einfach alle Bits des ICR-Registers der CIA2 zu löschen, so daß von dort keine Interrupts mehr an den Prozessor gelangen. Dies geschieht durch ein Schreiben des Wertes 127 (=$7F) in selbiges Register ($DD0D = dez.56589).
Eine weitere Besonderheit des NMIs ist, daß die RESTORE-Taste hardwaremäßig DIREKT an die NMI-Leitung des Prozessors angeschlossen ist, daß also auch von dort Interrupts ausgelöst werden können.
Dieses macht sich das Betriebssystem zunutze, denn bei einem Druck auf RUN/-STOP-RESTORE, was den C64 ja wieder in einen einigermaßen definierten Zustand zurückbringt, wird immer ein NMI ausgelöst. Was nun allerdings wirklich da- bei geschieht und wie es mit Sprungvektoren für NMIs aussieht, wollen wir uns jetzt einmal näher anschauen.
Dazu ist wieder einmal eine kleine Reise in die tieferen Gefilde des Betriebssystems angesagt.

Beginnen wir mit den elementaren Grundvoraussetzungen:

- Zunächst also wird ein NMI ausgelöst, indem der Benutzer auf die RESTORE-Taste drückt.
- Der Prozessor hält seine momentane Arbeit jetzt unverzüglich an, rettet wie bei jedem Interrupt die wichtigsten Daten auf den Stapel (das hatten wir ja schon), setzt das Interruptflag, so daß ihn keine IRQs mehr stören können und macht sich daran, wieder in einem eigenen Vektor für NMIs, am Ende seines Adressbereichs nachzuschauen, wo er jetzt weiterfahren soll. Dieser Vektor liegt bei $FFFA/$FFFB und zeigt auf eine Jobroutine des Betriebssystems bei $FE43.

Schauen wir uns einmal an, was dort so läuft:

$FE43	SEI	IRQs sperren.
$FE44	JMP ($0318)	Über NMI-Vektor springen.

Da hätten wir auch schon den angesprochenen NMI-Vektor, der für uns veränderbar ist. Er belegt die Speicherstellen $0318/$0319 (dez.792/793) und zeigt normalerweise auf die Adresse gleich hinter der soeben aufgelisteten Routine, auf $ FE47.
Was übrigens anzumerken ist, ist die Tatsache, daß wir beim Einbinden von eigenen NMIs in den System-NMI darauf achten müssen, daß wir auch die Prozessorregister quasi "von Hand" auf den Stapel retten müssen. Die IRQ-Vorbereitungsroutine hatte dies ja noch VOR dem Sprung über den RAM-Vektor gemacht, weshalb wir uns nicht mehr darum kümmern mußten. Beim NMI macht das Betriebssystem das erst NACH dem Sprung über den Vektor, in der nun folgenden Routine:

----------------		NMI vorbereiten.
$FE47	PHA	Akku auf Stapel.
$FE48	TXA	X-Register nach Akku...
$FE49	PHA	...und auf Stapel.
$FE4A	TYA	Y nach Akku...
$FE4B	PHA	...und auf Stapel.
$FE4C	LDY #$7F	Wert laden...
$FE4E	STA $DD0D	...und damit alle NMI-Quellen von der CIA2 kommend sperren.
----------------		Auf RS232-Betrieb prüfen.
$FE51	LDY $DD0D	Interruptquellen-Anzeige aus ICR lesen und somit löschen um weitere NMIs freizugeben.
$FE54	BMI $FE72	Wenn die RS232-Schnittstelle aktiv ist, verzweigen.

Anmerkung: Mit dem letzten Befehl wurde abgefragt, ob eines der Interruptquellenbits des ICR gesetzt ist. Da das Betriebssystem ja CIA2- gesteuerte NMIs nur dann benutzt, wenn die RS232 Schnittstelle läuft, genügt es, nur zu prüfen, ob der NMI überhaupt von der CIA2 kommt. In diesem Fall ist Bit 7 des ICR auf 1. Wenn das nicht der Fall ist, dann kann der Auslöser nur die RESTORE-Taste gewesen sein, und es wird wie folgt fortgefahren:

----------------		Auf ROM-Modul prüfen
$FE56	JSR $FD02	Prüft ob ein ROM-Modul im Expansions-Port steckt.
$FE59	BNE $FE5E	Wenn nein, dann ist das Zero-Flag gelöscht und wir überspringen den folgenden Befehl.
$FE5B	JMP ($8002)	Ja, wir haben ein Modul, also springen wir auf den Modul-NMI (siehe unten).
----------------		Prüfen, ob R-S/RESTORE
$FE5E	JSR $F6BC	Flag für STOP-Taste in der Zeropage ($91 = dez. 145) berechnen und setzen.
$FE61	JSR $FFE1	STOP-Taste abfragen.
$FE64	BNE $FE72	Wenn nicht gedrückt verzweigen, um den NMI zu beenden.
----------------		RUN-STOP/RESTORE ausführen
$FE66	JSR $FD15	Standard-Vektoren für Interrupts und Ein-/ Ausgabevektoren initialisieren.

```
$FE69       JSR $FDA3      Ein-/Ausgabebausteine initialisieren.
$FE6C       JSR $E518      Bildschirm löschen.
$FE6F       JMP ($A002)    BASIC-Warmstart ausführen.
----------------
```

Der Teil von $FE47-$FE59 rettet nun zunächst einmal die drei Prozessorregister auf den Stapel. Des Weiteren werden alle Interruptquellen die der CIA2 geben könnte, gesperrt. Dann, von $FE51-$FE55 wird das ICR der CIA2 ausgelesen und somit für neue Interrupts freigeben (ist im Moment zwar nicht möglich, da wir ja die Interrupts vorher sperrten, wird aber für die RS232-Behandlung gebraucht!). Gleichzeitig wird geprüft, ob der Befehl von der CIA2 kam, und wenn ja zur RS232- Unterroutine verzweigt.
Im nun folgenden Teil von $FE56-$FE5D wird geprüft, ob ein ROM-Modul im Expansionsport steckt. Wie so etwas funktioniert, will ich Ihnen nächsten Monat erklären. Wenn ein Modul da ist, dann wird auf einen moduleigenen NMI verzweigt, andernfalls wissen wir nun endgültig, daß der Benutzer wahrscheinlich den Computer zurücksetzen will, und wir können in den folgenden Teil verzweigen.
Dieser geht von $FE5E-$FE65 und prüft nach, ob die RUN/ STOP-Taste gleichzeitig auch noch gedrückt ist. Hierzu wird eine Unterroutine ab $F6BC benutzt, die direkt die Tastatur abfragt und in Speicherzelle $91 der Zeropage anzeigt, ob die RUN/ STOP-Taste gedrückt ist. Steht dort eine 0, so war dies der Fall. Andernfalls verzweigt das Programm nun doch in die RS232-Routine. Ehrlich gesagt, weiß ich nicht warum dies so ist, denn es läge näher, den NMI direkt zu beenden, aber die Wege des C64- Betriebssystems sind manchmal halt auch unergründlich…
Jetzt sind wir aber endlich im letzten Teil angelangt. RUN/STOP-RESTORE wurde gedrückt, was heißt, daß wir einen "Mini- Reset" ausführen sollen. Es werden nun drei Unterroutinen aufgerufen, die die wichtigsten Voreinstellungen im System vornehmen, nämlich das Zurücksetzen der Sprungvektoren von (inclusive) $0314-$333 (Routine ab $FD15), das Rücksetzen des Grafikchips (VIC) und des Soundchips (SID), sowie der CIAs (Routine ab $FDA3) und das Löschen des Bildschirms (Routine ab $E518). Zum Schluß wird dann mit einem indirekten Sprung auf den NMI-BASIC-Warmstart in den"READY"-Modus des BASICs verzweigt. Hierbei wird der NMI nicht wie üblich mit RTI beendet, sondern die Warmstartroutine setzt einfach den Stackpointer wieder zurück und springt dann in die BASIC-Eingabe-Warteschleife.
Soviel zum System-NMI. Wir werden in den nächsten Kursteilen auch noch auf die RS232-Schnittstelle und deren Bedienung, sowie auf die ROM-Modul- Behandlung näher eingehen, weshalb ich diese Themen diesmal aussparen möchte. Kommen wir nun zu der Programmierung von NMIs. Die Anwendungsbereiche für diese Interruptart sind vielfältig. Sie läßt sich gut bei zeitkritischen Problemen einsetzen, wie zum Beispiel das zyklische Lesen von Daten, in einer fest vorgeschriebenen Geschwindigkeit (Digitizer und Scanner arbeiten oft mit NMIs) . Am eindrucksvollsten ist aber bestimmt das Abspielen von Musik über den NMI. Viele Sound-Editoren benutzen ja schon häufig die Möglichkeit, ein Musikstück via Interrupt spielen zu lassen, jedoch gehen diese meist über den IRQ. Ich habe Ihnen einmal ein Beispielprogramm auf dieser MD mit abgespeichert. Es heißt "NMI/ IRQ-DEMO" und beinhaltet drei kleine Unterprogramme. Das Programm tut nichts anderes, als einen Interrupt zu initialisieren, der ständig einen Ton über Stimme 1 des SID spielt. Hierbei wird bei jedem Aufruf das HI-Byte der Tonfrequenz um 1 erhöht. Das Ergebnis ist ein ganz lustiger Soundeffekt, der nicht unähnlich dem Geräusch ist, das entsteht, wenn Scotty die Besatzung der "Enterprise" rumbeamt.
Eigentliche Aufgabe dieses Beispielprogramms ist nun aber, Ihnen den Unterschied zwischen IRQ und NMI zu verdeutlichen. Deshalb gibt es zwei Möglichkeiten, es aufzurufen. Zum Einen können Sie es mit "SYS 4096*9" starten. Dann initialisieren Sie einen NMI. Der Ton wird ständig über den NMI ausgegeben. Nun bietet sich zusätzlich noch die Möglichkeit, daß Sie durch "SYS 4096*9+3" eine kleine Unterroutine aufrufen, die alle IRQs sperrt. Zu erkennen ist dies daran, daß nach dem Aufruf der Cursor nicht mehr blinkt. Trotzdem aber hören Sie weiterhin den

Soundeffekt - dies also als Beweis, daß der NMI unabhängig vom IRQ arbeitet.
Die zweite Möglichkeit den Effekt zu starten ist die mit "SYS 4096*9+6" . Sie initialisieren dann einen IRQ, der jedoch genau dasselbe tut wie der NMI zuvor. Sie können nun noch einmal mit "SYS 4096*9+3" die IRQs sperren, und schon hören Sie nichts mehr.
Als Beispiel zu den Problematiken, diesich mit dem IRQ und dem Betriebssystem ergeben, empfehle ich Ihnen, während der IRQ läuft einmal ein Programm von Diskette zu laden. Sie werden merken, daß die Tonausgabe zwischenzeitlich desöfteren stockt. Wenn das passiert, dann hat gerade wieder einmal eine Routine des Betriebssystems den IRQ mittels SEI abgeschaltet. Leider können Sie dieses Problem nicht mit einem laufenden NMI untersuchen. Der stört nämlich dann die anfangs schon erwähnten Synchronisationsvorgänge, die beim Laden benötigt werden, wobei der 64 er nur Mist anstellt. Probieren können Sie es einmal. Manchmal hat man Glück, manchmal nicht. Bitte laden Sie nun den zweiten Teil des IRQ-Kurses aus dem Kurs-Menü. IRQ-Kurs Teil 4 .2 Wollen wir uns nun einmal anschauen, wie unser NMI-Programm aufgebaut ist. Die NMIs werden übrigens über Timer A der CIA2 ausgelöst, der denselben Wert wie der Timer des System-IRQs als Startwert bekommt (das wäre der Wert 16420=$4024). Hier also das Programm:

```
----------      NMI vorbereiten
LDA $7F         "NMI-Quellen sperren" laden,
STA $DD0D       und in ICR von CIA2.
LDX #$2B        Zeiger auf eigene...
LDY #$90        ...Routinen laden,
STX $0318       und NMI-Vektor...
STY $0319       ...setzen.
LDX #$24        Timerwert LO-Byte.
LDY #$40        Timerwert HI-Byte.
STX $DD04       In TALO und...
STY $DD05       ...in TAHI schreiben.
LDA #$81        Wert laden...
STA $DD0D       Timer A als Interruptquelle festlegen.
STA $DD0E       Timer A starten.
----------      SID einstellen.
LDA #$0F        Wert 15 für volle Lautstärke
STA $D418       ... ins Lautstärkeregister.
LDA #$00        Zählregister für Tonfrequenz
STA $02                  initialisieren.
RTS             Und zurück.
----------      NMI-Routine
CLI             IRQs wieder freigeben.
PHA             Akku,
TXA             X-,
PHA
TYA             und Y-Register auf Stapel
PHA             retten.
LDA #16         Wert für "Dreieckswelle aus"
STA $D404       ... in SID schreiben
LDA $02                  Zähler für Frequenz lesen…
STA $D401       und in Frequenz-HI schreiben
INC $02         Zähler um 1 erhöhen.
LDA #$17        Wert für "Dreieckswelle an"
STA $D404       ... in SID schreiben.
LDA $DD0D       NMIs wieder freigeben
PLA             Akku,
TAY             X-,
PLA
TAX             und Y-Register wieder vom
PLA             Stapel zurückholen.
RTI             Und Interrupt verlassen
```

Im ersten Teil dieses Listings haben wir die Initialisierungsroutine für unseren NMI. Hier werden zunächst auf die schon beschriebene Art und Weise alle Interruptquellen die von der CIA2 kommen, gesperrt. Anschließend wird der NMI-Vektor bei $0318/$0319 auf unsere eigene Routine verbogen (die Routine beginnt bei $9000 im Speicher, weshalb die eigentliche Interruptroutine bei $902B beginnt).
Ist dies getan, müssen wir als nächstes den Timerwert in die Timerregister für Timer A laden (wie bei CIA1 sind dies die Register 4 und 5- TALO und TAHI).
Dies ist der wie oben schon beschriebene Wert $4024. Jetzt müssen wir nur noch den Timer A als NMI-Interruptquelle setzen und ihn anschließend starten. Dies geschieht in den folgenden 3 Zeilen. Was der Wert $81 für Register 13(ICR) und 14(CRA) bedeutet wissen Sie ja schon aus Teil 1 dieses Kurses, als ich Ihnen die Funktionen dieser Register genauer erläutert habe. Zum Abschluß der Initialisierungsroutine müssen wir auch noch den SID darauf vorbereiten, Sound auszugeben. Dazu haben wir auch noch genug Zeit, da der schon laufende Timer zum nächsten Interrupt noch lange genug zählen wird (ich hätte die SID-Initialisierung auch vorher anbringen können). Also wird erst einmal die Lautstärke des Soundchips eingeschaltet, sowie den Anfangsfrequenzwert für unseren Soundeffekt in Adresse $02 in der Zeropage geschrieben. Diese Adresse wird als Zählregister benutzt, da man auf die Register des SID leider nicht zum Lesen zugreifen kann. Somit sind sie also auch nicht mittels INC hochzählbar. Nun sind alle Voreinstellungen getätigt, und wir können wieder zum aufrufenden Programm zurückverzweigen.
Im zweiten Teil des Listings sehen Sie nun die NMi-Routine selbst. Als erstes erlauben wir hier wieder das Auftreten von IRQs (sie erinnern sich, das Betriebssystem hatte sie ja gesperrt). Nun werden nach der mittlerweile schon altbekannten Methode die Prozessorregister auf den Stapel gerettet, was wir bei NMIs ja IMMER von Hand machen müssen, da das Betriebssystem uns diese Arbeit leider nicht abnimmt.
Nun kommt der Teil, in dem der nächste Ton gespielt wird. Hierzu wird erst einmal die Stimme 1 des SID abgeschaltet. Dies ist notwendig, weil wir keine Hüllkurve vorher festgelegt hatten, die einen Ton möglicherweise dauerhaft spielen würde. Deshalb befinden sich in den Hüllkurvenregistern die Werte 0, was bedeutet, daß ein Ton nur ganz kurz angeschlagen wird und gleich wieder verstummt. Damit man aber die nächste Frequenz nun hört müssen wir die Stimme also erst noch ausschalten. Anschließend wird der Inhalt des Zählregisters $02 in das HI-Byte- Frequenzregister von Stimme 1 geschrieben ($ D401), und der Zähler für den nächsten Interrupt um 1 erhöht. Nun schalten wir Stimme 1 wieder an, und zwar mit einer Dreieckswellenform - der Ton wird nun gespielt.
Die Arbeit des NMIs ist getan, machen wir uns also daran, den Interrupt zu beenden. Ebenso altbekannt werden also das ICR wieder freigegeben, die Prozessorregister zurückgeholt und mittels RTI der NMI beendet.
So. Nun wissen Sie also alles wissenswerte über NMIs. Bis auf einige kleine Ausnahmen, können Sie diese Interruptart genauso behandeln, wie einen IRQ. Da die CIAs ja baugleich sind, fällt die CIA gesteuterte Programmierung von NMIs ja ebenso aus, wie beim IRQ. Ein ebenfalls ganz interessantes Anwendungsgebiet von NMIs ist die Steuerung von gewissen Funktionen über einen Druck auf die RESTORE-Taste. Ich habe dies einmal bei einem Apfelmännchenprogramm benutzt. Diese Programme berechnen ja bekanntermaßen Grafiken aus der Mandelbrotmenge, die zwar ganz ansehnlich sind, deren Berechnung jedoch oft Stunden, wenn nicht sogar Tage dauern kann. Ich wollte nun eben jenes Programm beschleunigen, indem ich den Bildschirm abschalte. Wie Sie vielleicht wissen, kann durch diese Maßnahme eine Geschwindigkeitssteigerung von 5% erzielt werden, da der VIC bei abgeschaltetem Bildschirm nicht mehr auf den Speicher des Computers zugreifen muß, um die Daten für Grafiken, Zeichen, Sprites und ähnliches zu holen. Dadurch stört er den Prozessor nicht mehr beim Zugriff, wodurch dieser schneller arbeiten kann. Das Problem war nun jedoch, daß ich weiterhin sehen wollte, wie weit der Rechner nun mit der Grafikberechnung fortgefahren

ist. Mit einer einfachen NMI-Routine war dies möglich. Ohne noch zeitraubend die Tastatur abzufragen, habe ich einfach einen neuen NMI "eingekoppelt", der nichts anderes tut, als den Bildschirm aus-, bzw. einzuschalten, wenn man die RESTORE- Taste drückt. Dieser Trick läßt sich vielfältig anwenden und ist einfach zu programmieren, hier das kleine Programm:

```
               -----------------    Initialisierung
MAIN           LDA #$7F             CIA2-NMIs...
               STA CIA2+13          sperren.
               LDX #<(NMI)          NMI-RAM-Vektor
               LDY #>(NMI)          ...auf eigene
               STX $0318            ...NMI-Routine
               STY $0319            ...verbiegen.
               RTS                  Tschüß!
               -----------------    NMI-Routine
NMI            PHA                  Akku retten.
               LDA $D011            Register laden,
               EOR #16              Bildschirmbit invertieren.
               STA $D011            Und wieder speichern.
               PLA                  Akku zurückholen.
               RTI                  NMI-Ende.
               -----------------
```

Das ist tatsächlich alles! Das Programm ist im "Hypra-Ass"- Quellcode angegeben, dessen Sonderfunktionen ich Ihnen letzten Monat ja schon erklärte. Hier eine Dokumentation:
Im ersten Teil wird zunächst einmal die CIA2 als Interruptquelle gesperrt. Dies ist nicht unbedingt notwendig, da sie sowieso ausgeschaltet sein sollte, jedoch habe ich es hier zur Sicherheit einmal gemacht. Des Weiteren wird der NMI-Vektor auf unseren eigenen NMI verbogen, und die Initialisierung ist beendet.
Nun zum zweiten Teil: Zunächst einmal rette ich hier nur den Akku. X- und Y-Register werden in der NMI-Routine sowieso nicht benutzt, weshalb wir sie nicht unbedingt auch noch retten müssen. Als nächstes laden wir den Inhalt von Register 17 des VIC ($ D011= dez.53265) in den Akku, da mit dem 4 . Bit dieses Registers der Bildschirm ein und ausgeschaltet wird. Der Inhalt dieses Registers wird nun einfach mit dem Wert des 4 . Bits geEORt. Dabei wird der Wert des Bits immer invertiert. Ist es 1 (= Bildschirm an), so wird es nach dem EOR-Befehl 0 (= Bildschirm aus) sein und umgekehrt. Der neue Wert muß nun nur noch wieder in $D011 zurückwandern, und wir können den NMI beenden.
Das Programm finden Sie übrigens auch auf dieser MD unter dem Namen "NMI-SCREENOFF". Es muß absolut (",8,1") geladen werden und wird mit SYS 49152 gestartet. Ab dann können Sie per Tastendruck auf RESTORE den Bildschirm nach Belieben ein- und ausschalten. Das war es dann man wieder für diesen Monat. Ich wünsche Ihnen noch viel Spaß beim herumexperimentieren mit den NMIs und seien Sie nicht enttäuscht, wenn's mal nicht auf Anhieb klappen sollte, denn: Wer noch nie einen Rechnerabsturz erlebt hat, ist kein wahrer Programmierer!
In diesem Sinne bis nächsten Monat,

Ihr Uli Basters (ub).

Teil 5 – Magic Disk 03/91

Hallo zusammen, zum 5 . Teil dieses Kurses. Nachdem Sie nun ja ausgiebig über Interrupts Bescheid wissen, wollen wir uns diesen Monat noch einmal ein wenig intensiver um die CIA-Bausteine ansich kümmern, denen dieser Kurs ja gewidmet ist.
Diesmal wollen wir nämlich das Thema der Timerkopplung behandeln. Darunter versteht man die Verkettung von Timer A und Timer B einer CIA zu einem großen 32- Bit Timer (je ein Timer verfügt ja über je 16 Bit).
Wozu das gut ist, werden Sie spätestens dann gemerkt haben, als Sie einmal einen Timer-

Interrupt programmieren wollten, der weniger oft als 15 mal pro Sekunde auftritt. Dann reicht nämlich ein 16- Bit-Timer nicht mehr aus, insofern er Systemtakte zählt, was ja eigentlich die häufigste Anwendung der Timer-Triggerung ist (Triggerung gibt den auslösenden Faktor an, der den Timer dazu veranlaßt, den Wert, den er beinhaltet um 1 zu erniedrigen - ich erwähnte Timer-Trigger schon zu Anfang dieses Kurses).
Sie können in einen 16-Bit-Timer ja einen maximalen Wert von 2↑16-1=65535 laden. Bei 985248 .4 Taktzyklen, die der 64 er pro Sekunde bekommt, heißt das also, daß der Timer genau 985248 .4/65535=15 .03392691 mal pro Sekunde unterlaufen kann, wenn er am langsamsten läuft.
Langsamer (oder besser: weniger häufig) geht es nicht. Zu diesem Zweck besteht nun aber auch die Möglichkeit Timer A und Timer B einer CIA zu koppeln. Über Timer B hatten wir bisher ja wenig gesprochen, da er vom Betriebssystem sowohl in CIA1, als auch in CIA2 nicht benutzt wird. Jedoch ist es ebenso möglich ihn als Timer zu verwenden, wobei analog zu Timer A vorgegangen wird.
Nun jedoch zu jener Kopplung. Es gibt nämlich eine Möglichkeit, mit der wir Timer B anstelle von Systemtakten die Unterläufe von Timer A zählen lassen können. Das heißt also, daß jedesmal, wenn Timer A bei 0 angekommen ist, Timer B um 1 erniedrigt wird. Schaltet man nun einen Interrupt so, daß er dann von einer CIA ausgelöst wird, wenn Timer B unterläuft, so hat man einen vollen 32- Bit-Zähler, mit dem wir schon ganz andere Dimensionen in Sachen Häufigkeit von Unterläufen erreichen können. Mit 32 Bit können wir nämlich maximal 2↑32-1=42949672995 (in Worten: über zweiundvierzigmilliarden) Werte zählen, was bedeutet, daß wir auch dementsprechend lange Pausen zwischen zwei Timerinterrupts haben. Mal kurz durchgerechnet sind das alle 42949672955/985248.2=4359 .273556 Sekunden. Das sind mehr als72 Minuten, also eine ganze Menge!
Die dabei anfallenden Interrupts werden dann von Timer B ausgelöst, weshalb wir dann auch darauf achten müssen, das wir ihn als Interruptquelle im ICR (Register 13) setzen.
Kommen wir nun zu einer Anwendung. Ich muß gestehen, viele Möglichkeiten hierzu bieten sich mir nicht, jedoch könnte eine Timerkopplung durchaus zur Lösung des einen oder anderen speziellen Problems nützlich sein. Ich habe mir da eine ganz sinnvolle Anwendung einfallen lassen und Ihnen gleich einmal ein Beispielprogramm vorbereitet, anhand dessen ich Ihnen die Timerkopplung erläutern möchte. Es ist ein kleines Programm, das den Takzyklenverbrauch eines anderen Programms stoppen kann. Es heißt EVAL und ist auf dieser MD in zwei Versionen gespeichert. Zum einen habe ich da den ausführbaren Code, den Sie absolut laden müssen (mit ",8,1") und der ab Adresse $9000 (dez.36864) gestartet wird. Hierzu jedoch später mehr. Des Weiteren finden Sie auch noch den Quell-Code von EVAL unter dem Namen "EVAL.SRC". Er ist wie immer im Hypra-Ass-Format und kann auch ohne HYPRA-ASS mit ",8" zum Anschauen in den Basicspeicher geladen werden.
Doch nun zu EVAL selbst. Zunächst einmal wollen wir uns fragen, was nun genau geleistet werden soll. EVAL soll zunächst einmal ganz einfach die Taktzyklen zählen, die ein anderes Programm verbraucht. Das ist die Problemstellung.
Die Lösung wollen wir - na, Sie werden es nicht glauben über die Timer einer CIA bewerkstelligen. Ich habe zu diesem Zweck die CIA2 ausgesucht, deren Timer normalerweise, solange die RS232- Schnittstelle des C64 nicht genutzt wird, unbenutzt sind. Zur Ermittlung der verstrichenen Zeiteinheiten, sprich Taktzyklen, müssen wir nun einfach nur einen bestimmten Grundwert in beide Timer laden, sie starten und anschließend das zu prüfende Programm aufrufen. Dies wollen wir mittels eines "JSR"-Befehls tun. Springt das aufgerufene Programm nun zurück, so müssen wir den Timer direkt anhalten und anschließend den in ihm enthaltenen Wert von unserem Anfangswert subtrahieren.
Dadurch erhalten wir die Anzahl der Taktzyklen, die verstrichen sind, zwischen Start und Stop des Timers. Soviel zum theoretischen Programmablauf von EVAL. Kommen wir nun zu den Timern selbst. Zunächst müssen wir zusehen, daß wir eine richtige Triggerung für Timer A und Timer B wählen. Timer B soll ja die Unterläufe von Timer A zählen und dieser widerum die

Systemtakte. Zu diesem Zweck schreiben wir also erst einmal den Wert $81 in das Control-Register von Timer A (=CRA, Reg.14), wie wir das ja auch schon von der Interruptprogrammierung her kennen. Weil bei diesem Wert Bit 5 gelöscht ist zählt Timer A also Systemtakte. Für Timer B wird das schon schwieriger. Ich hatte Ihnen ja schon einmal bei der Beschreibung der CIA-Register aufgelistet, welche Möglichkeiten es hier gibt. Timer B kann nämlich in Gegensatz zu Timer A vier (anstelle von zweien) verschiedene Triggerquellen haben. Dies wird von den Bits 5 und 6 gesteuert, deren Kombinationen ich Ihnen noch einmal auflisten möchte:

Bit 5	6	Timer B zählt
0	0	Systemtakte
0	1	Steigende CNT-Flanken
1	0	Unterläude von Timer A
1	1	Unterläufe von Timer A, wenn CNT=1 ist.

Für uns kommt da die Kombination "10" in Frage. Bit 5 ist also gesetzt und alle anderen gelöscht. Wie bei Timer A müssen wir jedoch auch Bit 0 setzen, weil wir beim Laden dieses Wertes in das Control-Register von Timer B (CRB, Reg.15) den Timer auch gleich starten wollen. Demnach brauchen wir diesmal den Wert $41.
Das war dann auch schon alles, was wir zur Timerkopplung brauchen. Timer A zählt nun Systemtakte und löst bei jedem Unterlauf ein Herabzählen von Timer B aus. Einen Interrupt wollen wir diesmal nicht erzeugen, doch könnte man auch durchaus im ICR festlegen, das einer erzeugt werden soll, wenn Timer B dann unterläuft.
Somit hätten wir also einen 32-Bit Timer der Systemtakte zählt. Die Reihenfolge der LO/ HI-Zählbytes sieht nun folgendermaßen aus:

TimerB-HI TimerB-LO TimerA-HI TimerA-LO

Sie müssen also nicht nur ein High- und Lowbytepaar berechnen, sondern gleich zwei. Hier wechselt man dann auch in die nächsthöhere Ebene der "Bits und Bytes". Eine 16- Bit-Zahl bezeichnet man nämlich als ein "Word" (engl.: word = Wort) und eine 32-Bit-Binärzahl als ein "Longword" (engl.: Langwort). Um eine Dezimalzahl nun in ein Longword umzuwandeln müssen Sie folgendermaßen vorgehen:

1. Zunächst teilen wir unsere Zahl durch 2↑16(=65536) und nehmen den Ganzzahlanteil des Ergebnisses als höherwertiges Word. Dieses wird nun wie gewohnt in Low- und Highbyte aufgespalten.
2. Nun multiplizieren wir das High-Word mit 2↑16 und subtrahieren den Wert von unserem Anfangswert. Das Ergebnis was wir hier erhalten ist das niederwertige Word, das ebenfalls, wie gewohnt, in Low- und Highbyte umgewandelt wird.

Als Beispiel habe ich einmal die Zahl 2000000 in ein Langword umgewandelt:

1. 2000000 / 65536 = 30.51757813
   1a. HI-Word=30 --> LO=30, HI=0
2. 2000000 – 30 x 65536 = 33920
   2a. LO-Word=33920 --> LO=128, HI=132

   Longword: 0 30 132 128 Binär: 00000000 00011110 10000100 10000000

Soviel also hierzu. Nun können wir also schon einmal beliebig lange 32-Bit-Timerwerte berechnen. Für EVAL habe ich übrigens nicht irgendeine Zahl genommen, sondern schlichtweg

die größte (also die 42 Milliarden von oben). Länger als 72 Minuten sollte eine zu testende Assemblerroutine sowieso nicht sein.
Kommen wir nun also zu der Programmierung von EVAL. Hier möchte ich Ihnen einmal den Anfang des Programms auflisten:

```
              -------------------
EVAL          LDA #$7F            Zunächst alle Inter-
              STA CIA2+13         rupts sperren.
              LDA #00             Und...
              STA CIA2+14         Timer A und
              STA CIA2+15         Timer B anhalten.
              -------------------
              LDA #$FF            Nun den Maximalwert
              STA CIA2+4          in Timer A
              STA CIA2+5          und
              STA CIA2+6          in Timer B
              STA CIA2+7          schreiben.
              -------------------
              LDA #$41            Timer B soll Timer A
              STA CIA2+15         zählen
              LDA #$81            Timer A soll System-
              STA CIA2+14         takte zählen.
              -------------------
              JSR $C000           Testprogramm aufrufen
              -------------------
```

Im ersten Teil dieses Listings sperren wir zunächst einmal alle Interruptquellen durch Löschen des ICR. Anschließend werden durch das Schreiben von 0 in die Control-Register der Timer selbige gestoppt. Diese Maßnahmen sind nur für den Fall gedacht, daß in diesen Timern ganz gegen unsrer Erwartung doch etwas laufen sollte. Zum korrekten Ablauf von EVAL ist es absolut notwendig, daß die Timer stehen, da sonst die Testwerte verfälscht würden.
Anschließend werden die Register TALO, TAHI, TBLO, TBHI mit dem Wert 255 (=$FF) geladen und so auf den Maximalwert 2↑32-1 gesetzt.
Im dritten Teil des Listings werden nun die beiden Timer wieder gestartet. Hierbei MUß Timer B unbedingt VOR Timer A aktiviert werden, da er zum Einen von diesem abhängig ist, und deshalb zuerst aktiv sein sollte (würde Timer A nämlich unterlaufen BEVOR Timer B zählbereit ist, wäre das Testergebnis ebenfalls nicht in Ordnung), und zum Anderen müssen wir so spät wie möglich den 32-Bit-Timer starten, damit auch tatsächlich die Taktzyklen der zu messenden Routine gezählt werden und nicht etwa noch einige Taktzyklen die EVAL benötigt. Im letzten Teil wird nun noch die zu testende Routine aufgerufen, die ich hier einmal bei $C000 (dez.49152) angesiedelt habe.
Jetzt ist also unser 32-Bit-Timer aktiv und zählt schön brav von 4294967295 Richtung 0 hinab. Währenddessen läuft unser Testprogramm ab und jeder Taktzyklus der dabei verstreicht wird mitgezählt.
Wird das Programm dann mittels "RTS" beendet, so kehrt der Rechner wieder in EVAL zurück. Nun müssen wir uns um die weitere Behandlung der Timer kümmern. Damit wieder keine unnötige Rechnerzeit mitgezählt wird stoppen wir also zunächst beide Timer:

```
              -------------------
              LDA #$80            Wert für "Timer-Stop"
              STA CIA2+14         Timer A und
              STA CIA2+15         Timer B anhalten.
              -------------------
```

Da der unser 32- Bit-Timer nun also nur während des Ablaufs des zu testenden Programms lief, müsste nun in den Timerregistern logischerweise haargenau die Anzahl der verbrauchten Taktzyklen stehen, jedoch in einem invertierten Format. Das heißt, daß dadurch, daß der Timer

rückwärts lief, die Taktzyklenanzahl mit folgender Formel berechnet werden muß:

(2↑32-1)-(Longword in Timerregistern)= tatsächlich verbrauchte Taktzyklen.

Na aber holla, das könnte ja schwer werden, die 4-Byte Werte voneinander zu subtrahieren! Doch keine Panik, auch das läßt sich ganz einfach lösen. Dadurch nämlich, daß wir beim Start den absoluten Maximalwert in die Timer geladen hatten, können wir diese Umrechnung durch schlichtes invertieren der einzelnen Bytes vornehmen. Dabei wird das sogenannte Einerkomplement unserer Zahl gebildet, was ebenfalls eine Art ist, Bytes voneinander zu subtrahieren. Zum Beweis möchte ich Ihnen hier einmal ein Beipiel geben: Wir hatten den Startwert $FF $FF $FF $FF im Timer stehen. Eine Invertierung dieses Werts mittels des "EOR"-Befehls ergäbe dann: $00 $00 $00 $00 - mit anderen Worten, der Startwert war 0. Angenommen, nun war der Wert, den wir nach Ablauf eines Testprogramms in den Timern vorfanden folgender: $FF $FD $03 $AE. Dies entspricht dem Dezimalwert 4294771630. Subtrahieren wir diesen Wert von 2↑32-1, so ergibt sich folgendes Ergebnis:195665. Das wäre also die Anzahl der Taktzyklen, die das aufgerufene Programm verbrauchte. Nun wollen wir doch testweise einfach einmal das Einerkomplement der Ergebniszahl bilden (ebenfalls mittels " EOR"):

$FF $FD $03 $AE --> $00 $02 $FC $51.

Dieses Longword umgerechnet ergibt nun aber ebenfalls die Zahl 195665 - "quod erat demonstandum!" Kommen wir nun also zum nächsten Teil von EVAL, dem Retten der Timerwerte und dem gleichzeitigen Umwandeln in die absolute Zahl an Taktzyklen:

```
                -------------------
                LDY #$03            Quellzähler initialisieren.
                LDX #$00            Zielzähler initialisieren.
    LOOP1       LDA CIA2+4,Y        Akku mit dem hintersten Timerregister laden
                EOR #$FF            Komplement bilden
                STA $62,X           und umgekehrt sichern.
                INX                 Zielzähler +1.
                DEY                 Quellzähler -1.
                BPL LOOP1           Wenn noch nicht untergelaufen, dann wiederholen.
                -------------------
```

Diese Schleife nimmt sich nun nacheinander die CIA-Register 7,6,5 und 4 vor, bildet ihr Komplement und speichert sie umgekehrt in die Zeropageadressen $62, $63, $64 und $65. Damit hätten wir dann auch gleichzeitig das Problem gelöst, daß die Bytes ja in der Reihenfolge TBHI TBLO TAHI TALO aufeinander folgen müssen, damit Sie einen Sinn ergeben. Das hat aber auch noch einen anderen, weitaus wichtigeren Grund:
Ich habe die Adressen $62-$65 nämlich nicht etwa willkürlich als Zwischenspeicher gewählt, sondern in diesen Adressen befinden sich nämlich die Mantissenbytes des Floatingpoint-ACcumualtors (kurz: "FAC") des 64ers. Damit wir unsere 32- Bit-Zahl nämlich auch richtig in dezimaler Schreibweise ausgeben können, brauchen wir nämlich den FAC. Der FAC ist ein bestimmter Speicherbereich in der Zeropage der vom Betriebssystem dazu genutzt wird, Fließkommazahlen aufzunehmen und mit ihnen herumzurechnen. Ebenso gibt es auch eine Routine des Betriebssystems, mit deren Hilfe wir die 32- Bit-Zahl nachher in Dezimalschreibweise umwandeln, um sie ausgeben zu können. Dazu jedoch später. Zunächst einmal hätten wir also die absolute Anzahl der vergangenen Taktzyklen, die zwischen Start und Stop des Timers verstrichen sind im FAC stehen. Leider haben wir nun aber doch noch ei- nige Taktzyklen, die durch EVAL verbraucht wurden, in unserer Rechnung. WO? Werden Sie jetzt fragen, na ganz einfach - schauen wir uns noch einmal die Zeilen vor und hinter dem Aufruf des Testprogramms an:

```
                ...
                LDA #$81            Timer A soll System-
```

```
        STA CIA2+14     takte zählen.
        ------------------
        JSR $C000       Testprgramm aufrufen.
        ------------------
        LDA #$80        Wert für " Timer-Stop"
        STA CIA2+14     Timer A und
        STA CIA2+15     Timer B anhalten.
        ------------------
        ...
```

Na? Wissen Sie wo? Natürlich! Ab dem Befehl "STA CIA2+14" war der Timer aktiv. Der anschließende JSR-Befehl gehört nun aber noch nicht in die zu testende Routine, wurde aber trotzdem mitgezählt.
Ebenso wie die Befehle "LDA #$80" und "STA CIA2+14" . Erst dann war der Timer wieder aus. Die hier erwähnten Befehle wurden also alle mitgezählt. Deshalb müssen wir nun noch von unserem Gesamtergebnis die Anzahl der Takte die sie verbrauchten subtrahieren. Ein JSR-Befehl braucht immer 6 Taktzyklen, das direkte Laden des Akkus 2 und das absolute Entleeren des Akkus 4. Demnach haben wir also 6+2+4=12 Taktzyklen zuviel, die nun von der folgenden Subtraktionsroutine von dem Gesamtwert subtrahiert werden:

```
        ------------------
        SEC             Carry setzen.
        SBC #12         Vom Akku (=TALO, in $65) 12 subtrahieren.
        BCS L1          Wenn kein Unterlauf, dann auf Ende verzweigen.
        DEC $64         Sonst nächsthöheres  Byte -1.
        LDX $64         Prüfen, ob dieses
        CPX #$FF        untergelaufen ist,
        BNE L1          Nö, also zum Ende.
        DEC $63         Ja, also nächsthöheres Byte -1
        LDX $63         Prüfen, ob dieses
        CPX #$FF        untergelaufen ist,
        BNE L1          Nö, also zum Ende.
        DEC $62         Sonst, das höchste Byte -1 (ohne Prüfung, weil es nicht
                        unterlaufen kann!)
L1      STA $65         Alles klar, also Akku im niedrigsten Byte sichern.
        ------------------
```

Ich erspare mir hier einen Kommentar, weil Sie bestimmt schon wissen, wie man zwei Integerzahlen in Assembler voneinander subtrahiert. Ich will nur noch darauf hinweisen, daß am Anfang dieses Programmteils der Inhalt von Speicherzelle $65 ja immer noch im Akku steht, da die Komplementierungsschleife von vorhin ihn dort noch hat stehen lassen …
So. Nun hätten wir also die tatsächliche Anzahl der Taktzyklen als Longword im FAC stehen. Nun müssen wir sie nur noch ausgeben. Hierzu möchte ich eine Routine des Betriebssystems benutzen, die den Inhalt des FAC in ASCII-Code umwandelt und ihn anschließend ab $0100 ablegt. Nach dem letzten ASCII-Zeichen fügt sie noch ein Nullbyte ein, was später von Bedeutung sein wird. Zunächst haben wir noch ein kleines Problem - die erwähnte Umwandlungsroutine ("FACTOASC" ist übrigens ihr Name), verlangt nämlich reine Floating-Point- Zahlen im FAC. Diese haben ein eigenes Format, nämlich das MFLPT-Format. Leider entspricht unser 32- Bit-Integer aber noch nicht diesem Format, weshalb wir ihn erst "normieren" müssen. Was dabei passiert, und wie das MFLPT-Format aussieht möchte ich hier auslassen, weil es den Rahmen dieses Kurses sprengen würde. Geben Sie sich einfach zufrieden damit, daß der nun folgende Programmabschnitt von EVAL einfach eine 32- Bit-Zahl im FAC normiert:

```
        ----------------
        LDA #$A0        Exponent initialisie-
        STA $61         ren.
LOOP2   LDA #$62        Höchstes Byte laden, und prüfen, ob höchstes Bit gesetzt.
```

```
            BMI L2          Ja, also fertig!
            DEC $61         Nein, also die ganze
            ASL $65         Mantisse um 1
            ROL $64         nach
            ROL $63         links
            ROL $62         rollen.
            BCC LOOP2  Wenn Carrybit gelöscht, wiederholen
            -----------------
L2          …               Sonst weiter ...
```

So. Jetzt können wir aber endlich die Taktzyklenanzahl ins ASCII-Format umwandeln, also rufen wir FACTOASC auf. Die Einsprungsadresse dieser Routine lautet übrigens:$BDDD (dez.48605). Nun haben wir also den ASCII-Text unserer Zahl im Speicher ab $0100 stehen. Jetzt müssen wir ihn nur noch auf den Bildschirm bringen. Dies geschieht mittels der Routine STROUT. Sie ist ebenfalls eine Betriebssystemroutine und steht ab $AB1E. Als Parameter müssen wir ihr die Anfangsadresse des auszugebenden Textes in LO/ HI im Akku und im Y-Register übergeben. Das Textende erkennt sie an einem Nullbyte am Ende des Textes, das FACTOASC ja freundlicherweise schon eingefügt hat. Kommen wir nun also zum letzten Teil von EVAL:

```
            --------------------
L2          JSR FACTOASC    Erst FAC in ASCII umwandeln.
            LDA #<(TXT1)    Jetzt geben wir zu-
            LDY #>(TXT1)    nächst den Text "BENOETIGTE TAKTZYKLEN:"
            JSR TXTOUT      aus.
            LDA #$00        Nun kommt die Takt-
            LDY #$01        zyklenanzahl auf den
            JSR TXTOUT      Bildschirm.
            LDA #$13        um Schluß Cursor in
            JMP BSOUT       die nächste Zeile setzen.
            --------------------
TXT1        .TX "BENOETIGTE TAKTZYKLEN:"
            .BY 0
            --------------------
```

So. Das wars. Die letzten beiden Befehle am Ende des Listings geben nun noch ein "Carriage Return" aus, damit der Cursor nicht am Ende der Taktzyklenzahl stehenbleibt. Mit dem JMP auf BSOUT (bei $FFD2) beenden wir dann EVAL. Der Rechner kehrt direkt von dort in den Eingabemodus zurück.
Damit sind wir dann auch wieder am Ende eines CIA-Kurses angelangt. Ich hoffe, Sie kennen sich jetzt mit der Timerkopplung aus. Wenn Sie wollen, können Sie ja einmal versuchen, einen Interrupt von einem 32-Bit-Timer auslösen zu lassen, das Know-How sollten Sie aus den vorhergehenden Kursteilen schon haben.
In diesem Sinne möchte ich mich nun bis nächsten Monat von Ihnen verabschieden, wenn wir uns einmal die Echtzeituhren in den CIAs anschauen wollen.
Bis dann,

Ihr Uli Basters (ub).

PS: Zum Test von EVAL laden Sie den Assemblercode bitte absolut in den Speicher und sorgen Sie dafür, daß ihr Testprogramm bei Adresse $C000 beginnt. Nun rufen Sie EVAL einfach mit "SYS 36864" auf.

Teil 6 – Magic Disk 04/91

Willkommen zum 6. Teil unseres CIA-Kurses. Diesmal wollen wir, wie letzten Monat schon versprochen, die Echtzeituhren der CIAs programmieren. Hierbei handelt es sich pro CIA um je

eine 24- Stunden-Uhr, die weitaus genauer als die von BASIC bekannte TI$-Uhr gehen. Sie werden nämlich über die Frequenz des Wechselstroms der aus der Steckdose kommt getriggert. Da diese Frequenz in der Regel immer 50 Hertz beträgt (dafür sorgt das Elektrizitätswerk, das ihn in das Stromnetz einspeist), gehen die Uhren der CIA so gut wie nie falsch! Zudem haben wir die Möglichkeit, eine Alarmzeit zu programmieren. Stimmt irgendwann die aktuelle Uhrzeit mit der angegebenen Alarmzeit überein, so löst die jeweilige CIA einen Interrupt aus. Dieser wird uns im ICR dann auch getrennt als Alarm-Interrupt angezeigt. Auch dieses Mal habe ich Ihnen bezüglich unseres Schwerpunktes ein Programm geschrieben, daß sich auf dieser MD befindet. Es heißt "CLOCK" und wird mit RUN gestartet. Des Weiteren finden Sie wie immer auch den Source-Code von CLOCK unter dem Namen "CLOCK.SRC" im Hypra-Ass-Format auf dieser Diskette. Mit einem Laden an den BASIC-Anfang (",8") können Sie ihn sich mit LIST anschauen.
Kommen wir nun jedoch erst einmal zur Theorie. Wie programmiert man denn einen CIA-Timer?
Das gestaltet sich eigentlich als relativ einfach. In den Registern 8-11 einer jeden CIA werden Zehntelsekunden, Sekunden, Minuten und Stunden abgelegt. Aus diesen Registern kann ebenso die aktuelle Uhrzeit ausgelesen werden. Dennoch gibt es einige Besonderheiten, die wir beachten müssen. Hier zunächst einmal eine Auflistung der besagten Register mit ihrer Funktion:

Register	Funktion
8	Zehntelsekunden
9	Sekunden
10	Minuten
11	Stunden

Erfreulicherweise sind die Uhren der CIAs so ausgelegt, daß sie im BCD-Format arbeiten. Dies ist ein spezielles Darstellungsformat für Zahlen, das uns die Umwandlung der Ziffern für eine Bildschirmausgabe erheblich vereinfacht. BCD steht für "Binary Coded Decimal", was übersetzt "Binär kodierte Dezimalzahl" bedeutet. Dieser Code ist unter anderem auch deshalb so vorteilhaft, weil der Prozessor des C64, der 6510, es uns erlaubt, mit diesen Zahlen zu rechnen. Vielleicht kennen Sie ja die Assemblerbefehle SED und CLD. Mit ihnen kann man das Dezimal-Flag des Prozessorstatusregisters setzen oder löschen, um dem 6510 mitzuteilen, daß man nun mit BCD-Zahlen rechnen möchte.
Ich will Ihnen das einmal anhand einiger Beispiele erläutern. Kommen wir zunächst zu dem Zahlenformat selbst. Im BCD-Format wird ein Byte nicht wie sonst anhand seiner gesetzten, oder gelöschten Bits codiert, sondern ein Byte wird in zwei 4- Bit Bereiche aufgespalten. Einen solchen 4- Bit Abschnitt bezeichnet man im Fachjargon als Nibble. Hier einmal eine keine Verdeutlichung:

Nibble1 Nibble2 10010011

Nibble1 ist hierbei das **höherwertige**, Nibble2 das **niederwertige** Nibble eines Bytes. Im BCD-Format stellt nun jedes Nibble eine Zahl zwischen 0 und 9 dar, die dem normalen Binärformat entspricht(deshalb auch "binär kodiert"). Hier einmal eine Tabelle mit den Werten für die Ziffern:

Wert	Binär
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Sie sehen, die Werte entsprechen also denselben Werten die sie im Binärformat haben. Die Besonderheit des BCD-Formates ist nun, daß, wie oben schon erwähnt, jedes Nibble eines Bytes eine Dezimalziffer kodiert. Die Binärzahl aus dem obigen Beispiel kann also folgendermaßen interpretiert werden:

Binär: 1001 0011

BCD-Format: 9 3 = 93

Dezimal umgewandelt: 147

Das höherwertige Nibble der Zahl hat den Binärwert 9, das niederwertige 3, weshalb die BCD-Zahl 93 lautet! So kann nun jede Zahl zwischen 0 und 99 kodiert werden. Die sonst 256 verschiedenen Binärzahlen werden auf 100 Kombinationen reduziert. Zahlen wie 11001010 gibt es im BCD-Format nicht. Hier wäre die Wertigkeit der Nibbles 12 und 10. Im BCD-Format gibt dies jedoch keinen Sinn, weshalb Bytewerte wie diese wegfallen.
In der Informatik spricht man dann von einem "redundanten Code"- man kann mit ihm weniger Elemente kodieren, als es Möglichkeiten gibt. Welchen Vorteil gibt uns nun das BCD-Format in Bezug auf die Timerprogrammierung? Nun aufgrund der einzelnen Nibblecodierung können wir sehr leicht die Ziffern der Minuten, Stunden, etc. herausfinden. Eine komplizierte Binär-Dezimal-Umwandlung fällt weg. Des Weiteren können wir mit dem Prozessor im BCD-Format rechnen. Hierzu setzt man zunächst das Dezimal-Flag mit dem Assembler- Befehl SED. Hiernach verhalten sich die Befehle ADC und SBC so, daß sie immer BCD-Werte liefern, vorausgesetzt, man addiert auch BCD-Werte miteinander. Hierzu drei kleine Beispiele:

1) $11 + $12 = $23
2) $11 + $0C = $1D
3) $12 + $19 = $31

Ich habe in den Beispielen Hexadezimalzahlen verwendet, weil mit Ihnen BCD-Zahlen einfacher anzuzeigen sind. Hexzahlen stellen ja ebenfalls die zwei Nibbles eines Bytes dar, nur daß man alle Möglichkeiten eines Nibbles berücksichtigt (von 0 bis F). Beachten wir nun, daß Zahlen wie $1C gar keine BCD-Zahlen sind, und verwenden wir sie auch nicht, so können durch das Hexadezimalsystem sehr einfach BCD-Zahlen dargestellt werden. Hierbei entspricht die Zahl $12 nicht wie sonst dem Dezimalwert 18, sondern tatsächlich dem Wert 12 !!!
Das erste und dritte Beispiel soll Ihnen nun verdeutlichen, wie im BCD-Format gerechnet wird.11+12 ergibt tatsächlich 23. Das wäre nun jedoch nichts neues, da $11+$12 auch $23 ergäbe. Deshalb zeigt Beispiel 3 das Aufreten eines Überlaufs. Ist der BCD-Modus eingeschaltet, so erhalten wir aus der Addition 12+19 das Ergebnis 31, was auch richtig ist. Bei abgeschaltetem BCD-Modus wäre $12+$19 gleich $2B! Beispiel 2 dient als Gegenbeispiel für eine BCD-Rechung. Weil wir hier mit $0C addierten, was ja keine BCD-Zahl ist, kommt auch

keine BCD-Zahl als Ergebnis heraus. Soviel zum BCD-Format.
Kommen wir nun zurück zu den Uhren der CIAs. Um die Uhr einer CIA zu Setzen müssen wir also nur BCD-Zahlen in die jeweiligen Register schreiben um die aktuelle Zeit einzustellen. Hierbei müssen wir jedoch noch einige Dinge beachten:

1. Beim Setzen der Uhrzeit sollte IMMER zunächst das Register für die Stunden (Reg.11) beschrieben werden. Wird nämlich auf dieses Register zugegriffen, so hält die entsprechende CIA die komplette Uhr an. Dies ist deshalb so wichtig, da die Uhr ja gerade in dem Moment, in dem wir schreiben auf die nächste Stunde umspringen könnte. Würden wir eine 10 in das Stundenregister schreiben und die restlichen Register stünden auf der Zeit ":59:59.9", so würde noch während wir die Minuten schreiben die Uhr die volle Stunde erreichen und unsere 10 wäre eine 11, und das ist nicht die aktuelle Uhrzeit!
   Umgekehrt verhält es sich mit dem Register für die Zehntelsekunden (Reg.8). Erst, wenn es beschrieben wurde, wird die Uhr wieder in Gang gesetzt. Deshalb müssen wir also immer gleich die komplette Uhrzeit setzen, damit die Uhr auch wirklich läuft! Ähnlich verhält es sich auch beim Lesen. Hier sollten wir ebenfalls IMMER zuerst die Stundenzahl lesen. Dies veranlaßt die CIA zum Zwischenspeichern der Uhrzeit in den 4 Uhrregistern zum Zeitpunkt des Zugriffs. Intern läuft die Uhr allerdings weiter, sie wird also NICHT angehalten.
   So können wir immer die richtige Zeit, nämlich die, die zum Zeitpunkt des Zugriffs in der Uhr stand, auslesen. Auch hier muß das Zehntelsekundenregister als letztes ausgelesen werden, damit die tatsächliche Uhrzeit wieder in die 4 Uhrregister übertragen wird.
2. Die CIA-Uhren sind zwar echte 24h-Uhren, jedoch zählen sie nicht, wie wir es gewohnt sind von 0 bis 23 Uhr, sondern sie arbeiten nach dem amerikanischen Zeitsystem, das eine Unterscheidung von Vor- und Nachmittag berücksichtigt und nur die Stunden von 0 bis 12 kennt. Sie haben das sicher schon einmal bei einer Digitaluhr beobachtet - sobald es 12 Uhr mittags ist springt die Anzeige auf "PM" um. Das steht für " post meridian" und bedeutet nichts anderes als " nach Mittag". Ebenso erscheint auf dem Display ein "AM" für "ante meridian" (="vor Mittag") wenn die Uhr von 11:59 PM auf 12 Uhr nachts umschaltet. Dieses Verfahren wird ebenso von den CIA-Uhren benutzt. Das 7. Bit des Stundenregisters gibt an, ob es Vor-, oder Nachmittag ist. Ist es gelöscht, so haben wir "AM", ist es gesetzt, so ist "PM" . Dies müssen wir also ebenfalls bei der Programmierung berücksichtigen - sowohl beim Lesen, als auch beim Schreiben. CLOCK ist übrigens so ausgelegt, daß es beide Arten der Zeitdarstellung berücksichtigt. Dazu später mehr.
3. Bei den CIA-Uhren ist mir noch eine kleine Besonderheit aufgefallen, von der ich nicht weiß, ob sie absichtlich ist und einem Standard entspricht, oder ob sie eine Fehlfunktion darstellt.
   Es ist nämlich möglich, wie sollte es auch anders sein, die Stunde 0 Uhr (also 12 Uhr nachts) in das Stundenregister einzutragen. Die CIA liefert dabei keine Fehler sondern übernimmt die Zeit wie sie ist. Sie zählt nun bis 12 Uhr PM und springt dann auf 1 Uhr PM (=13 Uhr) um. Soweit nichts besonderes. Der Witz ist nun, daß nachts um 11 :59 PM nicht auf 0:00 AM geschaltet wird, sondern auf 12:00 Uhr AM. Hier scheint also ein kleiner Fehler zu sein (vielleicht begründet durch den internen Aufbau der CIAs), oder haben Sie schon einmal eine Uhr gesehen, die diese Zeit anzeigt? Ich nicht. Deshalb ist CLOCK auch so programmiert, daß es bei 12 Uhr AM nicht 12:00 Uhr sondern 0 :00 anzeigt. Dies nur als Hinweis.

So. Nun wissen Sie also, wie man die aktuelle Uhrzeit in den 4 Uhrregistern einer CIA unterbringt, und wie man sie dort wieder herausholt. Das ist jedoch noch nicht alles, was man zu einer korrekten Uhr-Programmierung braucht. Wir müssen nämlich vor dem Einstellen der Uhr noch 2 Dinge beachten.

1. Zunächst müssen wir der CIA, die unsere Uhr steuert, mitteilen, mit welcher Netzfrequenz der 64 er arbeitet, in dem sie drinsteckt. Dies liegt daran, daß in Amerika eine andere Stromnorm benutzt wird, als hier bei uns in Europa. Dort beträgt die Frequenz des Wechselstroms aus der Steckdose nämlich 60 Hertz und nicht etwa 50, wie das bei uns üblich ist. Aus diesem Grund kann man der CIA auch mitteilen, welche der beiden Frequenzen nun benutzt wird, damit sie in beiden Stromnetzen richtig arbeitet kann. Diese Funktion legt Bit 7 des CRA-Registers (Reg.14) fest. Steht es auf 1, so beträgt der Echtzeituhrtrigger 50 Hz, steht es auf 0, so ist er 60 Hz. Weil wir hier in Europa eine Netzfrequenz von 50 Hz haben, müssen wir auch dementsprechend das 7. Bit von CRA setzen!
2. Beim Einstellen der Uhrzeit verlangt die CIA noch eine weitere Information von uns. Wie ich anfangs ja schon erwähnte, kann der Echtzeituhr auch eine Alarmzeit mitgeteilt werden, zu der ein Interrupt auftreten soll.
   Diese Alarmzeit wird nun aber in die- selben Register geschrieben, wie die Uhrzeit, also ebenfalls in die Register 8 bis 11. Bit 7 von CRB (Reg. 15) ist nun dafür zuständig zu unterscheiden, ob gerade die Alarm-, oder die Uhrzeit gesetzt wird. Steht es auf 1, so setzen wir die Alarmzeit, steht es auf 0 so wird die Uhrzeit geschrieben. Dies müssen wir also berücksichtigen, wenn wir eine der beiden Zeiten einstellen wollen.

Dies wäre alles, was Sie zur Programmierung der Echtzeituhr wissen müssen. Lassen Sie mich nun zum praktischen Beispiel schreiten und Ihnen die Funktionsweise von CLOCK erklären. Hierzu wollen wir uns erst einmal überlegen, wie CLOCK überhaupt arbeiten soll:

1. Zunächst soll CLOCK in den Systeminterrupt eingebunden werden, von wo aus es ständig die Ausgabe aktualisiert.
2. CLOCK soll eine Alarmfunktion beinhalten, die bei erreichen der Alarmzeit einen Piepton ausgibt.
3. Es soll möglich sein zwischen der 24h und der AM/ PM-Darstellung der Uhrzeit zu wählen. Hierzu habe ich mit die Speicherzelle 3 als Modusregister ausgesucht, die normalerweise vom Basic-Befehl USR benutzt wird. Da dieser Befehl jedoch wenig Anwendung findet, kann man bedenkenlos diese Speicherzelle für eigene Zwecke nutzen. Wenn sie 0 ist, so soll die AM/ PM-Darstellung verwendet werden. Ist sie ungleich 0, so wird die 24 h-Darstellung gewünscht. Egal, in welchem Modus CLOCK laufen soll, die Eingabe der Uhrzeit soll immer in der 24 h-Darstellung geschehen. Diese wird in der Form " HHMMSS" angegeben.
4. Zur optischen Aufmachung habe ich die Ziffern 0 bis 9 als Sprites in den Spriteblöcken 33 bis 42(inklusive) untergebracht. DesWeiteren befinden sich den den Blöcken 43 und 44 je ein Sprite für die AMund PM-Anzeige. Die Uhrzeit wird durch die Sprites des 64ers auf dem Bildschirm angezeigt.

Kommen wir nun also zum Source-Code Listing von CLOCK. Zunächst wollen wir uns einmal die IRQ-Initialierung anschauen:

```
========================================
     init      LDA #$81          Uhr-Trigger auf 50Hz und Timer A starten
               STA cia1+14       in CRA festlegen
               LDA #<(txt1)      LO-Byte Text1
               LDY #>(txt1)      HI-Byte Text1
               LDX #$00          Wert für CRA (=Uhrzeit setzen)
               JSR setit         Unterroutine für Zeit einlesen und setzen aufrufen
               LDA #<(txt2)      LO-Byte Text2
               LDY #>(txt2)      HI-Byte Text2
               LDX #$80          Wert für CRA (=Alarm setzen)
               JSR setit         Alarmzeit Lesen und Setzen
               SEI               IRQs sperren
               LDX #<(newirq)    ..und die neue
```

```
        LDY #>(newirq)      IRQ-Routine
        STX $0314           im IRQ-Pointer
        STY $0315           setzen
        LDX #$15            Kopiert Liste mit
loop1   LDA xtab,X          Sprite Koordina-
        STA v,X             ten in die ent-
        DEX                 sprechenden VIC-
        BPL loop1           register.
        LDX #$07            Setzt die Farben
        LDA #$01            für alle
loop2   STA v+39,X          7 Sprites
        DEX                 auf
        BPL loop2           "Weiß"
        LDA #$85            Alarm- und Timer-IRQs erlauben
        STA cia1+13         und im ICR festlegen
        LDA #$15            Hüllkurve für Pieton...
        STA sid+5           ...festlegen.
        lda #$c0            Frequenz von Piepton
        sta sid+1           festlegen.
        ldx #<(end+1)       CLOCK-Endadr. LO
        ldy #>(end+1)       CLOCK-Endadr. HI
        stx $2b             und BASIC-Anfang
        sty $2c             ...setzen
        jsr $a642           "NEW" ausführen
        cli                 IRQs wieder frei-
        rts                 geben und Ende.
========================================
```

Dies wäre also die Initialisierungsroutine für CLOCK. Als Erstes wird der Wert $81(= binär 10000001) in CRA geschrieben. Damit setzen wir die Echtzeituhrtriggerung auf 50 Hz (Bit 7=1) . Gleichzeitig müssen wir aufpassen, daß wir nicht versehens Timer A anhalten, der ja den System-IRQ steuert. Deshalb muß also auch Bit 0 gesetzt sein, was für "Timer A starten" steht. Er läuft zwar bereits, jedoch würden wir ihn mit einer 0 anhalten. Die 1 ändert also nichts an dem momentanen Zustand von Timer A, sie verhindert nur eine Änderung.
Als Nächstes wird die absolute Anfangsadresse eines Textes, den ich im Speicher abgelegt habe, in LO/ HI-Darstellung in A/ Y geladen und der Wert 0 in das X-Register. Dies dient der Parameterübergabe für die Routine "SETIT", die anschließend aufgerufen wird. Sie gibt den Text, dessen Anfangsadresse in A / Y steht aus und liest anschließend eine Uhrzeit ein, die sie in den Uhrregistern von CIA1 speichert. Zuvor schreibt sie den übergebenen Wert des X-Registers in CRB und setzt somit den Alarm oder Uhzeit-Setzen-Modus. Der erste Aufruf von SETIT setzt die aktuelle Uhrzeit, der zweite die Alarmzeit. Anschließend wird der neue Interrupt eingestellt - alle IRQs werden gesperrt und der IRQ-Zeiger des Betriebssystems wird auf unsere neue Interruptroutine mit Namen " NEWIRQ" verbogen. Nun folgen zwei Schleifen, die die Sprites, die wir benutzen wollen um die Uhrzeit auszugeben vorbereiten. Die erste Schleife kopiert eine Liste, die ich etwas weiter hinten im Source-Code abgelegt habe in den Bereich von $ D000 bis $ D00 F. Diese Register des VIC sind für die Koordinaten der Sprites verantworlich, die nun entsprechend gesetzt sind. Die zweite Schleife schreibt in die VIC-Register 39 bis 46 den Wert 1, und setzt somit die Farbe für alle Sprites auf Weiß. Nun müssen wir die CIA1 noch darauf vorbereiten, daß Echtzeituhr-Alarm- IRQs ausgelöst werden sollen. Dies geschieht, indem wir den Wert $85 in das ICR (Reg. 13) von CIA schreiben. Damit lassen wir zum Einen die immer noch gültigen Timer-IRQs zu die von Timer A erzeugt werden zu (für den System-IRQ), zum Anderen haben wir mit dem Wert $85 auch noch das 2 . Bit gesetzt, das Alarm-IRQs als Interruptquelle festlegt. Bit 7 muß wie immer gesetzt sein, damit die übrigen Bits als IRQ-Quelle übernommen werden. Nun müssen wir noch den SID darauf vorbereiten, einen Piepton auszugeben, wenn ein IRQ auftritt. Deshalb wird das HI-Frequenzregister von Stimme1(Reg.1) mit dem Wert $C0 geladen und eine Hüllkurve in Reg.5 des SID geschrieben. Register 6, das ja

ebenfalls Attribute der Hüllkurve angibt, wird nicht neu beschrieben, da es in der Regel den Wert 0 enthalten sollte, den ich dort auch haben wollte.
Zum Abschluß folgen nun noch 5 Befehle, die den Anfang des BASIC-Speichers höhersetzen. Dies ist erforderlich, da ich nämlich am normalen Anfang die Sprites und den Code von CLOCK untergebracht habe. Würden Sie nun ein BASIC-Programm eingeben, so würden Sie CLOCK überschreiben, was nicht sein sollte. In den Adressen $2B und $2C hat das Betriebssystem die Adresse des BASIC-Starts gespeichert. Wir setzen diese Adressen nun einfach auf die Endadresse von CLOCK und rufen anschließend die Routine bei $A642 auf. Dort steht die BASIC-Routine für den Befehl "NEW" .Dies ist notwendig, damit die neue Anfangsadresse des BASIC- Speichers auch vom Betriebssystem angenommen wird.
Die Initialisierungsroutine gibt nun die IRQs wieder frei und springt zurück. Kommen wir zu der neuen IRQ-Routine, die die Uhr steuert. Sie muß zunächst prüfen, ob der ausgelöste Interrupt ein Timer-, oder ein Alarm-IRQ war und dementsprechend reagieren. Bei Alarm gibt sie einen Piepton aus, bei einem Timer-IRQ wird nur die aktuelle Zeit aus CIA1 ausgelesen und mit Sprites auf dem Bildschirm dargestellt. Hierbei muß sie unterscheiden, ob die Zeit nun in 24 hoder AM/ PM-Darstellung auf dem Bildschirm erscheinen soll. Werfen wir doch einfach einmal einen Blick auf diese Routine:

```
========================================
        NEWIRQ ist die IRQ-Routine von CLOCK.
        Hier wird zunächst einmal die vom Dar-
        stellungsmodus abhängige Spriteanzahl
        eingeschaltet.
========================================

        newirq      LDA #$7F        Wert für Sprites 0-6 einschalten (für 24h)
                    LDX mode        Uhr-Modus prüfen.
                    BNE l5          ungleich 0, also 24h
                    LDA #$FF        Sonst AM/PM, deshalb Wert für "Sprites 0-7 einschalten" laden
        l5          STA v+21        und einschalten

        ===
        Hier wird geprüft ob ein Alarm-IRQ (Bit2
        von ICR=1) Auslöser war. Wenn ja, so wird
        ein Piepton ausgegeben.

        ===
                    LDA cia1+13     ICR auslesen
                    AND #$04        Bit2 isolieren
                    BEQ timeref     Wenn =0, dann kein Alarm und weiter
                    LDA #15         Sonst Lautstärke
                    STA sid+24      einschalten
                    LDA #33             Stimme 1 als "Sägezahn"
                    STA sid+4       einschalten
                    LDX #$FF            Und warten...
        loop5       DEY             (dies...
                    BNE loop5       ...ist...
                    DEX             ...eine...
                    BNE loop5       ...Warteschleife)
                    LDA #00         Ton gespielt, deshalb
                    STA sid+4       Stimme1 aus
                    STA sid+24      Lautstärke aus

========================================
        Dies ist der Teil von NEWIRQ, der die
        Zeit neu ausgibt.
========================================

        timeref     LDX #43             Spriteblock "AM" in X laden
                    LDA cia1+11     Stunden in Akku
```

```
                BMI l6              Wenn 7.Bit=1 habe wir "PM", dehalb weiter
                CMP #$12            Akku=12 (in BCD)?
                BNE l2              Nein, also weiter
                LDA #00                 12 Uhr AM gibts nicht, deshalb 00 Uhr AM
                BEQ l2              Unbedingter Sprungadresse

===
Hier wird hinverzweigt, wenn PM ist.
===

l6              INX                 X-Reg enthält Spriteblock für  "AM" - jetzt für "PM"
                AND #$7F            Bit7 (=PM) der Stunden löschen
                LDY mode            Modus prüfen
                BEQ l2              Wenn AM/PM dann weiter
                CMP #$12            Akku = 12?
                BEQ l3              Ja, dann nicht 12 addieren
                SED                 BCD-Mode an
                CLC                 Weil PM, 12 ad-
                ADC #$12            dieren (für 24h)
                CLD                 BCD-Mode aus
l2              STX sprpoi+7        AM/PM-Sprite schalten
========================================
        Nun folgt der Teil von NEWIRQ der die aktuelle
        Zeit ausgibt.
========================================

l3              JSR getnum          Akkuinhalt in Spritepointer umwandeln
                STX sprpoi+0        Sprites für Stun
                STA sprpoi+1        den setzen
                LDA cia1+10         Minuten laden
                JSR getnum          ...wandeln
                STX sprpoi+2        ...und
                STA sprpoi+3        ...setzen
                LDA cia1+9          Sekunden laden
                JSR getnum          ...wandeln
                STX sprpoi+4        ...und
                STA sprpoi+5        ...setzen
                LDA cia1+8          Zehtelsek. laden
                CLC                 Spritepointer der
                ADC #sprbas         Ziffer 0 addiern
                STA sprpoi+6        und setzen
                JMP $EA31           IRQ beenden
========================================
```

Zunächst einmal möchte ich Ihnen die In halte und die Bedeutung verschiedene Labels in diesem Teil des Listings erläutern:

- MODE enthält den Wert 3 und steht fü die Speicherzelle 3, in der wir ja festgelegt hatten in welchem Darstellungsmodus CLOCK arbeiten soll.
- SPRPOI enthält den Wert 2040, die Basisadresse der Spritepointer. In diese Pointern wird angegeben, welcher Spriteblock in einem Sprite dargestellt werden soll.
- SPRBAS enthält den Wert 33 und steht für den ersten Spriteblock, den wir verwenden. In ihm steht die Ziffer 0 al Sprite. Addieren wir zu einem Wert i Akku (zwischen 0 und 9) den Wert SPRBA hinzu, so erhalten wir den Zeiger au den Spriteblock mit der entsprechende Ziffer des Wertes, den wir im Akku stehen hatten.
- GETNUM ist eine Routine, die eine BCD Zahl im Akku in die zwei Ziffernwert aufspaltet und SPRBAS zu diesem Wer hinzuaddiert. In X-Register und Akku werden die Spritepointer der beide Ziffern zurückgegeben.

Im ersten Teil von NEWIRQ wird der Darstellungsmodus geprüft und, abhängig da von, entweder sieben oder acht Sprite eingeschaltet. Im AM/ PM-Modus wird das 8 Sprite nämlich zur Darstellung von A oder PM verwendet.
Als nächstes prüft NEWIRQ, was die Interruptquelle des IRQs war. Ist Bit 2 im IC von CIA1 gesetzt, so war es der Alarm IRQ. Wenn dieser der Auslöser war, wird der voreingestellte Piepton eingeschaltet und für die Dauer der folgende Warteschleife gespielt. Anschließend wir er wieder abgeschaltet und NEWIRQ fähr mit der Zeitausgabe fort. Sicherlich ist die Tonausgabe mit einer Warteschleife nicht unbedingt die eleganteste Art und Weise dieses Problem zu lösen, jedoch genügt Sie für den Zweck als Beispielprogramm. Sehen Sie sich doch einfach gefordert, und versuchen Sie eine eigen Alarm-Routine zu schreiben, die läuft ohne den IRQ zu verzögern.
Es folgt nun die Routine TIMEREF, die di Ausgabe der Uhrzeit erledigt. Hierzu müssen wir zunächst einmal herausfinden, ob die Stunden in AM/ PM oder in 24 h Darstellung erscheinen soll. Dementsprechend muß nämlich dann die Stundenzahl modifiziert werden. Das geschieht am Anfang von TIMEREF. Das Sprite für AM/ PM wir übrigens immer mit gesetzt, selbst wen die 24 h-Darstellung gewählt wurde. In diesem Fall ist es jedoch nicht zu sehen weil es am Anfang von NEWIRQ abgeschalten wurde. Wenn das Programm herausfindet, daß die Uhr 12 Uhr AM anzeigt, was ja nach unserer Definition keine logische Uhrzeit ist, so lädt es 0 in den Akku, um später bei der Ausgabe ein "00" als Stunde er scheinen zu lassen.
Wenn es Nachmittag ist, stellt TIMEREF zunächst einmal den Spritepointer für da AM/ PM-Sprite auf "PM", indem es das X Register um 1 erhöht. Anschließend lösch es das 7. Bit der Stundenzahl, da diese ja nur angibt, daß Nachmittag ist, und später bei der Umwandlung in eine Ziffer stören würde. Wenn die AM/ PM-Darstellung aktiv ist, wird direkt zu dieser Umwandlung weiterverzweigt. Andernfalls prüft TIMEREF, ob es nicht gerade 12 Uhr PM ist, da bei der 24 h-Darstellung in diese Fall ja NICHT der Wert 12 zu der Stundenzahl addiert werden soll. Ist dies nicht der Fall, so müssen wir den Wert 12 doch noch zur Stundenzahl hinzuaddieren, wobei wir den BCD-Modus des Prozessors benutzen. Nun ist die Stundenzahl berechnet und kann an die Umwandlungsroutine weitergegeben werden.
Der restliche Teil von NEWIRQ erklärt sich von selbst.
Damit wissen Sie nun, wie man eine CIA Echtzeituhr programmiert. Die übrigen Programmteile von CLOCK (die schon erwähnten Routinen SETIT und GETNUM) möchte ich hier aussparen, da sie mit dem Thema unseres Kurses wenig zu tun haben und allgemeingültige Probleme lösen. Nichts destotrotz können Sie sie sich ja einmal im Source-Code-Listing auf dieser MD an schauen.
Probieren Sie CLOCK doch einmal aus. Wen Sie es starten so werden Sie zunächst nach Uhr und Alarmzeit gefragt. Anschließend können Sie die Uhr am unteren Bildschirmrand beobachten. Wenn Sie CLOC starten, so müßte die Zeit in der 24 h Darstellung erscheinen, da die Speicherstelle 3 in der Regel einen Wert ungleich 0 enthält. Schalten Sie dann doch einfach einmal mit "POKE 3,0" und "POKE 3,1" zwischen den beiden Modi um. Sie werden sehen, daß CLOCK durch die IRQ Programmierung immer direkt auf Ihre Eingaben reagiert.
So. Das war es dann mal wieder für diesen Monat. Sie können ja einmal versuchen eine noch komfortablere Uhr zu programmieren - wie es geht wissen Sie jetzt ja. Wie wäre es zum Beispiel mit einer Funktion, die stündlich einen Piepton aus gibt? Oder sogar gleich eine ganze Reihe von Intervall-Tönen, wie man es von Quartz-Weckern her kennt?
Ich hoffe, Ihnen damit einige Übungsanregungen gegeben zu haben und verabschiedet mich nun bis nächsten Monat. Dann werde wir uns mit einem der Hauptanwendungsgebiete der CIAs beschäftigen: der Ein- und Ausgabe.
Bis dahin wünsche ich Ihnen viel Zeit und Piep,

Ihr Uli Basters (ub)

Teil 7 – Magic Disk 05/91

Hallo und willkommen zum 7. Teil des CIA-Kurses. Diesen Monat wollen wir uns mit der

wichtigsten Funktion der CIA-Bausteine beschäftigen, durch die sie erst so richtig leitungsfähig werden.
Die beiden kleinen Chips steuern nämlich den kompletten Verkehr mit der Außenwelt des C64 . Sei das nun die Bedienung eines Diskettenlaufwerks, eines Druckers, der Tastatur oder sogar die Kommunikation mit einer Hardware-Erweiterung am Userport, alles geht nur mit den CIAs. Und das sogar relativ einfach. Kommen wir zunächst einmal zu den Grundeinheiten in den CIAs, die die für Ein-/ Ausgabe bestimmt sind, den Portregistern.
Jede CIA verfügt nämlich über jeweils zwei frei programmierbare 8-Bit-Ports. Jeder dieser Ports wird von je einem Register der CIA repräsentiert. Die Bits, die Sie dort auslesen, beziehungsweise hineinschreiben, kommen von, oder erscheinen an den jeweiligen Portleitungen der entsprechenden CIA und sind aus ihr herausgeleitet. "Frei programmierbar" heißt, daß diese Ports sowohl zur Ein-, als auch zur Ausgabe benutzt werden können. Die jeweilige Funktion, die man benutzen möchte, kann softwaremäßig festgelegt werden. Und das nicht nur für einen ganzen Port, sondern wir können sogar die einzelnen Bits eines Ports, je nach Bedarf auf Ein- oder Ausgabe schalten. Die Portleitungen erscheinen an den verschiedensten Stellen wieder. So sind zum Beispiel die Portbits der CIA2 am Userport zu finden, oder die der CIA1 an den beiden Control Ports für Joysticks, beziehungsweise intern an der Tastatur. Sie sehen also, daß die Ports auch mehrfach benutzt werden. Daher erklärt es sich auch, daß Sie, wenn Sie den Joystick in Port1 ein wenig hin und her bewegen, wirre Zeichen auf dem Bildschirm erscheinen. Das Betriebssystem des C64 glaubt nämlich, die Tastatur würde bedient und gibt die entsprechenden Zeichen aus. Doch dazu wollen wir später noch einmal kommen. Zunächst will ich Ihnen erst einmal erläutern, wie die Abfrage dieser Geräte funktioniert. Kommen wir also zu der Funktionsweise der Portprogrammierung. Man unterscheidet die Ports einer CIA mit den Bezeichnungen "Port A" und "Port B" . Die Register dieser Ports finden sich in den Registern 0 und 1 der entsprechenden CIA wieder. Sie heißen Portregister A und B. Zu jedem dieser Ports gibt es nun auch die sogenannten Datenrichtungsregister. Hier wird bestimmt, welches Bit eines Ports auf "Eingabe", und welches auf "Ausgabe" steht. Die Datenrichtungsregister sind in den Registern 2 und 3 einer CIA zu finden. Hier nochmal eine kleine Übersicht (die Abkürzungen werden wir der Einfachheit ab jetzt immer benutzen):

Register	Abkürzung	Name
0	PRA	Portregister A
1	PRB	Portregister B
2	DDRA	Datenrichtungsregister Port A
3	DDRB	Datenrichtungsregister Port B

Ist ein Bit im Datenrichtungsregister eines Ports gelöscht, so ist das entsprechende Bit des Ports auf "Eingang" geschaltet; ist es gesetzt, so steht das Bit des Ports auf "Ausgang". Dem Programmierer sind hier keine Grenzen gesetzt. Schreibt man einen 8-Bit-Wert in einen Port, dessen Bits unterschiedlich, also in beide Richtungen geschaltet sind (z.B. Bits 0-3 auf "Ausgang"=1, Bits 4-7 auf "Eingang"=0), so werden auch nur die Bits des entsprechenden Portregister an den Ausgang gelegt, die als solcher geschaltet sind (im Beispiel die Bits 0-3). Die übrigen Bits werden ignoriert, beziehungsweise vom Eingang überschrieben. Im Handling mit den Portregister müssen wir übrigens dringend darauf achten, daß in der CIA ein Inverter eingebaut ist, der die Ein- und Ausgangssignale invertiert. Fragen Sie mich nicht warum, für die Programmierung ist es jedoch unablässig dies zu wissen.
So müssen wir beim Lesen eines Datenports darauf achten, daß die Eingangsbits immer in invertierter Schreibweise im Register erscheinen. Ist dabei also ein Port, der auf "Eingang" geschaltet ist, unbelegt, das heißt, daß an ihm kein Signal anlegt, so sind die Bits im Datenportregister auf 1. Liegt eine Spannung an einem der Bits an, so erscheint es im Portregister als 0! Das ist wichtig zu wissen, da wir den gelesenen Wert dann nämlich erst

einmal wieder invertieren müssen, wenn wir ihn als normalen Zahlenwert lesen wollen. Dazu benutzt man dann den EOR-Befehl. Mit EOR #$FF kann man den Inhalt des Akkus invertieren. Umgekehrt müssen wir beim Beschreiben der Portregister darauf achten, daß wir das was wir an Eins-Bits erscheinen lassen wollen umgekehrt schreiben müssen.
Wenn Sie also zum Beipsiel die Bitfolge 11110000 am Port A der CIA2 ausgeben wollen (dieser ist am Userport herausgeführt), dann müssen Sie die Bitfolge 00001111 in das Portregister (Reg.0 von CIA2=$ DD00) schreiben!
Kommen wir nun zur Tastaturabfrage. Die Tastatur wird mit Hilfe der Datenports A und B der CIA1 abgefragt. Dies geschieht auf eine besonders pfiffige Art und Weise. Da jeder Port über jeweils 8 Bits verfügt, könnte man eigentlich nur 16 Tasten abfragen. Mit diesen 2 x8 Bits kann man aber auch eine Matrix bilden, mit der 2↑8=64 Kombinationen abgefragt werden können. Die Abfrage dieser Kombinationen ist nun etwas kompliziert. Deshalb gibt es jetzt erst einmal eine Grafik mit der Belegung der Matrix. Drucken Sie sie sich am besten aus, oder malen Sie sie ab, da ich sie später bei der Erklärung dringend benötige...

Port A	**Bit 0**	CRSR ↑↓	F5	F3	F1	F7	CRSR ←→	Return	DEL
	Bit 1	SHIFT LEFT	E	S	Z	4	A	W	3
	Bit 2	X	T	F	C	6	D	R	5
	Bit 3	V	U	H	B	8	G	Y	7
	Bit 4	N	O	K	M	0	J	I	9
	Bit 5	,	@	:	.	-	L	P	+
	Bit 6	/	↑	=	SHIFT RIGHT	HOME	;	*	£
	Bit 7	RUN STOP	Q	Comodore Taste	SPACE	2	CTRL	←	1
Port B		**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**

Die Entwickler des C64 haben nun 2 Voraussetzungen geschaffen, die eine Matritzenabfrage der Tastatur ermöglichen:

1. Jede einzelne Taste der Tastatur kann man sich als einen Ein/ Aus-Schalter vorstellen, der nur solange einen Strom durchschaltet, wie die Taste gedrückt ist.
2. Der Strom, der durchgeschaltet wird kommt - jetzt halten Sie sich fest - von Port A der CIA1. Die Bits dieses Ports sind alle auf " Ausgang" geschaltet und gesetzt. Wie Sie aus der obigen Grafik erkennen konnten, sind die Leitungen der einzelnen Bits von Port A in der Waagerechten für jeweils acht Tasten durchgeschleift.

Wird nun eine Taste gedrückt, so schaltet sie ein Signal senkrecht durch, wo widerum acht Tasten miteinander verbunden sind. Port B dient nun als Eingang für widerum acht solcher Signale. Mit Port A sind also die waagerechten, mit Port B die senkrechten Tasten vesorgt. Über eine Kreuzpeilung kann man nun genau feststellen, welche Taste gerade gedrückt wird.

Hierzu eine genaue Beschreibung:
Die Tastaturabfrage des Betriebssystems legt nun zunächst an allen Bits von Port A Eins-

Signale an. Wird keine Taste gedrückt, so ist auch nichts in Port B zu sehen. Alle seine Bits sind auf Null (das heißt für uns auf 1, weil die Eingangsbits ja von der CIA invertiert werden). Nun brauchen Sie die Tabelle von eben. Wird jetzt nämlich eine Taste gedrückt, als Beipiel nehme ich mal die Taste "J", so wird das Eins-Signal von Port A, Bit4, an Port B, Bit2 durchgeschaltet. Die Tastaturabfrageroutine erkennt so, daß überhaupt eine Taste gedrückt wurde, da der Inhalt von Port B jetzt ja nicht mehr Null ist. Nun beginnt sie, alle Bits einzeln durchzutesten, indem sie der Reihe nach die Bits von 0 bis 7 von Port A auf Eins setzt und prüft, ob Bit2 von Port B immer noch gesetzt ist. Bei Bit0 ist das nicht der Fall, ebenso bei Bit1,2, und 3. Wenn sie jetzt Bit4 von Port A auf Eins legt, so erscheint es an Bit2 von Port B wieder. Die Taste wäre lokalisiert! Nun sucht die Abfrageroutine sich noch aus einer Tabelle im Betriebssystem den entsprechenden Tastencode heraus, speichert ihn zwischen und schreibt ihn zusätzlich in den Tastaturpuffer, von wo aus die Ausgaberoutine des Betriebssystems das Zeichen auf dem Bildschirm ausgibt. Als Beispiel habe ich Ihnen einmal ein kleines Programm vorbereitet. Es wartet, bis eine der SHIFT-Tasten gedrückt wird und kehrt dann wieder zum aufrufenden Programm zurück. Dabei müssen wir jedoch darauf achten, daß die SHIFT-Tasten bei der Tastatur unterschieden werden in linke und rechte Taste. Unser kleines Prögrämmchen muß also zwei Abfragen machen, damit es erkennt, wann wir SHIFT gedrückt haben. Zunächst möchte ich Ihnen jedoch den Source-Code des Programms hier auflisten. Sie finden ihn auch wie immer auf dieser MD unter dem Namen "WAITSHIFT. SRC". Das lauffähige Assemblerprogramm heißt "WAITSHIFT.OBJ" und muß absolut (",8,1") geladen werden. Es liegt ab Adresse $ C000(dez. 49152) und muß mit SYS49152 gestartet werden. Nun aber zum Listing:

```
****************************************
start   SEI           System-IRQ sperren
        LDA #$42      waagerechte Reihen mit SHIFT left und right (Bits 1 und
        STA cia1+2    auf "Ausgang" schalten.
        LDA #$00      Alle Bits von Port
        STA cia1+3    auf "Eingang" schalten.
Loop1   LDA #$FC      Bit 1 auf "Eins" legen
        STA cia1+0    und ab ins PRA.
        LDA cia1+1    Jetzt PRB prüfen..
        CMP #$7F      ...wenn SHIFT-LEFT gedrückt, dann ist Bit7 gelöscht!
        BEQ end       Jau, ist so, also Ende.
        LDA #$BF      Sonst Bit 6 auf "Eins" setzen
        STA cia1+0    und wieder ins PRA
        LDA cia1+1    PRB holen...
        CMP #$EF      ...wenn SHIFT-RIGH gedrückt, dann ist Bit4 gelöscht!
        BNE loop1     Ist nicht, also noch mal von vorne!
End     LDA #$FF      DDRA muß für System-Interrupt
        STA cia1+2    zurückgesetzt werden
        CLI           System-IRQs wieder frei geben.
        RTS           Und Tschüß!
****************************************
```

Kommen wir nun zur Dokumentation:
Wie Sie aus der Tastentabelle entnehm können, sind die SHIFT-Tasten folgendermaßen codiert: SHIFT-LEFT wird durch Bi von PRA angesprochen und schaltet zu Bit von PRB durch. Ebenso bekommt SHIFT-RIGHT sein Signal von Bit6 von PRA und schaltet nach Bit4 von PRB durch. Möchten wir nun also ganz gezielt diese Tastenabfrage, so müssen wir zunächst genau das Bit von PRA setzen, das die entsprechende Taste ansteuert. Dann muß geprüft werden, ob das Bit, das von dieser, und NUR von dieser Taste durchgeschaltet wird auch auf 1 ist. Ist dies der Fall, so war die Taste tatsächlich gedrückt.
Zuvor müssen wir jedoch noch ein paar Vorbereitungen treffen. Dies geschieht in den ersten 5 Zeilen von WAITSHIFT. Zunächst müssen wir den System-IRQ sperren, da ja sonst die Tastatur für uns abfragt, und uns nur stören würde. Da er über CI läuft, genügt es, mit SEI alle IRQs zu unterbinden. Anschließend müssen die Datenrichtungsregister der beiden Ports für unsere

Zwecke ausgerichtet werden. Hier schreiben wir zunächst den Wert $42 in DDRA. $42 entspricht dem Binärwert 01000010. Hier sind die Bits 1 und 6 gesetzt, womit wir sie auf Ausgang schalten. Alle anderen sind Eingang. Dadurch legen wir fest, daß Signale nun nur noch von den Tasten die in diesen Reihen (der Tabelle) liegen kommen. In DDRB kommt eine 0. Es würde zwar genügen, wenn wir nur die Bits 4 und 7 als Eingang schalten, jedoch habe ich mich der Einfachheit halber für diese Kombination entschieden, da sie nicht zuletzt einfacher und eben effektiv ist.
Nun beginnt erst die eigentliche Abfrage. Begonnen wird mit der linken SHIFT-Taste. Durch den Wert $FC, den wir in PRA schreiben, legen wir Bit2 auf Eins (00000010 invertiert ergibt 11111101). Es liegt nun ein Pegel dort an, der zu den Tasten SHIFT-LEFT, E, S, Z,4, A, W und 3 durchgeschaltet ist. Wird nun eine dieser Tasten gedrückt, so erscheint die Eins von Bit2 an einem der 8 Bits von PRB wieder. Für die SHIFT-LEFT-Taste wäre das Bit7. Da wir die Signal-Invertierung beachten müssen, muß in PRB beim Auslesen also der Wert $7F ($7F=01111111, das ist ein invertiertes 10000000=$80) stehen. Das wir sogleich durch den CMP-Befehl überprüfen. War es tatsächlich der Fall, so wird am Ende der Routine verzweigt, wenn nicht dann müssen wir nun die rechte SHIFT-Taste abfragen.
Dies erfolgt auf demselben Wege wie vorher. Nur legen wir diesmal Bit6 von PRA auf 1 und untersuchen, ob es an PRB wieder erscheint. Ist dies auch nicht der Fall so werden beide Tasten nochmals überprüft solange, bis eine Taste gedrückt wird.
Zum Schluß muß WAITSHIFT noch den alten Wert (ALLE Bits als Ausgang) in DDRA zurückschreiben, damit die Tastaturabfrage des System-IRQs auch wieder alle Tastenabfragen kann. Nun können wir die IRQs mit CLI wieder freigeben und die Routine beenden.
Nun wissen Sie also, wie man die Tastatur hardwaremäßig abfragt. Das kann sehr hilfreich bei der Programmierung von ein Spielsteuerung sein, da diese Art der Abfrage es einem ermöglicht mehrere Tasten gleichzeitig abzufragen. Bei einem Rennwagenspiel zum Beispiel, oder bei einer einfachen Auto-Simulation kann so zum Beispiel eine Taste für die Kupplung des Autos herhalten und eine andere für den Gang, der eingelegt werden soll. Nur wenn die Kupplung gedrückt ist, kann der Gang eingelegt werden. Dadurch bekommt das Spiel gewissermaßen einen "seriösen Touch"…
Damit soll es dann für diesen Monat gewesen sein in Sachen Geheimdienst. Nächstes Mal geht es weiter mit der Ein-/ Ausgabe-Programmierung. Wir wollen uns dann um eine Joystickabfrage kümmern und noch etwas über ein spezielles Ein-/ Ausgaberegister der CIA erfahren. Bis dahin veli Spaß beim Tastendrücken,

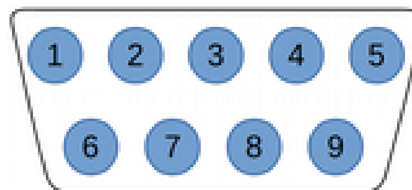
Ihr Uli Baster

Teil 8 – Magic Disk 06/91

Hallo und Willkommen zum 8. Teil unseres Kurses. Diesen Monat soll es weitergehen, mit der Ein-/ Ausgabe über die CIA-Bausteine. Wir werden eine Joystickabfrage programmieren und darüber hinaus die Funktionsweise von anderen Eingabegeräten einmal etwas genauer unter die Lupe nehmen.
Von der letzten Ausgabe her sollten Sie ja noch wissen, daß jede CIA über zwei Ein-/ Ausgabeports verfügt. Diese befinden sich jeweils in den ersten beiden Registern einer CIA. Des Weiteren finden wir zu jedem Port ein Datenrichtungsregister (Register 2 und 3 einer CIA), in denen wir jeweils festlegten, in welcher Richtung Daten verarbeitet werden sollen (Eingabe, oder Ausgabe). Dies soll uns nun für die folgenden Themen als Grundlage dienen. Kommen wir zum ersten Kernthema, der Joystickabfrage:
Wahrscheinlich ist dies für Sie nichts neues mehr; Sie haben sicherlich schon einmal so etwas programmiert, jedoch werden Sie jetzt vielleicht auch die hardwaremäßigen Hintergründe verstehen. Des Weiteren will ich Ihnen hiermit auch die Joyports und ihre Verbindungen zu den CIAs und anderen Bausteinen innerhalb unseres "Brotkastens" erläutern.
Zunächst einmal können wir CIA2 für dieses Thema ausklammern, weil nämlich ausschließlich

die CIA1 für die Bedienung der Joyports verantwortlich ist. An jedem der beiden Ports sind jeweils fünf Leitungen (für die 4 Joystickrichtungen und den Feuerknopf) mit entsprechenden Portleitungen der CIA1 verbunden. Für den Joyport1 sind das die Leitungen PB0-PB4, für Joyport2 die Leitungen PA0-PA4. Des Weiteren sind pro Joyport auch noch weitere Signale zu finden, die ich Ihnen in der folgenden Grafik einmal aufführen möchte:
Hier Grafik 1…

Joyport 1				**Joyport 2**		
Pin	Leitung			Pin	Leitung	
1	Joystick hoch	Port B Bits 0-3		1	Joystick hoch	Port A Bits 0-3
2	Joystick runter			2	Joystick runter	
3	Joystick links			3	Joystick links	
4	Joystick rechts			4	Joystick rechts	
5	Paddle A (Y-Pos)			5	Paddle B (Y-Pos)	
6	Feuerknopf 1 / LightPen			6	Feuerknopf 2 / LightPen	
7	+5V			7	+5V	
8	GND (Masse)			8	GND (Masse)	
9	Paddle A (X-Pos)			9	Paddle B (X-Pos)	



Richtung	**Joystick 1**	**Joystick 2**
oben	PB0	PA0
unten	PB1	PA1
links	PB2	PA2
rechts	PB3	PA3
Feuerknopf	PB4	PA4

Natürlich ist es auch möglich, mehrere Richtungen gleichzeitig abzufragen. Wird der Joystick z.B. nach rechts oben gedrückt, so sind die Portleitungen für "oben" und "rechts"(PB0 und PB3 für Joyport1, bzw. PA0 und PA3 für Joyport2) auf 1, d.h. die Bits 0 und 3 im jeweiligen Datenregister sind gelöscht! Achten Sie für solche Fälle also immer darauf, daß Ihre Joystickabfrage dynamisch ist und mehrere Richtungen auch erkennen kann. Ich möchte da mit gutem Beispiel vorangehen und habe Ihnen einmal eine Abfrage des Joyport1 als Beispielprogramm programmiert. Sie finden es, wie immer, als Source-Code (mit ".SRC"-Extension) und als ausführbares Maschinenprogramm (mit ".OBJ"-Extension) unter dem Namen "JOYTEST" auf dieser MD.
Ich habe deshalb den Joyport1 gewählt, weil wir bei ihm das Datenrichtungsregister nicht zu ändern brauchen. Er läuft ja über Port B der CIA1) und wie wir aus dem letzten Kursteil wissen, ist dieser schon vom Betriebssystem her komplett auf "Eingang" geschaltet. Natürlich können Sie das auch bei Joyport2 tun (erscheint in Port A der CIA1), jedoch müssen Sie dabei berücksichtigen, daß Sie ihn nach der Abfrage wieder auf "Ausgang" schalten, weil sonst die Tastatur nicht mehr ansprechbar ist!
Kommen wir nun aber zu dem Beispielprogramm. Es steht ab $C000(= dez.49152) und wird auch dort gestartet (SYS49152).

```
****************************************
        start    LDA #01      Zeichenfarbe auf
                 STA 646      "weiß" setzen.
        loop1    LDA #147     Code für "CLR" laden
                 JSR$FFD2     und ausgeben.
                 LDA $DC01    Datenport B laden.
                 LSR          "oben"-Bit in Carry.
                 BCS l1       Gesetzt, also nix "oben".
                 JSR pup      Gelöscht, also "oben" ausgeben.
        l1       LSR          "unten"-Bit in Carry.
                 BCS l2       Gesetzt, also nix "unten".
                 JSR pdown    Gelöscht, also "unten" ausgeben.
        l2       LSR          "links"- Bit in Carry.
                 BCS l3       Gesetzt, also nix "links".
                 JSR pleft    Gelöscht, also "links" ausgeben.
        l3       LSR          "rechts"-Bit in Carry.
                 BCS l4       Gesetzt, also nix "rechts".
                 JSR prigh    Gelöscht, also "rechts" ausgeben.
        l4       LSR          "Knopf"-Bit in Carry.
                 BCS l6       Gesetzt, kein Knopf gedrückt.
                 LDA #02      Sonst, Farbe "rot" in Akku.
                 BNE l5       Unbedingt verzweigen
        l6       LDA #00      Kein Knopf, also Farbe " schwarz" in Akku.
        l5       STA $ D020   und Bildschirmfarbe sta $ d021 setzen.
                 JMP loop1    Schleife wiederholen
****************************************
```

JOYTEST tut nun nichts anderes, als eine Schleife zu durchlaufen, die zunächst den Bildschirm löscht, dann prüft, welche der Joystickrichtungen gedrückt sind und den dazu passenden Text "oben","unten","links" oder "rechts" ausgibt.
Zusätzlich berücksichtigt sie dabei, daß zwei Richtungen gleichzeitig aktiv sein können und gibt auch dementsprechende Texte aus ("oben links","unten rechts", etc...). Wird der Feuerknopf gedrückt, so erscheint der ganze Bildschirm rot. Beachten Sie bitte, daß auch in diesem Fall alle 8 Joystickrichtungen abgefragt werden! Die Funktionsweise der Routine ist so simpel, daß sie keiner großen Erklärung bedarf. Am Anfang der Schleife wird der Inhalt des Datenports B in den Akku geholt und nun Bit für Bit nach rechts herausgeschoben. Dabei gelangen nacheinander die fünf Joystick-Bits in das Carrybit, von wo aus man prüfen kann, ob die einzelnen Bits nun gesetzt, oder gelöscht sind. Ist eines der Bits gelöscht, so wird eine entsprechende Routine aufgerufen, die einen passenden Text ausgibt ("pup","pdown","pleft","

prigh"). Diese Routinen habe ich hier nicht aufgeführt, jedoch können Sie sie sich im Source-Code ja einmal anschauen.
Soviel zur Joystickabfrage. Wie Sie jedoch sicherlich in der Grafik von oben gesehen haben, gibt es noch weitere Anschlüsse am Joyport, über die man gewisse Geräte betreiben kann. Diese sind zum einen die Paddles und zum anderen der Light-Pen. Von beiden Eingabegeräten haben Sie bestimmt schon einmal gehört.
Ich möchte Ihnen nun kurz erläutern, wie sie Funktionieren und wie man sie abfragen kann. Ein Paddle, ist ein Eingabegerät, bei dem prinzipiell nur zwei Werte übertragen werden, nämlich die X- und die Y-Position eines Grafikcursors. Sicherlich erinnern Sie sich noch an diese kleinen "Magic-Tables", wie man sie als Kind oft gehabt hat. Mit Hilfe zweier Drehknöpfe konnte man da auf einen grauen Glasschirm malen, wobei man mit dem einen Knopf den Zeichenstift nach links und rechts bewegte, mit dem anderen nach oben und unten. So in etwa kann man sich auch Paddles vorstellen, wobei ein Paddle einem der beiden Drehknöpfe entspricht. Grundsätzlich können (oder müssen) also ZWEI Paddles an EINEM Joyport angeschlossen werden, wobei diese aufgeteilt werden in X- und Y-Richtungspaddle (Pin 9 und 5 am Joyport). Wie kann man nun aber eine Position von nur einem Pin ablesen? Nun das ist ganz pfiffig: ein Paddle ist nämlich nichts anderes als ein Potentiometer, wie man es aus der Elektronik kennt. Also ein Stufenlos verstellbarer elektrischer Widerstand, der je nach Drehrichtung größer oder kleiner wird. Er wird über das Potentiometer in den Eingang an Pin 5 oder 9 eingeleitet. Diese Pins sind nun mit dem SID, dem Soundchip des 64 ers, verbunden, der über zwei Analog/ Digital-Wandler verfügt. Da Widerstand eine analoge Größe ist (er kann unendlich fein aufgelöst werden), muß er zur Verarbeitung mit dem Rechner erst in einen digitalen Wert gewandelt werden, was über jene A/ D-Wandler geschieht. Sie haben je eine Auflösung von 8 Bit und legen den digitalen Wert in den Registern 25 und 26(Adressen 54297 und 54298) des SID ab. Durch Auslesen der Werte dieser Register erhalten wir einen Digitalwert des Widerstandes des Potentiometers, wobei wir 256 verschiedene " Positionen" unterscheiden können. Zu beachten ist jedoch, daß die A/ D-Wandler des SID nur einen bestimmten Bereich abtasten können, nämlich von 200 Ohm (Wert 0) bis 200000 Ohm (=200 Kiloohm, Wert 255).
Zum Lightpen gibt es nicht viel zu sagen. Er kann ja ebenfalls am Joyport angeschlossen werden, wobei dies ausschließlich nur bei Joyport2 der Fall ist. Pin 6 dieses Ports, an dem normalerweise der Joystickfeuerknopf hängt, ist für den Lightpen zuständig. Zur Abfrage eines Lightpens sollte man aber gewisse Grundkenntnisse über den VIC und den Bildschirmaufbau an sich haben. Ein Lightpen ist im Prinzip nichts anderes, als eine einfache und schlichte Fotozelle, wie man sie im Fachhandel für wenig Geld erstehen kann. Sie ist in der Lage, Licht, das auf sie einfällt zu registrieren und in diesem Fall einen Strom zu erzeugen. Dieser Strom nun wird an Pin 6 von Joyport2 angelegt und veranlaßt somit ein Löschen des Bits 4 vom Datenport A (dasselbe Bit, wie für den Feuerknopf) . Dieses Ereignis tritt genau dann ein, wenn der Rasterstrahl des Monitors ganz genau an der Stelle des Bildschirms vorbeifährt, an dem der Lightpen positioniert ist. In dem Fall muß nun ein pfiffiges Programm feststellen, an genau welcher Position sich der Rasterstrahl nun befindet um die Position des Lightpens zu ermitteln. Dabei kann einem der VIC helfen, der ein solches Lightpen-Signal als Interruptquelle vorgesehen hat, jedoch müssen Sie berücksichtigen, daß Sie über den VIC lediglich die aktuelle Rasterzeile, nicht aber die Rasterspalte abfragen können. Die muß man kleinlich berechnen, was nur geht, wenn man weiß, wie lange es dauert, bis der Rasterstrahl eine Zeile gezeichnet hat, und vor allen Dingen wann er damit begonnen hat. Da das alles sehr schnell geht (25 Mal pro Sekunde läuft der Rasterstrahl über den GESAMTEN Bildschirm), bekommt man meist sehr ungenaue Ergebnisse, was ein hin- und herspringen des Grafikcursors bewirkt. Obwohl einige Lightpens für den 64 er schon auf dem Markt waren, hat sich diese Eingabeart auf unserem Rechner wohl nie so richtig durchgesetzt. Schade eigentlich, aber wenn Sie wollen, können Sie es ja einmal versuchen, das nötige Wissen dazu sollten Sie jetzt ja haben. Wie man mit Raster-Interrupts richtig umgeht, sollte Ihnen mein Kollege Ivo Herzeg, dessen Kurs vor diesem hier lief, hinreichend erklärt haben.

In diesem Sinne möchte ich mich nun wieder von Ihnen verabschieden. Nächsten Monat geht es, ab dann wieder in gewohnter Länge, um eine Mausabfrage. Ich habe Ihnen als Leckerbissen ein Programm vorbereitet, mit dem Sie eine AMIGA-Maus am 64 er anschließen und betreiben können.
Des Weiteren wollen wir uns dann auch noch ein wenig mit dem Userport befassen. Bis dahin Servus, Ihr Uli Basters (ub) age. Wie Sie jedoch sicherlich in der Grafik von oben gesehen haben, gibt es noch weitere Anschlüße.

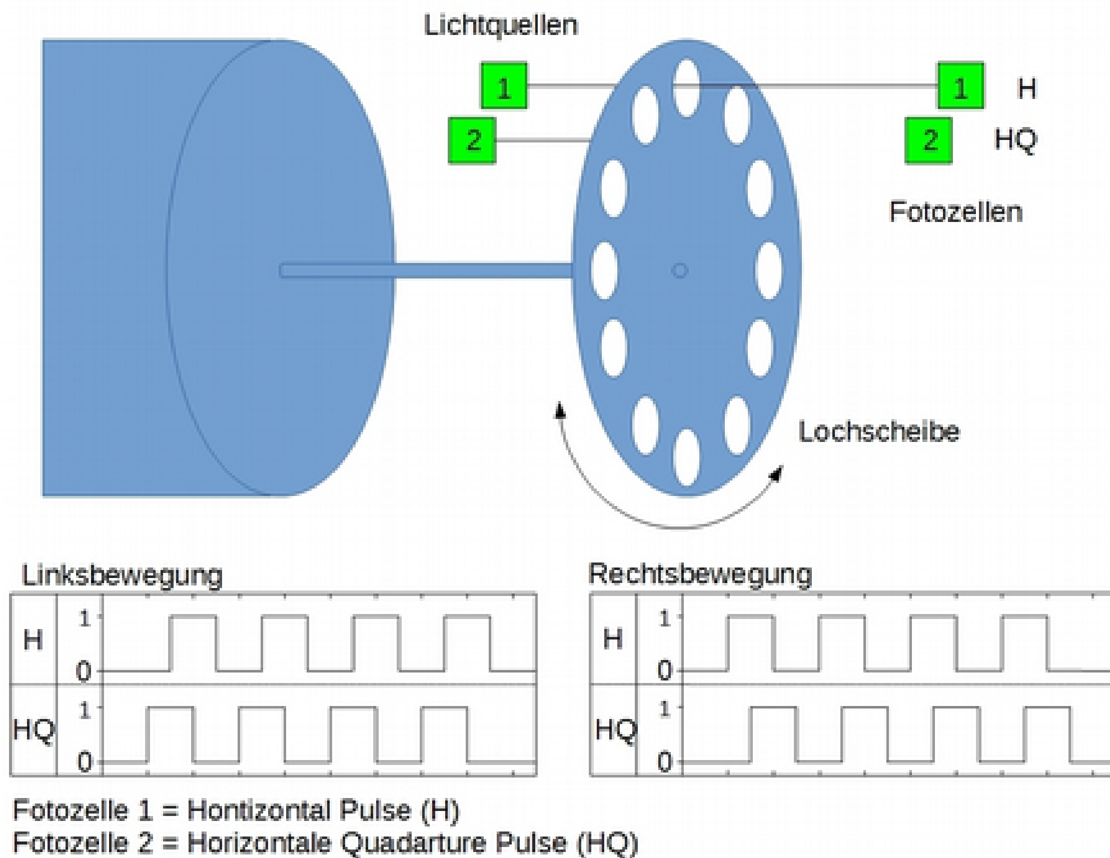
Teil 9 – Magic Disk 08/91

Hallo zusammen zum 9. Teil des CIA-Kurses. Wie schon versprochen, wollen wir uns heute mit einer "echten" Mausabfrage befassen.
Die Maus - sicher haben Sie schon vieles von diesem Eingabegerät gehört, durch das die einfache und komfortable Benutzung eines Computers erst richtig möglich wurde. Die "neuen" Rechner wie AMIGA, ATARI ST, oder der MAC werden standardmässig mit diesen kleinen viereckigen Kästchen ausgeliefert. Auch das vielgerühmte C64- System GEOS prahlt damit, über Maus-Steuerung benutzbar zu sein. Doch hier wollen wir gleich einmal eine Unterscheidung machen:

1. Die Maussteuerung von GEOS ist nichts besonderes. Für GEOS ist die Maus nichts anderes als ein "getarnter" Joystick. Bewegt man sie, so werden absolut dieselben Signale erzeugt, wie der Joystick sie liefert, womit GEOS-Mäuse eigentlich unbrauchbar sind, da sie nicht die Vorteile von "echten" Mäusen bieten.
2. 2) Die " Echten", wie sie an den oben genannten Rechnern zu finden sind, sind nämlich nur deshalb so gut, weil sie jede Bewegung, die die Hand des Benutzers vollzieht absolut naturgetreu wiedergeben. Das heißt im Klartext, daß wenn die Maus nur um ein paar Millimeter bewegt wird, sich auch der Mauscursor nur um ein paar Pixel über den Bildschirm bewegt; wird die Maus jedoch um mehrere Zentimeter bewegt, so bewegt sich auch der Mauscursor um eine äquivalent größere Anzahl von Pixeln weiter

Damit wird es also möglich mit Hilfe einer passenden Maus Bilder naturgetreu abzuzeichnen. Des Weiteren bewegt sich der Mauscursor auch immer mit der Geschwindigkeit, mit der die Hand die Maus bewegt. Wenn Sie die Maus schnell bewegen, so bewegt sich der Mauscursor auch schnell; und ebenso bewegt er sich langsam, wenn die Maus langsam bewegt wird. Eine " falsche"Maus bewegt sich immer nur so schnell, wie sie abgefragt wird - egal wieviel Meter Sie sie "über den Tisch ziehen".
Uns soll es hier nun um den zweiten Typ von Mäusen gehen. Um eine Abfrage zu programmieren, sollten Sie zunächst einmal wissen, wie eine "echte" Maus funktioniert. Dazu möchte ich Ihnen nun erklären, welche "Hardware" in einer Maus so drinsteckt: Jede Maus verfügt an der Unterseite über eine, aus dem Gehäuse herausschauende, Stahlkugel, die zur besseren Haftung mit einer Gummischicht überzogen ist. Diese Kugel kann man über eine Klappe zu Reinigungszwecken entfernen. Sieht man nun in die Aussparung hinein, so erkennt man dort zwei kleine Walzen, auf die die Bewegungen der Maus übertragen werden. Die Walzen stehen in einem rechten Winken zueinander, so daß die horizontale und die vertikale Richtung der Maus erfasst wird. Je nach dem, wie stark die Maus nun in eine Richtung bewegt wird, drehen sich auch die Walzen mehr oder weniger schnell, wobei bei Schrägbewegungen die Bewegung durch die Stahlkugel immer in die horizontale und die vertikale Bewegungsrichtung aufgespalten wird.
An jeder der Achsen der beiden Walzen sind nun kleine Lochscheiben angebracht, die sich mit der Walze drehen. An jeder Lochscheibe wird über eine Lichtschranke festgestellt, ob die Maus in einer der beiden Bewegungsachsen bewegt wird. Jedesmal, wenn ein Loch in der Scheibe an der Lichtschranke vorbeifährt, wird ein Impuls an den Rechner gegeben, der so erkennt, daß die Maus in der entsprechenden Bewegungsachse bewegt wurde. Je höher die Frequenz

dieser Impulse ist, desto schneller war die Bewegung.



Nun wissen wir also, daß eine Bewegung stattfand, nicht aber, in welche Richtung bewegt wurde, weil die Impulse ja in beiden möglichen Richtungen dieselben sind. Um nun die richtige Richtung herauszufinden, befindet sich noch eine zweite Lichtschranke an jeder Lochscheibe. Beide Schranken sind um ein halbes Loch voneinander versetzt, so daß immer eine der beiden Schranken vor der anderen ein Signal liefert. Je nach dem, welche der Schranken zuerst einen Impuls gibt, wird die Maus in die eine, oder in die andere Richtung bewegt (bei der horizontalen Achse nach links oder rechts, bei der vertikalen Achse nach oben oder unten). Zur besseren Erläuterung hier einmal eine Grafik. Das Signal "H" ist das Signal der ersten Lichtschranke."H" steht für "Horitontal Pulse" und zeigt dem Rechner, daß überhaupt eine Bewegung in der Horizontalen Achse geschieht. Die zweite Lichtschranke erzeugt das "HQ"-Signal, was für „Horizontal Quadrature Pulse" steht. Durch dieses Signal kann die Bewegungsrichtung festgestellt werden. Ich habe mich bei der Grafik auf die horizontale Bewegungsachse beschränkt. Bei der vertikalen Achse ist der Ablauf jedoch derselbe, nur daß hier die Signale " V" und "VQ" heißen.
Zunächst habe ich Ihnen den Aufbau der Mausmechanik einmal aufgezeichnet. Sie sehen, wie durch die Versetzung um ein halbes Loch, die zweite Lichtschranke, die das HQ-Signal erzeugt, in dieser Position der Lochscheibe noch keinen Impuls liefert, während die erste Lichtschranke schon aktiv ist. Beachten Sie bitte auch die Oszillator-Darstellung der Signale im unteren Bereich des Bildes. Soviel also zum Aufbau der Mausmechanik.
Nun wollen wir uns einmal überlegen, was wir zu einer Mausabfrage beachten müssen. Zunächst einmal wollen wir die einzelnen Zustände der Maussignale untersuchen um herauszufinden, in welche Richtung die Maus bewegt wird. Hierbei beschränke ich mich wieder auf die Horizontalbewegung, da der Ablauf für die Vertikalbewegung identisch ist.
Betrachtet man einmal die Oszillatorkurven in der Grafik oben und " übersetzt" man sie in Bitmuster (0 für "low",1 für "High"), dann erhält man für die einzelnen Bewegungen folgende

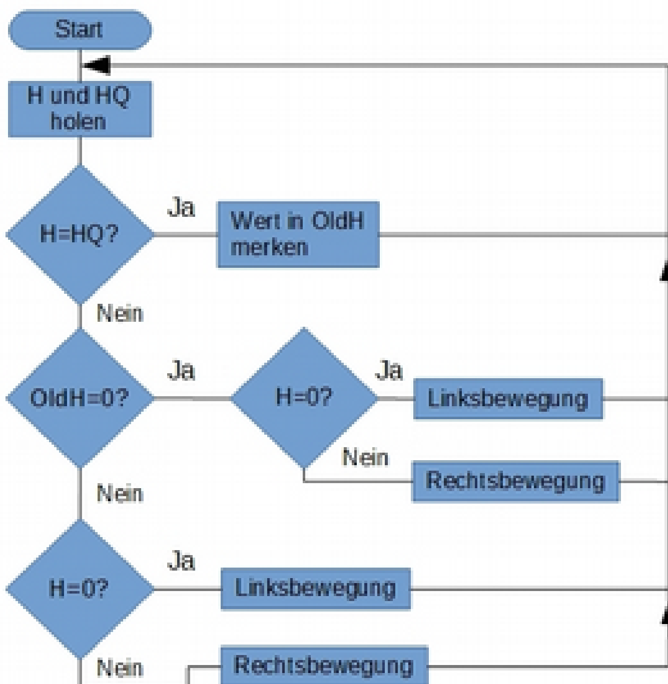
Bilder:

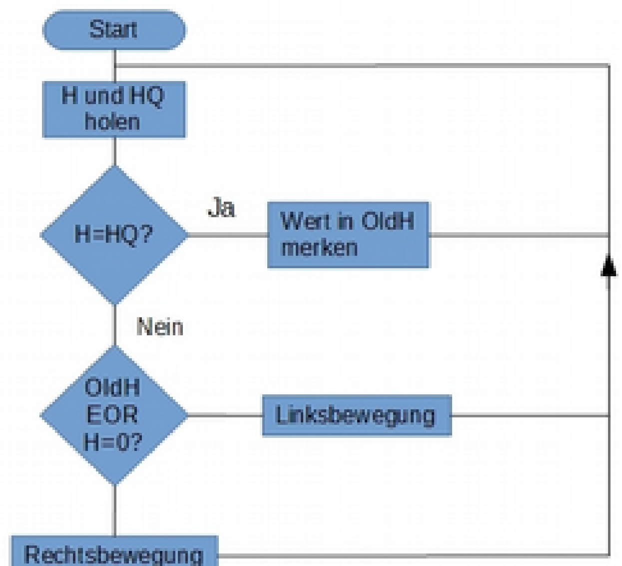
Linksbewegung:	H	-	...0011 0011...
	HQ	-	0110 0110
Rechtsbewegung:	H	-	...01100110…
	HQ	-	00110011

Nach jeweils 4 Signalen wiederholt sich das Ganze immer wieder. Wie Sie sehen kommen in beiden Bewegungen die H/HQ-Kombinationen 00,01,10 und 11 vor. Die Unterschiede der beiden Signale, an denen wir dann die Richtung erkennen können sind folgende:

1. Ist die Bedingung H= HQ=0 erfüllt, und sind sind folgenden Signale H=0 und HQ=1, so fand eine Bewegung nach LINKS statt.
2. Ist die Bedingung H= HQ=0 erfüllt, und sind die folgenden Signale H=1 und HQ=0(also genau umgekehrt), so fand eine Bewegung nach RECHTS statt.
3. Ist die Bedingung H= HQ=1 erfüllt, und sind die folgenden Signale H=1 und HQ=0, so fand eine Bewegung nach LINKS statt.
4. Ist die Bedingung H= HQ=1 erfüllt, und sind die folgenden Signale H=0 und HQ=1, dann fand eine Bewegung nach RECHTS statt.

Zur genauen Bestimmung der Richtung müssen wir also ZWEI zeitlich voneinander versetzte Signale GLEICHZEITIG auswerten! Das kann unter Umständen sehr kompliziert werden, da mehrere Vergleiche ineinander verschachtelt werden müssen, um die genaue Richtung herauszufinden.
Aus diesem Grund habe ich eine sinnvolle Vereinfachung durchgeführt, die durch folgende Grafik erklärt werden soll:





Durch eine Exklusiv-Oder- Verknüpfung der Signale Old H (entspricht dem alten Wert von H, wobei H=HQ war) und H erhält man folgende Aussagetabelle:

OldH	H	Ergebnis
0	0	0
0	1	1
1	0	1
1	1	0

Vergleichen wir nun diese Werte mit den oben aufgezählten Richtungsbedingungen, so sieht man, daß das Ergebnis immer dann 0 wird, wenn eine Linksbewegung durchgeführt werden muß, und ebenso daß es immer 1 wird, wenn eine Rechtsbewegung durchzuführen ist.
Dadurch haben wir uns eine Menge Vergleichsbefehle gespart. Wir müssen jetzt lediglich das Ergebnis der EOR-Operation prüfen, um die Bewegungsrichtung zu bestimmen. Entsprechend wird mit dem Vertikal-Signal verfahren.
Kommen wir nun zu der fertigen Steuerroutine. Hierbei müssen wir auf vier wichtige Dinge achten:

1. Das Programm zur Mausabfrage MUSS in jedem Fall in der Hauptschleife des Rechners laufen, da sich die Signale der Maus jederzeit ändern können.
   Eine zyklische Abfrage aus dem IRQ, so wie das z. B. beim Joystick möglich ist, wäre also undenkbar, da sie unter Umständen das eine oder andere Signal verpassen würde und somit die Interpretation desselben von unserer Mausabfrage falsch sein könnte. Die Maus würde ruckeln und sich gegebenenfalls sogar in die umgekehrte Richtung bewegen.
2. Des Weiteren sollte ich Sie darauf aufmerksam machen, daß die Tastatur in der Zeit, in der die Maus angeschlossen ist nicht zu benutzen ist, da sie IMMER ein Signal sendet (es sei denn V, VQ, H und HQ sind zufällig gerade alle auf 0) und somit die Tastaturabfrage stört (wie wir wissen benutzt diese ja die selben Datenregister der CIA wie die Joyports).Wenn Sie also das Beispielprogramm "AMIGA-MAUS1" auf dieser MD ausprobieren wollen, sollten Sie immer daran denken, das Programm ZUERST zu laden und mit RUN zu starten und dann erst die Maus einzustecken!
3. Durch die ständige Abfrage müssen wir ebenfalls daran denken, das Maussignal zu " entprellen" . Das heißt, daß wir prüfen müssen, ob die Maus nun gerade bewegt wird oder nicht. In jedem Fall werden wir nämlich ein und dasselbe Signal mehrmals lesen. Für uns sind jedoch immer nur die Signaländerungen von Bedeutung, weshalb wir prüfen müssen, ob das neu gelesene Signal nun gleich, oder verschieden vom letzten Signal ist. Bei Gleichheit wird der Wert einfach ignoriert und wieder zur Leseschleife zurückverzweigt.
4. Als Letztes will ich Ihnen nun noch die Belegung der Maussignale erklären. Die Maus hat ja, wie der Joystick auch, einen 9- poligen Gameportstecker, dessen Signale sich glücklicherweise so auf die 9 Pins verteilen, daß wir sie (fast) alle über die CIA abfragen können.

Die Belegung ist hierbei wie folgt:

Pin	Funktion	Pin	Funktion
1	V-Impluse	6	Knopf 1 (links)
2	H-Impulse	7	+5 Volt (Betriebsspannung)
3	VQ-Impulse	8	GND (Masse)
4	HQ-Impulse	9	Knopf 2 (rechts)
5	Knopf 3 (unbenutzt, da nicht vor-handen)		

Die Pins 1-4 und 6 der Joyports sind ja, wie Sie aus dem letzten Kursteil noch wissen sollten, mit den Bits 0-4 der Datenports der CIA1 verbunden. Demnach können wir also problemlos die Richtung der Maus abfragen, sowie die linke Maustaste. Die rechte Maustaste hängt an einem der A/ D-Wandler des SID und soll uns deshalb hier nicht interessieren.
Der Einfachheit halber habe ich die Abfrage über Joyport2 programmiert, die Signale der Maus erscheinen also im Datenregister A der CIA1(Reg.0-Adresse $DC00) . Dies ist deshalb einfacher, weil dieser Port schon von der Tastaturabfrage her auf "Eingang" geschaltet ist. Die benötigten Signale finden wir an folgenden Bitpositionen im Datenregister A:

Bit	Signal
0	V
1	H
2	VQ
3	HQ
4	Linke Maustaste

Kommen wir nun zu unserem Beispielprogramm "AMIGA-MAUS1", das zusammen mit seinem Quellcode ("AMIGA-MAUS1.SRC") auf dieser MD zu finden ist. Starten Sie es bitte mit RUN und schließen Sie dann eine original AMIGA-Maus am Joyport2 an um den Mauspfeil über den Bildschirm zu bewegen. Mit der linken Maustaste wird das Programm abgebrochen. Hier nun die Dokumentation des Programms:

```
****************************************
        start       LDA #00          Bildschirm-
                    STA 53280        farben
                    LDA #11          auf schwarz/grau
                    STA 53281        setzen.
                    LDA #<(text)     Text ab Label
                    LDY #>(text)     "TEXT"
                    JSR strout       ausgeben.
                    LDA #01          Spritefarbe auf
                    STA vic+39       weiß setzen.
                    LDA #150         Sprite in den
                    STA vic          Bildschirm
                    STA vic+1        positionieren.
                    LDA #36          Sprite-Pointer
                    STA 2040         setzen.
                    LDA #01          Und Sprite 0
                    STA vic+21       einschalten
****************************************
        loop2       LDA $DC00        Joyport2 lesen
                    EOR #$FF         und invertieren
                    AND #$1F         Bits 0-4 isolieren
                    CMP #16          Vergleiche mit 1
                    BCC l1           Wenn kleiner,dann Bewegung
                    RTS              Sonst wurde Knop gedrückt, also Ende
        ***
        l1          LDX #00          Alle
                    STX h            Puls-
                    STX hq           Register
                    STX v            werden
                    STX vq           gelöscht
                    LSR              V-Bit in V-Reg
                    ROL v            hineinrollen
                    LSR              H-Bit in H-Reg
                    ROL h            hineinrollen
```

```
            LSR             VQ-Bit in VQ-Reg
            ROL vq          hineinrollen
            LSR             HQ-Bit in HQ-Reg
            ROL hq          hineinrollen
****************************************
   horizon  LDA h           H mit HQ
            EOR hq          verknüpfen
            CMP hmem        und prüfen, ob gleich altem Wert
            BEQ vertical    Ja, also entprellen
            STA hmem        Sonst merken und
            CMP #00         prüfen, ob =0
            BNE l2          Nein, also Bewegung ausführen
            LDA h           Sonst war H=HQ, deshalb Bitmuster
            STA oldh        in OLDH merken
            JMP vertical    und weiter
   ***
   l2       LDA oldh        OLDH und H verknüpfen,
            EOR h           um die Richtung herauszufinden
            BNE l7          =1, also Rechts!
            JMP moveleft    =0, also Links!
   l7       JMP moveright
****************************************
   vertical LDA v           V mit VQ
            EOR vq          verknüpfen
            CMP vmem        und prüfen, ob gleich altem Wer
            BEQ loop2       Ja, also entprellen
            STA vmem        Sonst merken und
            CMP #00         prüfen, ob =0
            BNE l8          Nein, also Bewegung ausführen
            LDA v           Sonst war V=VQ, deshalb Bitmuster
            STA oldv        in OLDV merken
            JMP loop2       und zurück
   ***
   l8       LDA oldv        OLDV und V verknüpfen
            EOR v           um die Richtung herauszufinden
            BNE l9          =1, also Hoch!
            JMP moveup      =0, also Runter!
   L9       JMP movedown
****************************************
```

Zum besseren Verständnis noch einig Erklärungen zum Source-Code:

1. Die Routinen MOVELEFT, MOVERIGHT, MOVEUP und MOVEDOWN sind Unterroutinen die das Sprite 0, das den Mauspfeil repräsentiert, entsprechend der geforderten Richtung bewegen. Ich habe sie hier nicht aufgeführt, da sie nichts mit der CIA zu tun haben. Sie können Sie sich jedoch im Source-Code von "AMIGA-MAUS1" ansehen.
2. Die Labels H, HQ, V, VQ, OLDH, OLDV HMEM, und VMEM stehen für Speicherzellen, in denen bestimmte Werte zwischengespeichert werden.
3. Beim Vergleich, ob H gleich HQ (bzw. gleich VQ) ist, habe ich ebenfalls die EOR-Verknüpfung benutzt, da so das Entprellen der Maussignale vereinfacht wird. HMEM (bzw. VMEM) ändern sich immer abwechselnd von 0 auf 1 und um gekehrt.
4. Ab dem Label "TEXT" steht der Text der beim Aufruf von "AMIGA-MAUS1" ausgedruckt wird.
5. Beachten Sie bitte, daß die Eingangswerte immer invertiert werden, da die CIA sie uns invertiert im Datenregister angibt (sollten Sie noch vom letztem Kursteil her wissen) Des Weiteren wird der Wert dann noch AND-Verknüpft um die Bits 0-4 zu isolieren. Dadurch wird es möglich die Maustaste durch einen einfachen Vergleich mit dem Wert 16 abzufragen. Sonst würde das Programm nämlich auch bei dem Druck auf eine Taste, die

di Bits über dem 4. Bit setzt abbrechen!

Das war es dann mal wieder für diese Monat. Behalten Sie die AMIGA-Maus, je doch noch bis nächsten Monat. Wir werde uns dann um den Userport kümmern und ein Mausabfrage programmieren, die die Nach teile der oben aufgeführten (nämlich da die Tastatur unbenutzbar ist, und da nicht aus dem Interrupt heraus abgefragt werden kann) verbessert. Bis dahin, wünsche ich ein "funny Mousing-Around",

Ihr Uli Basters (ub)

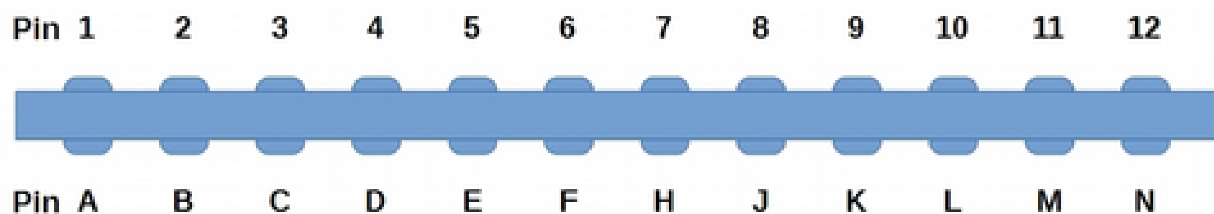
Teil 10 – Magic Disk 09/91

Herzlich willkommen zum 10. und letzten Teil dieses Kurses. Wir wollen uns heute weiterhin mit der Mausabfrage beschäftigen, wobei wir in Zusammenhang mit dem Userport - einem der wichtigsten Themen, wenn es um die CIA geht - die Abfrage von letztem Monat noch verbessern wollen. Also los…
Wie Sie sich bestimmt noch erinnern, hatten wir in der letzten MD eine Mausabfrage für eine AMIGA-Maus am Joyport programmiert. Hardwaremäßig war das auch am einfachsten, da wir die Maus direkt an den 64 er anschließen konnten. Doch es ergaben sich aber auch diverse Nachteile:

1. Solange die Maus angeschlossen war, war es uns nicht möglich die Tastatur zu benutzen.
2. Eine Abfrage aus dem Interrupt war nicht möglich, da die Maussignale ständig überwacht und ausgewertet werden mußten.
3. Die rechte Maustaste konnte leider nicht Abgefragt werden.

Diese drei Nachteile wollen wir nun elegant beseitigen, indem wir die Maus nicht am Joyport, sondern am Userport anschließen. Sicher haben Sie schon einmal von diesem Anschluß Gebrauch gemacht, ist er doch der vielseitigste von allen Anschlüssen am C64. Drucker, Modems, Eprommer, oder Digitizer werden über ihn angeschlossen und bedient - kurzum, ohne ihn wäre der 64 er nicht das was er ist! Deshalb soll er uns nun interessieren. Am Userport liegt eine Vielzahl von Signalen und internen Leitungen an, die es uns ermöglichen, direkt in die Hardware unseres Rechners eingreifen zu können. Und gerade weil fast alle Leitungen des Userports mit den beiden CIAs etwas zu tun haben, passt er hervorragend in diesen Kurs. Kommen wir also zum Aufbau, dieser Unscheinbaren Schnittstelle, an der Rückseite unseres „kleinen Brotkastens".
Insgesamt 24 Leitungen sind dort herausgeführt, die alle eine bestimmte Bedeutung haben. Hierzu gibt es jetzt erst einmal eine Grafik:

Die Belegung des Userports (Ansicht von hinten auf den C64)



Pin	Belegung		Pin	Belegung	
1	GND (Masse)		A	GND	
2	+5V (Gleichstrom)		B	**FLAG2**	
3	**RESET**		C	PB0	Portleitungen von Port B

4	CNT1		D	PB1	
5	SP1		E	PB2	
6	CNT2		F	PB3	
7	SP32		H	PB4	der CIA2
8	<u>**PC2**</u>		J	PB5	
9	Serieller ATN Pin		K	PB6	
10	9 V AC	9 Volt Wechselspannung	L	PB7	
11	9 V AC		M	PA2	Portleitung 2, Port A, CIA2
12	GND		N	GND	

Fett und unterstrichen entspricht invertierten Signal!

Ich möchte Ihnen nun die einzelnen Signale genauer erklären:

- Wie Sie sehen ist Masse (GND) viermal am Userport zu finden nämlich an den Pins 1,12, A und N. Auf diese Weise ist sichergestellt, daß man immer an den äußeren Pins den Gegenpol einer Gleichspannung findet.
- Die oben genannte Gleichspannung sind wohl in der Regel die +5 Volt an Pin 2, die man als Betriebsspannung für Hardwareerweiterungen benutzen kann. Die dort herausgeführte Leitung ist mit maximal 100 mA belastbar.
- Pin 3 enthält die RESET-Leitung des Rechners. Durch sie kann der gesamte C64 wieder in den Einschaltzustand zurückversetzt werden. Da das Signal invertiert ist (Fett / Unterstrichen "RESET"), muß kein Pegel angelegt werden (+5 V), um den Reset auszulösen, sondern man muß diesen Pin mit Masse verbinden.
- Die Leitungen CNT1(Pin4) und CNT2 (Pin6) sind die CNT-Leitungen von CIA1 und CIA2. Wir haben diese Leitungen schon bei den Timerinterrupts kennengelernt, da man sie zur Timersteuerung heranziehen kann. Sie spielen eine große Rolle bei unserer Mausabfrage nachher.
- Die Leitungen SP1(Pin 5) und SP2 (Pin 7) sind die SP-Leitungen von CIA1 und CIA2. Diese Leitungen werden zur interaktiven seriellen Datenübertragung benutzt und sollen uns ebenfalls nachher noch beschäftigen.
- Die Leitungen PC2 (Pin 8) und FLAG2 (Pin B) sind negierte Signale (deshalb auch der Strich über den beiden Worten). Sie sind die Handshake-Leitungen von CIA2. Ein "Handshake" ist wichtig, um bei einer Datenübertragung eine Verbindung korrekt aufzubauen. PC2 ist dabei der Handshake-Ausgang, FLAG2 der Handshake-Eingang.
- Die Leitung "ATN IN" ist ein hier ebenfalls zu findendes Signal vom IEC-Bus, an dem die Floppy angeschlossen ist. Dieses Signal stammt von der Portleitung 3, Port A (PA3) der CIA2.
- An den Pins 10 und 11 liegt eine Wechselspannung von 9 Volt an. Da es sich um Wechselspannung handelt benötigt man natürlich 2 Anschlüsse (Wechselspannung wird im Gegensatz zu Gleichspannung nicht mit Masse am Gegenpol angeschlossen).
- Die Pins C-L entsprechen dem gesamten Port B der CIA2. Dadurch wird es möglich eine "echte" parallele Datenübertragung zu realisieren (CENTRONICS), wie sie von vielen Druckern verlangt wird. Gleichzeitig dienen diese Leitungen dem Datenaustausch verschiedenster Art zwischen dem C64 und einer angeschlossenen Hardware.Auch wir werden diese Leitungen später benutzen.
- Pin M beinhaltet ebenfalls eine Portleitung, nämlich das 2. Bit von Port A. Auch diese Leitung kann als I/ O- Leitung genutzt werden (z.B. für Steuersignale einer Parallelschnittstelle)

Soviel also zu den Leitungen am Userport. Wie Sie sehen bedient Hauptsächlich die CIA2 den Datenverkehr an dieser Schnittstelle. Von CIA1 finden wir nur zwei Leitungen (CNT1 und SP1).

Doch gerade diese Leitungen, zusammen mit den äquivalenten Leitungen von CIA2 (CNT2 und SP2) spielen bei unserer Mausabfrage später noch eine große Rolle.
Kommen wir nun also zu der Abfrage selbst. Wie versprochen, wollen wir ja die drei oben genannten Nachteile unterbinden. Um es einmal langsam anzugehen wollen wir zunächst einmal die Tastatur wieder benutzbar machen. Wie Sie ja nun wissen, sind die Portleitungen der CIA2 am Userport herausgeführt. Diese sind glücklicherweise vom Betriebssystem ungenutzt, weshalb wir an diesen Leitungen getrost Signale einleiten können, ohne dabei den die "normale" Rechnerumgebung zu stören. Da die Tastaturabfrage von CIA1 erledigt wird stören die am Userport angeschlossenen Maussignale sie also in keinster Weise. Zudem können wir haargenau dieselbe Abfrageroutine wie beim letzten Mal benutzen, jedoch mit dem Unterschied, daß sie sich nun die Mausdaten aus Port B von CIA2 holen muß. Wir müssen also lediglich einen Befehl des Programms von letztem Monat ändern, nämlich das ehemalige "LDA $DC00" in ein "LDA $DD01" und schon funktioniert die Abfrage!
Zusätzlich brauchen wir jetzt natürlich noch einen Adapterstecker, der jedoch einfach nachzubauen ist. Alles was Sie dazu brauchen ist:

1. Ein Userportstecker
2. Eine Joyportbuchse
3. Etwas Kabel
4. und ein kleines bisschen Erfahrung mit dem Lötkolben.

Bis auf Punkt 4 ist alles für ein paar Mark im Elektronikfachhandel erhältlich. Wenn Sie alles zusammen haben, verbinden Sie bitte Pins der beiden Stecker wie folgt:

Joystickport	Userport	Signal
1 ⇨	C	Vertical Pulse
2 ⇨	D	Horizontal Pulse
3 ⇨	E	Vertical Quadrature
4 ⇨	F	Horizontal Quadrature
6 ⇨	H	Linker Mausbutton
7 ⇨	2	Betriebsspannung +5V
8 ⇨	1	GND
9 ⇨	J	Rechter Mausbutton

Die Signalnamen sollten Sie noch aus letztem Kursteil kennen. Achten Sie beim Löten bitte darauf, daß Sie die richtige Symmetrie benutzen. Beim Userport sind die Leitungen nämlich, wenn man von vorne auf den STECKER schaut genau spiegelverkehrt (Pins 1-12 und A-N von rechts nach links). Schauen Sie hinten auf die Lötpins des Steckers, so stimmt die Belegung wieder, so wie Sie in der obigen Grafik aufgeführt war.
Ähnliches gilt für die Joyportbuchse - wenn Sie von vorne auf sie draufsehen ist Pin 1 links oben und Pin 9 rechts unten. Von hinten ist Pin 1 rechts oben und Pin 9 links unten. In der Regel sollten aber beide Stecker auch mit Pinnummern versehen sein, so daß man keine Fehler machen kann. Achten Sie aber bitte doch darauf und prüfen Sie vor dem ersten Anschluß noch einmal alle Pins nach, da Sie sich bei einer falschen Belegung durchaus den Rechner kaputt machen können!!! Wir übernehmen in diesem Fall keinerlei Ersatzansprüche!
Worauf Sie ebenfalls achten sollten ist das sie eine Joyport-BUCHSE benötigen - nicht etwa einen STECKER. Das heißt, daß das Ding, welches Sie sich kaufen genau denen entsprechen muß, die an den Joyports des C64 herausschauen!
Wenn dann alles geklappt hat können Sie Ihren Adapterstecker ausprobieren. Stecken Sie ihn am Userport, bei abgeschaltetem Rechner, ein (bitte wieder darauf achten, daß Sie ihn nicht verkehrt hineinstecken, da sonst derselbe Effekt wie oben auftritt) und schließen Sie eine

orginal AMIGA-Maus an der anderen Seite an. Wie Sie sehen, können Sie die Tastatur immer noch ganz normal benutzen. Laden Sie nun das Programm "AMIGA-MAUS2" von dieser MD und starten Sie es mit RUN. Sie können jetzt den Mauspfeil über den Bildschirm bewegen. Das Programm kann mit der linken UND - im Gegensatz zu unserer alten Abfrage - mit der rechten Maustaste abgebrochen werden. Damit hätten wir also schon einmal zwei Nachteile beseitigt, nämlich der, daß die Tastatur unbenutzbar war und der daß die rechte Maustaste nicht abgefragt werden konnte. Letztere hatten wir ja beim Bau des Adaptersteckers mit Pin J des Userports verbunden, womit ihr Signal an der Leitung PB5 von CIA2 anliegt. Diese Leitung entspricht nun dem 5. Bit von PortB der CIA2, womit wir die Taste problemlos abfragen könnten! Falls es Sie interessiert - der Source-Code von "AMIGA-MAUS2" ist ebenfalls auf dieser MD unter dem Namen "AMIGA-MAUS2.SRC" enthalten. Er entspricht jedoch, abgesehen von der oben beschriebenen Änderung haargenau dem von "AMIGA-MAUS1". Nun gut - über den Userport haben wir die Maus und die Tastatur vollständig benutzbar gemacht, jedoch ist es schon hinderlich, wenn die Abfrage immer in der Hauptschleife des Rechners läuft. So muß bei einem Programm, das mit der Maus bedient wird, bei jeder einzelnen Routine, die etwas anderes tun soll als die Maus abzufragen, die Abfrage unterbrochen werden. Und das ist verbunden mit hohem organisatorischem Aufwand.
Deshalb wollen wir nun versuchen, eine Abfrage über Interrupts zu programmieren. Es IST möglich - ich habe mir da eine pfiffige Routine für Sie ausgedacht.
Wie ich oben schon erwähnte sollen dabei die Leitungen CNT1 und CNT2 am Userport eine wichtige Rolle spielen. Vielleicht erinnern Sie sich ja noch an die ersten Teile des CIA-Kurses, in denen ich Ihnen die Timerprogrammierung erklärte. Damals hatten wir festgestellt, daß die Timer der CIAs Interrupts auslösen können. Bei CIA1 waren das IRQs, bei CIA2 die NMIs. Dabei konnte man verschiedene Ereignisse als Timertrigger einstellen. Im Regelfall war das der Systemtakt; bei der Timerkopplung war es der Unterlauf von Timer A. Und nun kommts: wir konnten ebenso ein Signal an der CNT-Leitung einer CIA als Timertrigger einstellen. Und genau das ist der Punkt an dem wir ansetzen wollen. Genauer gesagt wird ein Timer, wenn er die CNT-Leitung als Timertrigger programmiert hat, immer dann um 1 heruntergezählt, wenn an der CNT-Leitung eine steigende Flanke anliegt. Wenn also der Pegel an dieser Leitung gerade von 0 auf 1 umspringt. Danach nicht mehr, solange die Leitung auch auf 1 liegen bleiben sollte. Sie muß erst wieder auf 0 abfallen, damit der Timer durch eine neue Flanke ein weiteres Mal erniedrigt werden kann. Vergleichen wir das einmal mit den Signalen, die uns die Maus liefert (letzten Monat hatte ich das ja genauer erklärt), so stellen wir fest, daß die Maussignale auch immer von 0 auf 1 wechseln, egal ob es nun das H-, HQ-, Voder VQ-Signal ist. Wir können also diese Signale als Timertrigger verwenden, und zwar so, daß der Timer immer alle Flanken der Maus mitzählt. Dadurch können wir genau erfahren, um wieviele Einheiten die Maus bewegt wurde! Das einzige Problem dabei ist die Richtungsbestimmung, weil ja in beiden Richtungen dieselben Signale erzeugt werden (beschränkt man sich auf nur EIN Signal, z.B. das H-Signal). Das heißt, daß wir bei JEDER Bewegung, die stattfindet, auch das entsprechende Quadrature-Signal überprüfen müssen um die genaue Richtung bestimmen zu können. Das reine Zählen der Impulse nutzt uns also nichts, dennoch reicht es zum Auslösen eines Interrupts bei jeder Bewegung.
Wenn wir einen Timer nämlich mit dem Wert 0 initialisieren, genügt ein einziger Impuls von der CNT-Leitung, um einen Interrupt auszulösen. Dieser muß nun feststellen, welche Bewegungsrichtung ausgeführt wurde. Der Witz ist, daß diese Abfrage sogar noch einfacher ist, als die von der ersten und zweiten Version von AMIGA-MAUS. Weil wir nämlich nicht zwei zeitlich voneinander versetzte Signale überprüfen müssen. Zur besseren Erläuterung will ich Ihnen noch einmal die Signalfolgen der Bewegungsrichtungen auflisten:

Linksbewegung:	H	-	... 1 1 0 0 1 1 0 0 ...
	HQ	-	1 0 0 1 1 0 0 1
			🠝 🠝
			Interrrupt
Rechntsbewegung:	H	-	… 1 1 0 0 1 1 0 0 ...
	HQ	-	… 0 1 1 0 0 1 1 0
			🠝 🠝
			Interrupt

Gehen wir nun davon aus, daß wir das H-Signal an der CNT-Leitung anliegen haben, und daß einer der Timer der dazugehörigen CIA diese Leitung als Trigger hat und zudem mit 0 initialisiert ist, so wird jedesmal, wenn das H-Signal auf 1 springt ein Interrupt ausgelöst, weil der Timer unterläuft. Diese Stellen habe ich in obiger Auflistung markiert ("🠝"). Wenn Sie genauer hinsehen, so erkennen Sie, daß das HQ-Signal bei einer Linksbewegung zum Zeitpunkt des Interrupts immer 1, bei einer Rechtsbewegung immer 0 ist. Um nun die Bewegungsrichtung zu bestimmen brauchen wir lediglich das HQ-Signal an einem der Porteingänge von PortB am Userport anzuschließen und bei Auftreten eines Interrupts auszulesen. Ist es 0, so müssen wir den Mauspfeil nach rechts bewegen, ist es 1, so muß der Mauspfeil nach links. Wir brauchen also noch nicht einmal das letzte Signal zur Bestimmung heranzuziehen. Ebenso wird übrigens mit den vertikalen Signalen verfahren. Hierbei liegt das V-Signal dann an der anderen CNT-Leitung an. Für unser Beispiel habe ich die Belegungen wievolgt belegt:

- V-Signal an CNT1- löst also IRQs aus (weil an CIA1).
- H-Signal an CNT2- löst also NMIs aus (weil an CIA2).
- * VQ und HQ wie bei Adapterstecker1 Bei den V-Signalen müssen wir noch etwas beachten: Da der Systeminterrupt über einen Systemtaktgetriggerten Timer A läuft, und wir diesen ja weiterbenutzen möchten, müssen wir die Auswertung des V-Signals über Timer B programmieren.

Wenn jetzt ein IRQ auftritt, so müssen wir erst anhand der gesetzten Bits im "Interrupt-Control-Register" (ICR) feststellen, welcher Timer der Interruptauslöser war. War es Timer A, so wird auf den Systeminterrupt weiterverzweigt, war es Timer B, so wird auf die Abfrage der Vertikalbewegung gesprungen. Ähnlich verhält sich dies bei den H-Signalen. Weil wir die Maustasten ja ebenfalls noch Abfragen wollen, müssen wir den zweiten freien Timer der NMI-CIA zyklische Interrupts auslösen lassen, die in regelmäßigen Abständen die Maustasten abfragen. Der Einfachheit halber habe ich die Horizontalbewegungsabfrage ebenfalls über Timer B der CIA2 programmiert, weshalb wir also Timer A zur Abfrage der Maustasten benutzen wollen.
Kommen wir nun zu dem Programm selbst. Hier ist der kommentierte Source-Code:

```
****************************************
start    SEI            IRQs sperren
         LDX #<(irq)    IRQ-Vektor
         LDY #>(irq)    auf neue
         STX $0314      IRQ-Routine
         STY $0315      verbiegen.
         LDX #<(nmi)    NMI-Vektor
         LDY #>(nmi)    auf neue
         STX $0318      NMI-Routine
         STY $0319      verbiegen.
         LDX #$83       Timer A und B als
         STX cia1+13    Interruptquelle für
         STX cia2+13    beide CIAs setzen
         LDA #00        Timer B von
```

```
                STA cia1+6          CIA1
                STA cia1+7          und
                STA cia2+6          CIA2 mit dem Wert
                STA cia2+7          0 initialisieren.
                LDY #$90            Timer A von CIA2
                STA cia2+4          initialisieren
                STY cia2+5          ($9000=27 IRQs/s)
                LDA #$21            Trigger=CNT und "Timer Start"
                STA cia1+15         in Control-Regist
                STA cia2+15         für Timer B von CIA1 und CIA2.
                LDA #$81            Timer A von CIA2
                STA cia2+14         mit SysTakt als Trigger starten
                LDA #00             Bildschirm-
                STA 53280           farben
                LDA #11             setzen
                STA 53281           und
                LDA #<(text)        Begrüßungs-
                LDY #>(text)        Text
                JSR strout          ausgeben.
                LDA #01             Sprite 0
                STA vic+39          als
                STA vic+21          Maus-
                LDA #150            pfeil
                STA vic             ini-
                STA vic+1           tiali-
                LDA #41             sieren.
                STA 2040
                CLI                 IRQs freigeben
                RTS                 und ENDE.
****************************************
    nmi         PHA                 Alle
                TXA                 Prozessorregister
                PHA                 erstmal
                TYA                 auf Stapel
                PHA                 retten.
                CLI                 IRQs freigeben.
                LDA $DD01           Datenport sporadis
                EOR #$FF            laden und invertie
                STA mem             merken.
                LDA cia2+13         ICR von CIA2 holen
                AND #$02            Bit 1 isolieren
                BEQ buttons         Wenn =0, dann war Timer A der NMI- Auslöser, also Knopfabfrage.
                LDA mem             Sonst H-Bewegung..
                AND #$08            Bit für HQ-Signal aus Datenport isolieren
                BEQ moveleft        Wenn 0 war ->links
                BNE moveright       Wenn 1 war ->rechts
    buttons     LDA mem             Aus Datenport die
                AND #$30            Bits 4 und 5 (Mausknöpfe) isolieren
                BEQ bye             Wenn =0, dann war einer gedrückt.
                AND #$20            Sonst Bit5 isolier
                BEQ leftone         Wenn =0, war der linke gedrückt
                LDA #42             Spritepointer
                LDX #01             und Timerwert laden
    l8          STA 2040            und setzen.
                STX cia1+6          (Timerwert in BEI
                STX cia2+6          CIAs!)
                JMP bye             NMI beenden.
    Leftone     LDA #41             Spritepointer und
                LDX #00             Timerwert für link
                JMP l8              Taste setzen
****************************************
    irq         LDA cia1+13         ICR von CIA1 laden
                AND #$02            Bit 1 isolieren
```

```
             BNE ok              Wenn =1, dann wars ein Maus-IRQ
             JMP sysirq          Sonst auf SysIRQ springen
ok           CLI                 IRQs freigeben
             LDA $DD01           Datenport laden und
             EOR #$FF            invertieren.
             AND #$04            Bit für VQ-Signal isolieren
             BEQ moveup          Wenn 0 war -> hoch
             BNE movedown        Wenn 1 war -> runter
```

****

Hier nun noch einige Erläuterungen:

1. Wie Sie sehen, müssen wir bei NMIs den Prozessorregister "von Hand" auf den Stapel retten. Bei NMIs geschieht dies nicht durch eine Routine im Betriebssystem, so wie es bei den IRQs der Fall war.
2. Nachdem eine der beiden Interruptroutinen aufgerufen wurde, wird so früh wie möglich das IRQ-Flag wieder gelöscht und die IRQs zugelassen. Das ist deshalb so wichtig, weil die IRQs beim Auftreten eines Interrupts (sei das ein IRQ oder ein NMI) gesperrt werde. Befindet sich nun aber der Rechner gerade in einem NMI, während die Maus der Vertikalen bewegt wird, so wird kein IRQ ausgelöst, weil dieser ja noch gesperrt ist. Um das doch noch zu ermöglichen müssen wir den IRQ explizit freigeben. Bei NMIs brauchen wir darauf nicht zu achten, weil NMIs ja nicht maskierbar sind. Das heißt, daß ein NMI immer einen NMI unterbrechen kann!
3. Denken Sie bitte nach wie vor daran daß die Daten der Datenports invertiert in den CIA-Registern stehen. Wir müssen sie beim Lesen deshalb gleich nochmal invertieren. Das ist gerade bei der Maustastenabfrage sehr wichtig, weshalb der Datenport in jedem Fall erst einmal invertiert wird, bevor eine Entscheidung getroffen wird, woher der NMI überhaupt kam.
   Bei der Vertikalabfrage, hätte ich den Datenport nicht unbedingt invertieren müssen. Da hier lediglich nur ein einziges Bit abgefragt wird, hätte ich die Branchbefehle zu den Bewegungsroutinen ebenso vertauschen können. Der Vollständigkeit halber wird hier der Wert jedoch ebenfalls invertiert.
4. Sicher hat Sie die merkwürdige Mausbutton abfrage etwas verwirrt. Ich habe hier noch eine kleine Funktion eingebaut die durch die Art unserer Abfrage ganz einfach zu programmieren wurde.
   Zunächst einmal wird durch den Druck auf einen der Mausknöpfe der Spritpointer zwischen den Spriteblöcken 00 und 42 hinund hergeschaltet. Dies nur, um Ihnen die Mausbuttonabfrage optisch zu signalisieren. Zusätzlich wird, je nach dem welchen Knopf sie noch drücken, ein anderer Wert in die Timer-Register geschrieben. Drücken Sie die linke Taste, so ist das der Wert 0, wie wir es für die Abfrage ja schon vereinbart hatten. Drücken Sie jedoch die rechte Maustaste, so wird der Wert 1 in die Timer geladen. Durch diesen kleinen, aber effektiven Trick können wir die Mausauflösung halbieren. Dadurch, daß nun 2 Impulse von der Maus kommen müssen, bis ein Interrupt auftritt müssen Sie die Maus doppelt weit über den Tisch bewegen, um den Mauspfeil eine bestimmte Strecke weg zu bewegen. Das kann oftmals ganz nützlich sein, z.B. wenn man genau zeichnen muß.
   Wenn Sie übrigens den Wert 2 in die Timer schreiben, so verkleinert sich die Auflösung um ein Drittel und so fort…
5. Achten Sie bitte auch auf die Abfragen in den Interruptroutinen, von welchen Quelle der Interrupt kam. Im ICR sind dann nämlich die entsprechenden Bits die auch beim Einstellen der Interrupquellen benutzt werden gesetzt. So können wir also unterscheiden, von wo ein Interrupt kam.
6. Die Routinen MOVELEFT, MOVERIGHT, MOVE-UP und MOVEDOWN sind Routinen die den Mauspfeil bewegen und sollen hier nicht näher erläutert werden. Sie können Sie sich aber gerne im Source-Listing "AMIGA-MAUS3.SRC" auf dieser MD anschauen 7) Das

Label ENDIRQ enthüllt die Adresse $EA7E. Ab dieser Adresse stehen im Betriebssystem die Befehle, mit denen jeder Interrupt (auch NMIs) beendet werden. Es werden einfach die Prozessorregister wieder vom Stapel geholt.
Um unsere eigenen Interrupts zu beenden springe ich also der Einfachheit halber gleich diese Adresse an.

Natürlich brauchen Sie zum Betrieb der neuen Mausabfrage einen neuen Adapterstecker. Damit das nicht allzu umständlich wird, habe ich die Belegungen im Großen und Ganzen so gelassen wie sie beim erst Adapterstecker waren. Sie müssen lediglich zwei Leitungen umlöten:

- Das V-Signal (Pin 1 an der Joyportbuchse) kommt jetzt an CNT1(= Pin 4 am Userport - vorher Pin C).
- Das H-Signal (Pin 2 an der Joyportbuchse) kommt jetzt an CNT2 (= Pin 6 am Userport - vorher Pin D).

Schließen Sie den neuen Stecker nun bei abgeschaltetem 64 er am Userport an, und stecken Sie eine AMIGA-Maus am anderen Ende ein. Jetzt können Sie das Programm "AMIGA-MAUS3" von dieser MD laden und mit RUN starten. Wie Sie sehen, können Sie nun auch weiterhin Eingaben machen, da der Cursor weiterhin blinkt. Denken Sie nun daran, daß es bei Diskettenzugriffen Probleme geben wird, da die NMIs den Datenverkehr stören(sollten Sie noch aus dem ersten Teilen dieses Kurses wissen). In solchen Fällen ist es ratsam die Abfrage doch abzuschalten (z.B. indem man die Timer einfach anhält, oder sie als Interruptquellen sperrt).
Zum Schluß möchte ich Ihnen noch eine weitere Funktion der CIAs erklären. Nur mit dem Userport erhält sie überhaupt einen Sinn, weshalb ich sie bis jetzt auslassen mußte.
Vielleicht erinnern Sie sich noch daran, daß das Register 12 einer CIA das sogenannte "Serial-Data-Register" ist. Mit diesem Register kann über die SP-Leitung (die ja von beiden CIAs am Userport anliegt) ein einfacher serieller Datenaustausch mit einem anderen Rechner (im einfachsten Fall ebenfalls ein 64' er stattfinden. Das kann sehr nützlich sein wenn man z.B. ein Spiel programmieren möchte, bei dem zwei Spieler an jeweils einem eigenen Rechner gegeneinander spielen Wenn das Spielprogramm also Daten über die Tätigkeiten und Bewegungen des anderen Spielers benötigt. Über das SD-Register wird dieser Datenaustausch stark vereinfacht und ist fast noch unkomplizierter, als wenn man sich der normal Seriellen Schnittstelle (RS232) des Betriebssystems bedient.
Bei der Datenübertragung müssen wir unterscheiden, ob die SP-Leitung nun auf Ein oder Ausgang geschaltet ist. Dies wird mit Bit 6 des Control-Registers- Timer A (Reg.14) angegeben. Ist es auf 1, so ist SP auf Ausgang, ist es auf 0, so ist SP auf Eingang geschaltet. Je nach Betriebsart verhält sich die Benutzung von SDR wie folgt:

- Wenn SP Ausgang ist, so wird ein Wert, der in das SDR geschrieben wird mit der halben Unterlauffrequenz von Timer A in den entsprechenden CIA an SP " herausgesch ben" . Das heißt, daß der Wert Bit für Bit, bei jedem Timerunter lauf an SP e scheint. Nach 8 Unterläufen ist SDR wi der leer und es wird ein Interrupt ausgelöst. Bit3 im ICR zeigt an, daß der Interrupt von dem leeren SDR-Regist kommt.
- Wenn SP Eingang ist, so wird mit jeder steigenden Flanke an CNT der entsprechende CIA der Wert, der gerade anliegt (0 oder 1) in ein internes Schieberegister übernommen. Ist dies Mal ge- schehen, so wird der Wert in das SDR übertragen und ebenfalls ein Interrupt ausgelöst. Hier erkennt man ebefalls an Bit 3 im ICR, daß das SDR voll ist und ausgelesen werden kann.

Kombiniert man das Ganze jetzt noch mit der Möglichkeit, daß die CIA bei einem Timerunterlauf ein Signal an PB6 anlegen kann, so kann man sehr einfach Daten austauschen. Möchten Sie z. B. Daten an einen anderen C64 senden, so müssen Sie sich ein Kabel bauen, daß die Userporte der beiden Rechner miteinander verbindet.
Dabei sollte der SP-Ausgang von Rechnern mit dem SP-Eingang von Rechner 2 verbunden

sein und die Leitung PB6 von Recher1 mit der Leitung CNT von Rechner2 (ob SP1 oder SP2, bzw. CNT1 oder CNT2 hängt davon ab mit welcher CIA sie die Daten übertragen wollen). Jetzt müssen Sie im Control-Register von Timer A (des sendenden Rechners) noch festlegen, daß das Signal an PB6 bei jedem Timerunterlauf in die jeweils andere Lage gekippt werden soll. Das geschieht, indem Sie die Bits 1 und 2 dieses Registers auf 1 setzen. Dadurch erscheint nämlich ebenfalls mit der halben Unterlauffrequenz von Timer A (des sendenden Rechners) ein Signal an PB6 und läst somit die Datenübernahme am empfangenden Rechner aus!
Das war es dann endgültig mit dem CIA-Kurs. Ich hoffe, daß Sie nun einen besseren Einblick in die Funktionen dieser beiden kleinen, aber extrem mächtigen Bausteine innerhalb unseres Rechners haben. Wie Sie sehen lassen sich sehr viele Probleme mit Interrupts leichter lösen (wie die Mausabfrage beweist). Zusätzlich können Sie mit dem Userport vielfältige Hardware-erweiterungen leicht und einfach bedienen.
Ich freue mich also, wenn es Ihnen ein wenig Spaß gemacht hat und bin für Kritik und Anregungen zu neuen Kursen immer zu haben (auch ein kleiner Kursautor kriegt gerne Leserpost).
Bis auf Weiteres also ein letztes Mal "Servus"

Ihr Uli Basters (ub)

Sourcecodes

Magic Disk 01/91 - Teil 3

Listing „MSGOUT-ROM.SRC“

```
10      .BA $8000
11      .EQ SYSIRQ=$EA31
12      .EQ COUNTER=$FB
13      .EQ PULSE=$FC
14      .EQ MSGMODE=$FD
80      ;********************************
100     MSGOUT          SEI
105                     STA PULSE
106                     STX LOOP1+1
107                     STY LOOP1+2
110                     LDX #<(NEWIRQ)
120                     LDY #>(NEWIRQ)
130                     STX $0314
140                     STY $0315
150 ;
160                     LDA #01
170                     STA COUNTER
190                     STA MSGMODE
200                     CLI
210                     RTS
220     ;********************************
230     NEWIRQ          DEC COUNTER
240                     BEQ L1
250                     JMP SYSIRQ
260 ;
270     L1              LDA MSGMODE
280                     BNE L2
290                     INC MSGMODE
300                     JSR DOMSG
310                     JMP PREPNEW
320 ;
330     L2              DEC MSGMODE
340                     JSR BLANKLIN
350 ;
360     END             DEC PULSE
370                     BPL PREPNEW
380                     LDX #<(SYSIRQ)
390                     LDY #>(SYSIRQ)
400                     STX $0314
410                     STY $0315
420                     JMP SYSIRQ
430 ;
440     PREPNEW         LDA #30
450                     STA COUNTER
460                     JMP SYSIRQ
470 ;********************************
480     DOMSG LDY#00
490     LOOP1           LDA$C000,Y
495                     BEQL3
500                     STA$07C0,Y
510                     INY
520                     BNELOOP1;UNBED
550 ;********************************
560     BLANKLIN        LDY#39
570                     LDA#32
580     LOOP2           STA$07C0,Y
590                     DEY
600                     BPL LOOP2
610     L3              RTS
```

Listing „MSGOUT-RAM.SRC“

```
10      .BA $8000
12      .EQ COUNTER=$FB
13      .EQ PULSE=$FC
14      .EQ MSGMODE=$FD
80      ;********************************
100     MSGOUT          SEI
105                     STA PULSE
106                     STX LOOP1+1
107                     STY LOOP1+2
110                     LDX #<(NEWIRQ)
120                     LDY #>(NEWIRQ)
130                     STX $FFFE
140                     STY $FFFF
141 ;
142                     LDA #$35
143                     STA $01
150 ;
160                     LDA #01
170                     STA COUNTER
180                     LDA #00
190                     STA MSGMODE
200                     CLI
210 ;
211     LOOP3           LDA $01
212                     CMP #$37
213                     BNELOOP3
214                     RTS
220     ;********************************
221     NEWIRQ          PHA
222                     TXA
223                     PHA
224                     TYA
225                     PHA
226 ;
230                     DEC COUNTER
240                     BEQL1
250                     JMPSYSIRQ
260 ;
270     L1              LDA MSGMODE
280                     BEQL2
290                     INC MSGMODE
300                     JSR DOMSG
310                     JMP PREPNEW
320 ;
330     L2              DEC MSGMODE
340                     JSR BLANKLIN
350 ;
360     END             DEC PULSE
370                     BPLPREPNEW
380                     LDA #$37
390                     STA $01
420                     JMP SYSIRQ
430 ;
440     PREPNEW         LDA #30
450                     STA COUNTER
455     ;********************************
460     SYSIRQ LDA$DC0D
461                     PLA
462                     TAY
463                     PLA
464                     TAX
465                     PLA
```

```
466                 RTI
470     ;*********************************
480     DOMSG LDY #00
490     LOOP1       LDA $C000,Y
495                 BEQ L3
500                 STA $07C0,Y
510                 INY
520                 BNE LOOP1;UNBED
550     ;*********************************
560     BLANKLIN    LDY #39
570                 LDA #32
580     LOOP2       STA $07C0,Y
590                 DEY
600                 BPL LOOP2
610     L3          RTS
```

Magic Disk 03/91 – Teil 5

Listing „EVAL.SRC“

```
10      .BA $9000
11      .EQ CIA2 = $DD00
12      .EQ TXTOUT = $AB1E
13      .EQ NUM = $62
14      .EQ FAC2ASC = $BDDD
15      .EQ BSOUT = $FFD2
80      ;*********************************
100     START       LDA #$7F
110                 STA CIA2+13
111                 LDA #00
112                 STA CIA2+14
113                 STA CIA2+15
120 ;
130                 LDA #$FF
140                 STA CIA2+4
150                 STA CIA2+5
160                 STA CIA2+6
170                 STA CIA2+7
185 ;
210                 LDA #$41
220                 STA CIA2+15
230                 LDA #$81
240                 STA CIA2+14
250 ;
260                 JSR $8000
270 ;
280                 LDA #$80
290                 STA CIA2+14
300                 STA CIA2+15
310 ;
320                 LDY #03
321                 LDX #00
330     LOOP1       LDA CIA2+4,Y
335                 EOR #$FF
340                 STA NUM,X
345                 INX
350                 DEY
360                 BPL LOOP1
370 ;
390                 SEC
400                 SBC #12
410                 BCS L1
420                 DEC NUM+2
425 ;
430                 LDX NUM+2
440                 CPX #$FF
450                 BNE L1
455 ;
```

```
460                 DEC NUM+1
465                 LDX NUM+1
470                 CPX #$FF
480                 BNE L1
490 ;
500                 DEC NUM+0
510 ;
520     L1          STA NUM+3
530 ;
535                 LDA #$A0
536                 STA $61
540     LOOP2       LDA $62
550                 BMI L2
555                 DEC $61
560                 ASL NUM+3
570                 ROL NUM+2
580                 ROL NUM+1
590                 ROL NUM
600                 BCC LOOP2           ;UNBED
610 ;
620     L2          JSR FAC2ASC
630                 LDA #<(TXT1)
640                 LDY #>(TXT1)
650                 JSR TXTOUT
660 ;
670                 LDA #00
680                 LDY #01
690                 JSR TXTOUT
695                 LDA#13
696                 JMP BSOUT
700     ;*********************************
710     TXT1        .TX"TAKTZYKLEN: "
720                 .BY0
```

Magic Disk 04/91

Listing „CLOCK.SRC“

```
10      .BA $0B40
11      .EQ V=$D000
12      .EQ IRR=V+25
13      .EQ IMR=V+26
14      .EQ IRQALT=$EA31
15      .EQ IRQ1=$F9
16      .EQ IRQ2=$FE
17      .EQ ZEILE=V+18
18      .EQ CIA1=$DC00
19      .EQ SPRPOI=2040
20      .EQ SPRBAS=33
21      .EQ RETTEN=$02
22      .EQ MEM=828
23      .EQ STROUT=$AB1E
24      .EQ BASIN=$FFCF
25      .EQ MODE=$03
26      .EQ SID=$D400
80      ;*********************************
90      INIT        LDA #$81
91                  STA CIA1+14
92 ;
100                 LDA #<(TXT1)
110                 LDY #>(TXT1)
120                 LDX #$00
130                 JSR SETIT
140 ;
150                 LDA #<(TXT2)
160                 LDY #>(TXT2)
170                 LDX #$80
180                 JSR SETIT
190 ;
```

```
200                     SEI
210                     LDX #<(NEWIRQ)
220                     LDY #>(NEWIRQ)
230                     STX $0314
240                     STY $0315
250 ;
260                     LDX #15
270     LOOP1           LDA XTAB,X
280                     STA V,X
290                     DEX
300                     BPL LOOP1
310 ;
320                     LDX #07
330                     LDA #01
340     LOOP2           STA V+39,X
350                     DEX
360                     BPL LOOP2
370 ;
380                     LDA     #$85
390                     STA CIA1+13
400                     LDA #15
410                     STA SID+5
420                     LDA #$C0
430                     STA SID+1
440 ;
450                     LDX #<(END+1)
460                     LDY #>(END+1)
470                     STX $2B
480                     STY $2C
490                     JSR $A642
500 ;
510                     CLI
520                     RTS
530     ;********************************
540     NEWIRQ          LDA #$7F
550                     LDX MODE
560                     BNE L5
570                     LDA #$FF
580     L5              STA V+21
590 ;
600                     LDA CIA1+13
610                     AND #$04
620                     BEQ TIMEREF
630                     LDA #15
640                     STA SID+24
650                     LDA #33
660                     STAS ID+4
670                     LDX #$FF
680     LOOP5           DEY
690                     BNE LOOP5
700                     DEX
710                     BNE LOOP5
720 ;
730                     LDA #00
740                     STA SID+4
750                     STA SID+24
760 ;
770     TIMEREF         LDX #43
780                     LDA CIA1+11
790                     BMI L6
800                     CMP #$12
810                     BNE L2
820                     LDA #00
830                     BEQ L2              ;UNBED
840 ;
850     L6              INX
860                     AND #$7F
870                     LDY MODE
880                     BEQ L2
890                     CMP#$12
```

```
900                     BEQ L3
910                     CMP #00
920                     BEQ L3
930                     SED
940                     CLC
950                     ADC #$12
960                     CLD
970 ;
980     L2              STX SPRPOI+7
990 ;
1000    L3              JSR GETNUM
1010                    STX SPRPOI+0
1020                    STA SPRPOI+1
1030 ;
1040                    LDA CIA1+10
1050                    JSR GETNUM
1060                    STX SPRPOI+2
1070                    STA SPRPOI+3
1080 ;
1090                    LDA CIA1+9
1100                    JSR GETNUM
1110                    STX SPRPOI+4
1120                    STA SPRPOI+5
1130 ;
1140                    LDA CIA1+8
1150                    CLC
1160                    ADC #SPRBAS
1170                    STA SPRPOI+6
1180                    JMP $EA31
1190    ;********************************
1200    GETNUM          PHA
1210                    LSR
1220                    LSR
1230                    LSR
1240                    LSR
1250                    CLC
1260                    ADC #SPRBAS
1270                    TAX
1280 ;
1290                    PLA
1300                    AND #$0F
1310                    CLC
1320                    ADC #SPRBAS
1330                    RTS
1340    ;********************************
1350    XTAB            .BY30,230,46,230,62,230,78,230,94,230,110,230,140,230,160,230
1360    ;********************************
1370    SETIT           STX CIA1+15
1380                    JSR STROUT
1390 ;
1400                    LDY #00
1410    LOOP3           JSR BASIN
1420                    CMP #13
1430                    BEQ L1
1440                    STA MEM,Y
1450                    INY
1460                    BNE LOOP3
1470 ;
1480    L1              LDX #$FF
1490                    LDA MEM
1500                    LDY MEM+1
1510                    JSR CONVERT
1520                    PHA
1530                    INX
1540 ;
1550                    LDA MEM+2
1560                    LDY MEM+3
1570                    JSR CONVERT
1580                    PHA
1590 ;
```

```
1600              LDA MEM+4
1610              LDY MEM+5
1620              JSR CONVERT
1630              TAY
1640              PLA
1650              TAX
1660              PLA
1670              STA CIA1+11
1680              STX CIA1+10
1690              STY CIA1+9
1700              LDA #00
1710              STA CIA1+8
1720              RTS
1730    ;*********************************
1740    CONVERT   SEC
1750              SBC #48
1760              ASL
1770              ASL
1780              ASL
1790              ASL
1800              STA RETTEN
1810 ;
1820              TYA
1830              SEC
1840              SBC #48
1850              ORA RETTEN
1860 ;
1870              CPX #$FF
1880              BNE L4
1890              SED
1900              CMP #$13
1910              BCC L4
1920              SBC #$12
1930              CLD
1940              ORA #$80
1950 ;
1960    L4        CLD
1970              RTS
1980    ;*********************************
1990    TXT1      .TX"<Shift+CLR/Home><CTRL+N>Clock V1.0 by Uli Basters."
2000              .BY13,13
2010              .TX"Dieses Programm demonstriert die Echt"
2020              .BY13
2030              .TX"zeituhr der CIA-Bausteine"
2040              .BY13
2050              .TX"Eingabe bitte in der Form! ´HHMMSS´ !"
2060              .BY13,13
2070              .TX"Uhrzeit   : "
2080              .BY0
2090    TXT2      .BY13
2100              .TX"Alarmzeit : "
2110    END       .BY0
```

<Shift+CLR/Home> = Steuercode für Bildschirm löschen
<CTRL+N> = Steuercode für Kleinschreibung

Magic Disk 05/91

Listing „WAITSHIFT.SRC“

```
10      .BA $C000
11      .EQ CIA1=$DC00
80      ;*********************************
100      X        SEI
110 ;
120               LDA #$42
130               STA CIA1+2
140               LDA #$00
150               STA CIA1+3
```

```
180 ;
190    LOOP1       LDA#$FC
200                STA CIA1+0
210                LDA CIA1+1
220                CMP #$7F
230                BEQ END
240 ;
250                LDA #$BF
260                STA CIA1+0
270                LDA CIA1+1
280                CMP #$EF
290                BNE LOOP1
300 ;
310    END         LDA #$FF
320                STA CIA1+2
330                CLI
340                RTS
```

Magic Disk 08/91

Listing „AMIGA-MAUS1.SRC“

```
10     .BA $9000
11     .EQ STROUT=$AB1E
12     .EQ OLDH=$02
13     .EQ H=$03
14     .EQ HQ=$04
15     .EQ V=$05
16     .EQ VQ=$06
17     .EQ LEFTSTOP=24
18     .EQ RIGHTSTOP=87
19     .EQ OLDINPUT=$0A
20     .EQ HMEM=$F9
21     .EQ VMEM=$FA
22     .EQ OLDV=$FB
23     .EQ UPSTOP=50
24     .EQ DOWNSTOP=249
30     .EQ VIC=$D000
31     .EQ TEXT=$093F
80     ;********************************
100    MOVELEFT    LDA VIC
110                BNE L3
120                STA VIC+16
130                BEQL4
140    L3          CMP #LEFTSTOP
150                BNE L4
160                LDA VIC+16
170                BEQ LOOP2
180 ;
190    L4          DEC VIC
200                JMP LOOP2
210    ;***
220    MOVERIGHT   LDA VIC
230                CMP #$FF
240                BNE L5
250                INC VIC+16
260                BNE L6          ;UNBED
270    L5          CMP #RIGHTSTOP
280                BNE L6
290                LDA VIC+16
300                BNE LOOP2
310 ;
320    L6          INC VIC
330                JMP LOOP2
340    ;********************************
350    MOVEUP      LDA     VIC+1
360                CMP #UPSTOP
370                BEQ LOOP2
380                DEC VIC+1
390                JMP LOOP2
```

```
400     ;***
410     MOVEDOWN    LDA VIC+1
420                 CMP #DOWNSTOP
430                 BEQ LOOP2
440                 INC VIC+1
450                 JMP LOOP2
460     ;*******************************
470     START       LDA#00
480                 STA 53280
490                 LDA #11
500                 STA 53281
510                 LDA #<(TEXT)
520                 LDY #>(TEXT)
530                 JSR STROUT
540 ;
550                 LDA #01
560                 STA VIC+39
570                 LDA #150
580                 STA VIC
590                 STA VIC+1
600                 LDA #13
610                 STA 2040
620                 LDA #01
630                 STA VIC+21
640     ;*******************************
650     LOOP1       STA OLDINPUT
660     LOOP2       LDA $DC00
670                 EOR#$FF
680                 AND #$1F
690                 CMP OLDINPUT
700                 BNE LOOP1
710                 STA OLDINPUT
720 ;
730                 CMP #16
740                 BCC L1
750                 RTS
760 ;***
770     L1          LDX #00
780                 STX H
790                 STX HQ
800                 STX V
810                 STX VQ
820 ;
830                 LSR
840                 ROL V
850                 LSR
860                 ROL H
870                 LSR
880                 ROL VQ
890                 LSR
900                 ROL HQ
910     ;*******************************
920     HORIZON     LDA H
930                 EOR HQ
940                 CMP HMEM
950                 BEQ VERTICAL
960                 STA HMEM
970                 CMP #00
980                 BNE L2
990                 LDA H
1000                STA OLDH
1010                JMP VERTICAL
1020     ;***
1030     L2         LDA OLDH
1040                EOR H
1050                BNE L7
1060                JMP MOVELEFT
1070    L7          JMP MOVERIGHT
1080    ;*******************************
1090    VERTICAL    LDA V
```

```
1100                    EOR VQ
1110                    CMP VMEM
1120                    BEQ LOOP2
1130                    STA VMEM
1140                    CMP #00
1150                    BNE L8
1160                    LDA V
1170                    STA OLDV
1180                    JMP LOOP2
1190    ;***
1200    L8              LDA OLDV
1210                    EOR V
1220                    BNE L9
1230                    JMP MOVEUP
1240    L9              JMP MOVEDOWN
1250    ;********************************
```

Magic Disk 09/91

Listing „AMIGA-MAUS2.SRC“

```
10      .BA $9000
11      .EQ STROUT=$AB1E
12      .EQ OLDH=$02
13      .EQ H=$03
14      .EQ HQ=$04
15      .EQ V=$05
16      .EQ VQ=$06
17      .EQ LEFTSTOP=24
18      .EQ RIGHTSTOP=87
19      .EQ OLDINPUT=$0A
20      .EQ HMEM=$F9
21      .EQ VMEM=$FA
22      .EQ OLDV=$FB
23      .EQ UPSTOP=50
24      .EQ DOWNSTOP=249
30      .EQ VIC=$D000
31      .EQ TEXT=$093F
80      ;********************************
100     MOVELEFT        LDA VIC
110                     BNE L3
120                     STA VIC+16
130                     BEQ L4
140     L3              CMP #LEFTSTOP
150                     BNE L4
160                     LDA VIC+16
170                     BEQ LOOP2
180;
190     L4              DEC VIC
200                     JMP LOOP2
210     ;***
220     MOVERIGHT       LDA VIC
230                     CMP #$FF
240                     BNE L5
250                     INC VIC+16
260                     BNE L6          ;UNBED
270     L5              CMP #RIGHTSTOP
280                     BNE L6
290                     LDA VIC+16
300                     BNE LOOP2
310 ;
320     L6              INC VIC
330                     JMP LOOP2
340     ;********************************
350     MOVEUP          LDA VIC+1
360                     CMP# UPSTOP
370                     BEQ LOOP2
380                     DEC VIC+1
390                     JMP LOOP2
```

```
400     ;***
410     MOVEDOWN    LDA VIC+1
420                 CMP #DOWNSTOP
430                 BEQ LOOP2
440                 INC VIC+1
450                 JMP LOOP2
460     ;********************************
470     START       LDA #00
480                 STA 53280
490                 LDA #11
500                 STA 53281
510                 LDA #<(TEXT)
520                 LDY #>(TEXT)
530                 JSR STROUT
540 ;
550                 LDA #01
560                 STA VIC+39
570                 LDA #150
580                 STA VIC
590                 STA VIC+1
600                 LDA #13
610                 STA 2040
620                 LDA #01
630                 STAVIC+21
640     ;********************************
650     LOOP1       STA OLDINPUT
660     LOOP2       LDA $DD01
670                 EOR #$FF
680                 AND #$1F
690                 CMP OLDINPUT
700                 BNE LOOP1
710                 STA OLDINPUT
720 ;
730                 CMP #16
740                 BCC L1
750                 RTS
760     ;***
770     L1          LDX #00
780                 STX H
790                 STX HQ
800                 STX V
810                 STX VQ
820 ;
830                 LSR
840                 ROL V
850                 LSR
860                 ROL H
870                 LSR
880                 ROL VQ
890                 LSR
900                 ROL HQ
910     ;********************************
920     HORIZON     LDA H
930                 EOR HQ
940                 CMP HMEM
950                 BEQ VERTICAL
960                 STA HMEM
970                 CMP#00
980                 BNE L2
990                 LDA H
1000                STA OLDH
1010                JMP VERTICAL
1020    ;***
1030    L2          LDAOLDH
1040                EOR H
1050                BNE L7
1060                JMP MOVELEFT
1070    L7          JMP MOVERIGHT
1080    ;********************************
1090    VERTICAL    LDA V
```

```
1100                    EOR VQ
1110                    CMP VMEM
1120                    BEQ LOOP2
1130                    STA VMEM
1140                    CMP #00
1150                    BNE L8
1160                    LDA V
1170                    STA OLDV
1180                    JMP LOOP2
1190    ;***
1200    L8              LDA OLDV
1210                    EOR V
1220                    BNE L9
1230                    JMP MOVEUP
1240    L9              JMP MOVEDOWN
1250    ;********************************
```

Listing „AMIGA-MAUS3.SRC“

```
5       .OB"@:MAUS3.OBJ,P,W"
10      .BA $0801
11      .EQ CIA1=$DC00
12      .EQ CIA2=$DD00
13      .EQ LEFTSTOP=24
14      .EQ RIGHTSTOP=87
15      .EQ UPSTOP=50
16      .EQ DOWNSTOP=249
17      .EQ VIC=$D000
18      .EQ STROUT=$AB1E
19      .EQ ENDIRQ=$EA7E
20      .EQ SYSIRQ=$EA31
21      .EQ MEM=$02
80      ;********************************
81      .BY $0B,$08,$0A,$00,$9E
82      .TX "2061"
83      .BY $00,$00,$00
90      ;********************************
100     START           SEI
110                     LDX #<(IRQ)
120                     LDY #>(IRQ)
130                     STX $0314
140                     STY $0315
150 ;
160                     LDX #<(NMI)
170                     LDY #>(NMI)
180                     STX $0318
190                     STY $0319
200 ;
210                     LDX #$83
220                     STX CIA1+13
230                     STX CIA2+13
240 ;
250                     LDA #00
270                     STA CIA1+6
280                     STA CIA1+7
290                     STA CIA2+6
300                     STA CIA2+7
310 ;
320                     LDY #$90
330                     STA CIA2+4
340                     STY CIA2+5
350 ;
360                     LDA #$21
370                     STA CIA1+15
380                     STA CIA2+15
390                     LDA #$81
400                     STA CIA2+14
410 ;
420                     LDA #00
```

```
430                     STA 53280
440                     LDA #11
450                     STA 53281
460                     LDA #<(TEXT)
470                     LDY #>(TEXT)
480                     JSR STROUT
490 ;
500                     LDA #01
510                     STA VIC+39
511                     STA VIC+21
520                     LDA #150
530                     STA VIC
550                     STA VIC+1
560                     LDA #41
570                     STA 2040
600 ;
610                     CLI
620                     RTS
630     ;********************************
640     NMI             PHA
650                     TXA
660                     PHA
670                     TYA
680                     PHA
690                     CLI
700 ;
710                     LDA $DD01
720                     EOR #$FF
730                     STA MEM
740                     LDA CIA2+13
750                     AND #$02
760                     BEQ BUTTONS
770 ;
780                     LDA MEM
790                     AND #$08
800                     BEQ MOVELEFT
810                     BNE MOVERIGHT
820 ;
830     BUTTONS         LDA MEM
840                     AND #$30
850                     BEQ BYE
860 ;
870                     AND #$20
880                     BEQ LEFTONE
890 ;
900                     LDA #42
910                     LDX #01
920     L8              STA 2040
930                     STX CIA1+6
940                     STX CIA2+6
950                     JMP BYE                 ;UNBED
960 ;
970     LEFTONE         LDA #41
980                     LDX #00
990                     JMP L8
1000    ;********************************
1010    IRQ             LDA CIA1+13
1020                    AND #$02
1030                    BNE OK
1040                    JMP SYSIRQ
1050 ;
1060    OK              CLI
1070                    LDA $DD01
1080                    EOR #$FF
1090                    AND #$04
1100                    BEQ MOVEUP
1110                    BNE MOVEDOWN
1120    ;********************************
1130    MOVELEFT        LDA VIC
1140                    BNE L3
```

```
1150                    STA VIC+16
1160                    BEQ L4
1170    L3              CMP #LEFTSTOP
1180                    BNE L4
1190                    LDA VIC+16
1200                    BEQ BYE
1210 ;
1220    L4              DEC VIC
1230    BYE             JMPENDIRQ
1240    ;***
1250    MOVERIGHT       LDA VIC
1260                    CMP #$FF
1270                    BNE L5
1280                    INC VIC+16
1290                    BNE L6          ;UNBED
1300    L5              CMP #RIGHTSTOP
1310                    BNEL 6
1320                    LDA VIC+16
1330                    BNE BYE
1340 ;
1350    L6              INC VIC
1360                    JMP ENDIRQ
1370    ;*********************************
1380    MOVEUP          LDA VIC+1
1390                    CMP #UPSTOP
1400                    BEQ BYE
1410                    DEC VIC+1
1420                    JMP ENDIRQ
1430    ;***
1440    MOVEDOWN        LDA VIC+1
1450                    CMP #DOWNSTOP
1460                    BEQ BYE
1470                    INC VIC+1
1480                    JMP ENDIRQ
1490    ;*********************************
1500    TEXT            .TX"<Shift+CLR/Home><CTRL+N><CTRL+8><CTRL+9>          AMIGA-MAUS3
<CTRL+0>"
1510                    .TX "Dieses Programm demonstriert eine Maus-"
1520                    .TX"abfrage ueber den Userport."
1530                    .BY 13
1540                    .TX"Schliessen Sie bitte eine original AMIGA-"
1550                    .TX"maus ueber den im CIA-Kurs beschriebenen"
1560                    .TX "Adapterstecker2 an."
1570                    .BY 13,13
1580                    .TX"Maustasten : L=schnell R=langsam"
1590                    .BY 13,0
10000                   .EN
```

<Shift+CLR/Home> = Steuercode für Bildschirm löschen
<CTRL+N> = Steuercode für Kleinschreibung
<CTRL+8> = Steuercode für Zeichenfarbe Gelb
<CTRL+9> = Steuercode für Reverse Modus ein
<CTRL+0> = Steuercode für Reverse Modus aus