Assembler ist keine Alchimie — Teil 10

Auf zum Endspurt: Die letzten 4 Assembler-Befehle werden in Angriff genommen (jedoch nicht die letzte Folge des Kurses). Wir steigen ein in ein unerhört interessantes, wenn auch nicht einfaches Thema: Die Interrupt-Technik.

Die Assembler-Befehle haben wir bis auf vier noch offenstehende alle behandelt. Diese vier, die alle mit dem Interrupt-Handling zusammenhängen, sollen diesmal unser Thema sein. Um nach längerer Zeit mal wieder auf den Titel dieser Serie zurückzukommen (nämlich die Alchimie!): Wenn wir diese 4 Befehle beherrschen, haben wir den ersten Schritt zum Meister der Assembler-Alchimie getan. Diese vier kleinen 1-Byte-Befehle öffnen uns eine geheime Pforte zu einem Universum an Programmier-Möglichkeiten, von dem wir bisher kaum zu träumen vermochten. Genug der Schwärmerei, erst kommt noch eine Menge Arbeit, die uns wohl mehrere Folgen dieser Serie in Atem halten wird.

Zuvor noch eine Bemerkung: es gibt kaum ein Thema im Rahmen der Programmierung in Assembler, welches so penetrant häufig Abstürze provoziert, wie das nunmehr angesteuerte! Falls Sie noch keine RESET-Taste an ihrem Computer haben, wird es nun höchste Zeit. Diese nützlichen Dinger werden inzwischen schon so preiswert angeboten (sehen Sie mal in den Kleinanzeigenteil!), daß Sie zur Grundausstattung eines Assembler Alchimisten zählen.

Was sind Interrupts?

Unser Computer ist — solange er eingeschaltet ist — ständig mit irgendwelchen Tätigkeiten beschäftigt. Im Direktmodus hängt er beispielsweise meistens in einer Warteschleife und harrt der Eingaben, im Programm-Modus arbeitet er sich mit Hilfe der Interpreterschleife durch einen Basic-Befehlstext hindurch und so weiter. Nun werden Sie ja sicher schon festgestellt haben, daß er im Direktmodus auch den Cursor blinken läßt, in beiden Modi die TI$-Uhr weiterzählt und weitere Dinge macht, die anscheinend so nebenher passieren. Schon in der ersten Folge dieser Serie aber haben wir einen Unterschied zwischen Mensch und Computer festgehalten: Der Mensch kann mehrere Dinge gleichzeitig tun, der Mikroprozessor ist nur fähig zu einer Arbeit pro Zeiteinheit. Weil aber diese Zeiteinheiten so unfaßbar kurz sind (etwa eine Millionstel Sekunde), haben wir Benutzer den Eindruck der Gleichzeitigkeit.

Wenn dem aber so ist, wie macht es der Computer, daß er beispielsweise ein Programm abarbeitet und trotzdem die TI$-Uhr weiterzählt? Durch Unterbrechungen (interrupt = unterbrechen) der gerade ausgeübten Tätigkeit. Ein Beispiel aus dem täglichen Leben soll uns das illustrieren: Sie lesen gerade diesen Artikel, als das Telefon klingelt und ein Freund von Ihnen wissen möchte, was eigentlich Unterbrechungen sind. Während Sie es ihm erklären, fängt in der Küche der Teekessel schrill zu pfeifen an. Sie sagen Ihrem Freund, er möge sich einen Moment gedulden, gehen in die Küche und nehmen den Kessel vom Feuer. Dann kehren Sie ans Telefon zurück und beenden nach einer Weile das Gespräch. Nach dem Auflegen des Telefonhörers setzen Sie die Lektüre des Artikels fort, fest entschlossen, sich nun nicht mehr unterbrechen zu lassen. Kurze Zeit später klingelt jemand an der Tür. Sie lassen sich dadurch nicht stören.

Dieses Gleichnis gibt ziemlich genau wieder, was sich im Computer — nur bei millionenfacher Geschwindigkeit — bei Unterbrechungen abspielt. In Bild 1 ist das Schema des Ablaufes grafisch dargestellt. In gewisser Weise ähnelt das ganze dem Abarbeiten von Unterprogrammsequenzen. Weshalb programmiert man dann nicht einfach mittels einiger JSR-Aufrufe? Dafür hat L.A.Leventhal einen einleuchtenden Vergleich: »Ein Unterbrechungs-System entspricht etwa einer Telefonklingel. Sie läutet, wenn ein Anruf empfangen wird, so daß man den Hörer nicht laufend abnehmen muß, um festzustellen, ob sich jemand in der Leitung befindet.« (L.A.Leventhal, »6502 Programmieren in Assembler«, München te-wi Verlag, S.121). Unterbrechungen können dann angefordert und abgearbeitet werden, wenn sie nötig sind, im Gegensatz zu Unterprogrammen, die erst dann berücksichtigt werden, wenn der Programmzähler einen JSR-Befehl erfaßt. Um also schnell reagieren zu können, müßte man sehr oft in einem Programm eine Unterroutine anspringen, die auf gewisse Registerinhalte prüft und dann zur Bearbeitung verzweigt oder — bei Nichtvorliegen einer Bedingung — im normalen Programm weiterfährt. Das kostet unnötig Zeit und Speicherraum. Mancher Verkehr des Computers mit Peripherie erfordert so schnelle Reaktionen, daß diese nur geleistet werden können durch Unterbrechen des laufenden Programmes.

Ich denke, daß Sie nun die Notwendigkeit von Unterbrechungen erkennen. Fast jede CPU kennt solche Unterbrechungssysteme. Man kann sie charakterisieren durch die Beantwortung folgender Fragen:

1) Welche Unterbrechungs-Eingänge weist die CPU auf?
2) Wie reagiert die CPU auf eine Unterbrechung?
3) Wie bestimmt die CPU die Unterbrechungsquelle, wenn die Anzahl der Quellen größer ist als die Anzahl der Eingänge?
4) Kann die CPU zwischen wichtigen und weniger wichtigen Unterbrechungen unterscheiden?
5) Wie und wann wird das Unterbrechungssystem freigegeben oder gesperrt?

All diese Fragen werden wir im Laufe dieser Serie für unseren Computer ergründen.

Das Unterbrechungssystem der CPU 6510/6502

Einige dieser Charakteristika sind schnell zu zeigen:

Zu 1: Unsere CPU hat ganau 2 Eingänge für Unterbrechungen (wenn man RESET außer acht läßt, was wir im folgenden meist tun werden).

Zu 3: Natürlich gibt es weitaus mehr denkbare Unterbrechungsquellen als diese 2 Eingänge, weshalb softwaremäßig eine Registerabfrage (das sogenannte Polling) durchgeführt wird, um die Quelle festzustellen.

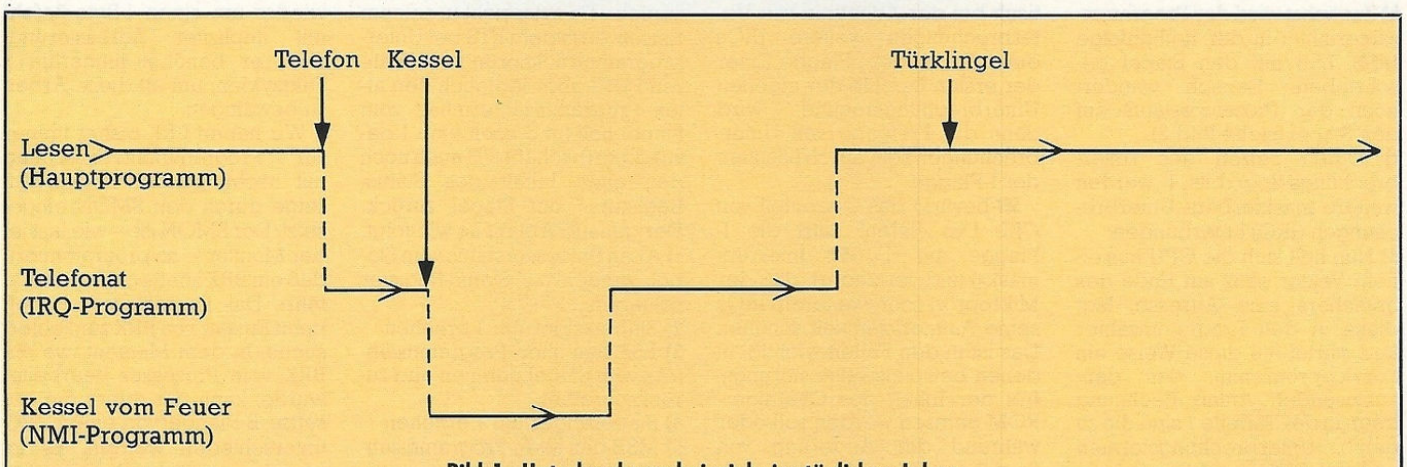

Bild 1. Unterbrechungsbeispiele im täglichen Leben

Zu 4: Zwischen wichtiger und nicht so wichtiger Unterbrechung kann unsere CPU unterscheiden durch die Priorität der beiden Eingänge. Wir haben eine sogenannte maskierbare Unterbrechung, genannt IRQ, welche per Befehl ignoriert (maskiert) werden kann und eine andere, nicht maskierbare, die daher auch NMI (not maskable interrupt = nicht maskierbare Unterbrechung) genannt wird. NMI hat eine höhere Priorität als IRQ und kann deshalb für die wichtigeren Aufgabenstellungen eingesetzt werden.

Zu 5: Freigegeben oder gesperrt werden kann die IRQ-Unterbrechung durch ein Sperrbit (auch Maskenbit genannt), welches sich als Bit 2 im Flaggen-Register des Prozessors befindet. Das ist die I-Flagge. Für den Empfang der NMI-Unterbrechung kann die CPU nicht gesperrt werden.

Um mal die Parallele zu unserem Beispiel zu zeigen: Das Lesen des Artikels ist die gerade stattfindende Tätigkeit des Computers. Die Telefonklingel signalisiert einen IRQ, der im folgenden bearbeitet wird. Das Pfeifen des Teekessels soll einem NMI entsprechen. Wenn dieser dann bearbeitet ist, geht es mit der Abarbeitung des IRQ weiter. Nach Beendigung des Telefonates wird das Unterbrechungs-Sperrbit gesetzt (sie nehmen sich vor, sich nicht mehr stören zu lassen) und mit der normalen Tätigkeit fortgefahren. Weil der nun folgende IRQ damit maskiert ist, wird das Türklingeln ignoriert.

Die **Frage 2**, nämlich wie unsere CPU auf eine Unterbrechung reagiert, blieb noch unbeantwortet. Nun soll sie behandelt werden:

a) Am Ende jedes Befehls überprüft die CPU automatisch den Zustand des Unterbrechungs-Systems. Wenn an einer der beiden Unterbrechungsleitungen eine Anforderung vorliegt und diese auch freigegeben ist, beginnt die Unterbrechung zu wirken.

b) Zunächst wird der Programmzählerinhalt in der Reihenfolge MSB, LSB auf den Stapel geschrieben. Danach wandert noch der Prozessorstatus auf den Stapel (siehe Bild 2).

c) Durch Setzen des Unterbrechungs-Sperrbits I werden weitere maskierbare Unterbrechungen (IRQ) unterbunden.

d) Nun holt sich die CPU aus einem Vektor ganz am Ende des Speichers eine Adresse, lädt diese in den Programmzähler und startet auf diese Weise ein Serviceprogramm, das dem auslösenden Anlaß Rechnung trägt. In der Tabelle 1 sind die zu den Unterbrechungsformen und zum RESET gehörigen Vektoren aufgeführt.

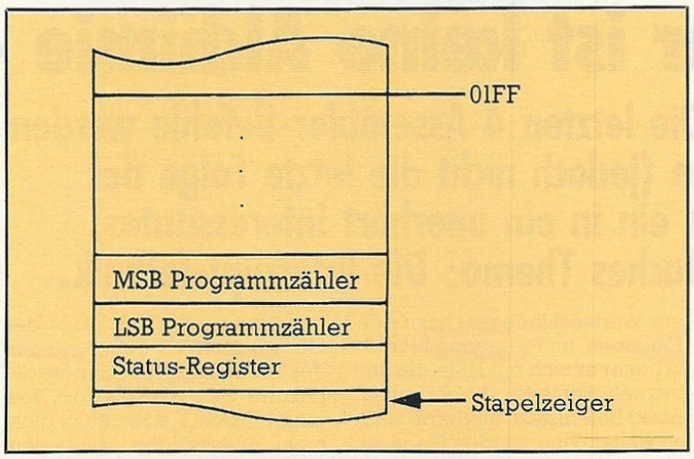

Bild 2. Die CPU rettet den Programmzähler und das Statusregister beim Eintreten einer Unterbrechung auf den Stapel

Bevor wir uns weiter mit den so angesteuerten Routinen befassen, wollen wir die 4 Befehle kennenlernen, die uns noch fehlen.

Schlüssel zur Unterbrechungsprogrammierung: CLI, SEI, RTI, BRK

Das Sperren der maskierbaren Unterbrechung IRQ und das Löschen der Maske erfolgt durch Setzen oder Löschen des Sperrbits im Prozessorstatus-Register. Dieses Bit, die I-Flagge, kann durch den Befehl CLI gelöscht werden. **CLI** kommt von »**CL**ear **I**nterrupt mask«, was bedeutet »lösche die Unterbrechungs-Maske«. Immer dann, wenn IRQs zugelassen sein sollen zur Bearbeitung durch den Mikroprozessor, muß damit die I-Flagge gelöscht werden. Wie Sie sehen, ist CLI ein 1-Byte-Befehl mit impliziter Adressierung. Er braucht genau 2 Taktzyklen zur Erledigung seiner Aufgabe.

Wenn wir später eigene Unterbrechungsroutinen schreiben, stehen wir oft vor der Frage, ob wir innerhalb unseres Unterbrechungsprogramms weitere Unterbrechungen zulassen wollen. Manchmal ist das wichtig, beispielsweise bei der Tastaturabfrage. Wie wir vorhin erwähnt haben, sperrt die CPU automatisch bei der Annahme von Unterbrechungen weitere IRQs durch Setzen der I-Flagge. Einer der ersten Befehle der eigenen Unterbrechungsroutine wird dann die Freigabe von Unterbrechungen sein durch Löschen der I-Flagge.

SEI bewirkt das Gegenteil von CLI. Der Befehl setzt die I-Flagge auf 1 (»**SE**t **I**nterrupt mask«) und verhindert, daß der Mikroprozessor weiteren IRQs seine Aufmerksamkeit schenkt. Das ist in den Fällen wichtig, in denen beispielsweise störungsfrei der Inhalt des Charakter-ROM gelesen werden soll oder während der Änderung von Speicherstellen, die die IRQ-Routine benutzt. Wie wichtig das Sperren von IRQs sein kann, haben Sie eventuell bemerkt, wenn Ihnen das Hilfsbildschirmprogramm aus der 6. Folge mal abgestürzt war. Seit der letzten Folge — wo wir die IRQs gesperrt haben — ist Ihnen das sicherlich nicht mehr passiert. Ebenso wie CLI ist SEI ein 1-Byte-Befehl mit impliziter Adressierung, und auch er braucht 2 Taktzyklen zur Bearbeitung.

Noch eine Bemerkung zum Verhindern der IRQs. Wir werden später sehen, was alles während der 60mal pro Sekunde aufgerufenen Unterbrechung erledigt wird. Jede Routine, die SEI verwendet, verbraucht Rechenzeit. Wenn sie so lange dauert, daß eine oder mehrere dieser regelmäßigen IRQs unterbunden werden, kann das unter Umständen zu Störungen von Programmabläufen führen. In solchen Fällen ist es sinnvoll, in die eigene Routine den Teil der regulären IRQ-Routine einzubauen, der im Programmablauf durch sein Fehlen Störungen verursacht. Meistens kann man aber durch gute Planung eines Programmes dieses Problem umgehen.

RTI heißt »**R**e**T**urn from **I**nterrupt«, zu deutsch also: »kehre aus dem Unterbrechungsprogramm zurück.« Es entspricht in seinem Einsatz etwa dem RTS bei Unterprogrammrücksprüngen. Während RTS aber lediglich den alten Programmzählerinhalt vom Stapel holt (und noch eine 1 dazuaddiert), schafft RTI auch noch den alten Inhalt des Status-Registers vom Stapel zurück. Der genaue Ablauf ist wie folgt:

1) Alten Prozessorstatus vom Stapel wieder ins Status-Register schieben.
2) Stapelzeiger um 1 erhöhen
3) LSB des alten Programmzählers vom Stapel nehmen und zurückschreiben.
4) Stapelzeiger um 1 erhöhen
5) MSB des alten Programmzählers vom Stapel nehmen und zurückschreiben.
6) Stapelzeiger um 1 erhöhen.

Damit ist der Zustand vor der Unterbrechung wieder hergestellt. Auch die I-Flagge ist so automatisch wieder gelöscht, denn vor der Unterbrechung war sie sicher nicht gesetzt gewesen und der alte Status-Zustand ist ja jetzt wieder vorhanden.

RTI ist ebenfalls ein 1-Byte-Befehl mit impliziter Adressierung. Seine vollständige Bearbeitung dauert 6 Taktzyklen.

Bei eigenen Unterbrechungs-Routinen verwendet man häufig nicht RTI, sondern springt durch JMP an eine sinnvolle Stelle des normalen Unterbrechungsprogrammes. Auf diese Weise kann man dann die normalen Arbeitsgänge der vorprogrammierten Unterbrechung oder Teile davon noch ausführen lassen.

Den Befehl **BRK** (break = Software-Unterbrechung) haben wir schon verwendet. Er entspricht in seinem Einsatz etwa dem STOP-Befehl in Basic und dient wie jeder Befehl dort hauptsächlich dem Testen von Programmen. Tatsächlich unterscheidet sich die Reaktion unserer CPU bei Auftreten eines BRK kaum von der bei einem IRQ. Folgendes passiert:

a) Der Programmzähler wird um 2 erhöht.
b) Bit 4 des Prozessorstatusregisters, die Break-Flagge B, wird auf 1 gesetzt.
c) Das MSB des Programmzählers wird auf den Stapel gebracht und der Stapelzähler um 1 heruntergezählt.
d) Dasselbe geschieht nun mit dem LSB des Programmzählers
e) und mit dem Statusregister.
f) Das Unterbrechungsmaskenbit, die I-Flagge, wird auf 1 gesetzt um IRQs zu sperren.
g) In den Programmzähler wird nun aus dem Vektor FFFE/FFFF dieselbe Adresse geladen, die auch bei IRQs benutzt wird. Damit startet nun das Programm, das diese Unterbrechung bearbeitet.

Sie sehen, daß der BRK-Befehl ein ziemlich komplizierter Geselle ist. Zwar handelt es sich wieder um einen 1-Byte-Befehl mit impliziter Adressierung, aber er benötigt immerhin 7 Taktzyklen, um all diese Arbeit zu bewältigen.

Wir haben BRK bisher immer zur Programmunterbrechung mit nachfolgender Registeranzeige durch den SMON eingesetzt. Der SMON ist — wie fast jeder Monitor — so programmiert, daß ein BRK zur Registeranzeige führt. Das ist natürlich sinnvoll beim Einsatz von BRK zur Fehlersuche. In dem Moment, wo ein BRK vom Prozessor bearbeitet wurde, kann nur durch die gesetzte B-Flagge von einem IRQ unterschieden werden. Es ist manchmal nötig, schon zu diesem Zeitpunkt diesen Unter-

schied festzustellen. Deshalb verwendet man den nachfolgend beschriebenen Test zu diesem Zweck:

PLA
in den Akku wird das zuletzt auf den Stapel geschobene Prozessorstatus-Register geholt.

PHA
und sogleich wieder zurückgeschoben

AND #$10
durch die AND-Verknüpfung mit der Binärzahl 0001 0000 kann eine eventuell vorhandene B-Flagge isoliert werden.

BNE BREAK
Falls eine B-Flagge gesetzt war, ist der Akku ungleich 0 und die Bearbeitung verzweigt zum von uns konstruierten BREAK-Programm. War der Akku nach dieser AND-Verknüpfung gleich 0, dann erfolgt keine Verzweigung und es handelt sich um einen IRQ, zu dessen Bearbeitung nun zu springen ist.

Es gibt noch eine andere — gebräuchlichere — Möglichkeit, zwischen einem BRK und einem IRQ zu unterscheiden, die allerdings erst zu einem späteren Zeitpunkt des computerinternen Unterprogrammes erfolgt. Von dieser zweiten Möglichkeit wird im SMON Gebrauch gemacht und wir werden sie nachher auch kennenlernen.

Natürlich kann der BRK-Befehl auch zu anderen Zwecken als zur Registeranzeige durch einen Monitor verwendet werden. Es kommt immer darauf an, welches Service-Programm wir dem Computer anbieten. Springt man aus so einem Service-Programm mittels RTI zurück ins Hauptprogramm, dann muß man berücksichtigen, daß der Programmzähler vor der Sicherung auf dem Stapel um 2 erhöht worden ist. Manchmal sind deshalb noch Korrekturen des Programms nötig.

Ich hoffe, daß Sie bisher diesen Artikel nicht zu frustrierend fanden, denn ständig ist die Rede vom eigenen Unterbrechungs-Programm und dabei wissen Sie — außer durch BRK — noch gar keine Möglichkeit, einen IRQ oder NMI auszulösen, und Sie sind sicher noch sehr vorsichtig mit dem Gedanken an eigene Unterbrechungs-Routinen, weil Ihnen ja noch unbekannt ist, wie die normale Firmware Unterbrechungen behandelt. Keine Angst: All das werden wir noch klären. Betrachten Sie diese Folge zum Thema Unterbrechungen vielleicht mehr wie ein Handbuch, in dem Sie dann, wenn Ihr Verständnis gestiegen ist, nochmal zurückblättern können:

Wir haben bisher nur betrachtet, wie unsere CPU reagiert, wenn an einem der beiden Unterbrechungs-Eingänge (IRQ und NMI) eine Unterbrechungs-Anforderung vorliegt. Um nun aber selbst ins Geschehen eingreifen zu können, ist es nötig zu wissen, wie diese Anforderung dorthin gelangt. Das erfordert von uns die Beschäftigung mit anderen Computerbausteinen als der CPU, die bisher im Mittelpunkt unseres Interesses stand.

Woher kommen die Unterbrechungs-Anforderungen?

Quellen für Unterbrechungen können viele genannt werden: Diskettenstation, Datasette, Drucker, Modem, Schaltelemente und so weiter. Um aber eine gewisse Übersicht zu bekommen, sollte man unterscheiden zwischen primären und sekundären Unterbrechungsquellen. Das soll kurz erläutert werden: Die Diskettenstation beispielsweise ist über den seriellen Port mit dem Computer verbunden. Dieser wiederum steht in direktem Kontakt zu 2 Bausteinen, den CIAs. Erst diese CIAs stehen in direktem Kontakt zur CPU. Alle Unterbrechungs-Quellen, die direkt Signale an die beiden Unterbrechungseingänge unserer CPU senden, sollen künftig »primäre« Quellen genannt werden, die anderen, die nur über solch eine primäre Quelle Unterbrechungs-Anforderungen stellen, werden von uns als »sekundäre« Quellen bezeichnet. Weil wir irgendwo einen Schnitt machen müssen — einmal, um nicht völlig auszuufern in der Erklärung von peripheren Geräten (das soll anderen, kompetenteren überlassen bleiben) und zum anderen, weil ich mich da auch nicht so gut auskenne — werden wir uns im folgenden auf die primären Unterbrechungsquellen beschränken. Da bleibt aber noch mehr als genug zu tun übrig und deshalb soll auch nur eine Auswahl dieser Primärquellen detailliert behandelt werden.

Welches sind nun die primären Unterbrechungsquellen? Hier sind sie aufgeführt:

1) Der VIC-II-Chip (MOS 6566/6567 Video Interface Controller)
2) Die beiden CIAs (MOS 6526 Complex Interface Adapter)
3) Die RESTORE-Taste
4) Der Expansion-Port
5) RESET (paßt hier nicht ganz her, woanders aber auch nicht besser)

Den Expansion-Port werden wir nicht behandeln und einen RESET nur ziemlich kurz betrachten, weil es sich dabei eigentlich nicht um eine Unterbrechung im bisher definierten Sinn handelt.

Der VIC-II-Chip als Unterbrechungsquelle

Soweit ich feststellen konnte, kommt der VIC-II-Chip in Bezug auf unsere CPU nur als Anforderer von maskierten Unterbrechungen (IRQ) in Frage. Die Handhabung seiner Unterbrechungs-Anforderungen geschieht im VIC-II-Chip durch zwei Register. Vier Ereignisse sind eingeplant, deren Eintreten zur Unterbrechung führen kann:

1) Rasterzeilen-Unterbrechung
2) Kollision eines Sprite mit Hintergrund
3) Kollision von Sprites untereinander
4) Lichtgriffel-Unterbrechung.

Die ersten 3 Auslöser werden wir uns in kommenden Folgen genau ansehen und dabei vielerlei interessante Möglichkei-

7	6	5	4	3	2	1	0
unbenutzt				Lichtgriffel-Unterbrechung	Kollision Sprite/Sprite	Kollision Sprite/Hintergrund	Raster-Unterbrechung

Bild 3. Das Interrupt-Enable-Register (53274 = $d01a) des VIC-II-Chip

7	6	5	4	3	2	1	0
Markierung, ob zugelassene Unterbrechung aufgetreten ist	unbenutzt			Lichtgriffel-Unterbrechung	Kollision Sprite/Sprite	Kollision Sprite/Hintergrund	Raster-Unterbrechung

Bild 4. Das Interrupt-Latch-Register (53273 = $d019) des VIC-II-Chip

ten feststellen. Die Option, die der Lichtgriffel bietet, wird nicht behandelt werden: Meine Kenntnisse auf diesem Sektor sind nur gering (nobody is perfect).

Das sogenannte Interrupt Enable Register (Unterbrechungs-Zulassungs-Register) des VIC-II-Chips ist Register 26. Es befindet sich in der Speicherstelle 53274 ($D01A) (siehe Bild 3).

In diesem Register wird festgelegt, ob eines — oder mehrere — der 4 möglichen auslösenden Ereignisse eine Unterbrechungsanforderung an den Mi-

Bild 5. Genereller Aufbau der Unterbrechungs-Kontroll-Register (13) der beiden CIA-Bausteine

kroprozessor senden soll. Jedem Ereignis ist ein Bit zugeordnet. Ist dieses Bit gleich 1, dann ist die Unterbrechung freigegeben, ist es gleich 0, dann liegt eine Sperrung vor. Die Zuordnung der Bits ist wie folgt:
Bit 0 Rasterzeilen-IRQ
Bit 1 Sprite/Hintergrund-Kollision
Bit 2 Sprite/Sprite-Kollision
Bit 3 Lichtgriffel-IRQ
Bits 4 bis 7 sind ungenutzt und haben immer den Wert 1.

Das Register 25 wird Interrupt Latch Register genannt, was etwa zu übersetzen wäre mit »Unterbrechungs-Einrast-Register« (siehe Bild 4). Der englische Ausdruck »latch«, der nur umschreibend oder sehr technisch übersetzt werden kann, beschreibt eigentlich recht genau, was in diesem Register geschieht. Ein »latch« ist nämlich so etwas wie ein Schnappriegel, also ein Riegel, der bei der Betätigung einrastet. Wenn eines der 4 möglichen Ereignisse eintritt, schnappt im dazugehörigen Bit dieses Registers der Inhalt auf 1. Die Bit-Zuordnung ist die gleiche wie in Register 26. Aber das Bit 7 hat hier noch eine Bedeutung: Ist eines der Bits 0 bis 3 auf 1 gesetzt und das dazugehörige Ereignis in Register 26 auch zur Unterbrechung zugelassen (also auch dort gleich 1), dann taucht in Register 25, Bit 7 eine 1 auf. So kann durch einfaches Lesen dieses Bits festgestellt werden, ob ein IRQ durch den VIC-II-Chip ausgelöst wurde.

Will man in diesem Register ein gesetztes Bit löschen, muß man — außergewöhnlich! — eine 1 in die Bitposition schreiben.

Mit Recht erwarten Sie nun eigentlich eine Anwendung des bisher gelernten. Bei Unterbrechungsprogrammen ist es aber dringend nötig, immer den gesamten Komplex im Auge zu haben. Ich habe mich daher entschlossen, zuerst alles zu erklären und dann Anwendungsmöglichkeiten vorzustellen. Ihre Geduld wird auf eine harte Probe gestellt, aber ich hoffe, daß Sie ab der nächsten Folge feststellen, daß es sich gelohnt hat, etwas zu warten.

An sich sind die beiden CIAs in unserem Computer völlig identisch. Sie werden aber unterschiedlich eingesetzt. Sehen wir uns zunächst einmal an, was beiden in Bezug auf Unterbrechungen gemeinsam ist, um danach die Unterschiede festzuhalten. Die Unterbrechungs-Steuerung geschieht in Register 13 dieser Bausteine. Dieses Register hat 2 Funktionen: Es bestimmt, ob eine Unterbrechungsanforderung an die CPU gesandt werden soll, und es stellt fest, ob ein Ereignis stattgefunden hat, das zur Unterbrechung führen kann. Die Bedienung dieses Registers ist demzufolge auch etwas unübersichtlich, aber wir haben schon ganz andere Probleme gemeistert.

Die beiden CIA-Bausteine als Unterbrechungsquellen

Sehen wir uns aber zuerst einmal an, welche Ereignisse vom Standpunkt eines CIA-Bausteines als Unterbrechungskriterium dienen können:
1) Unterlauf der Uhr A
2) Unterlauf der Uhr B
3) Die interne Uhr hat eine Alarmzeit erreicht
4) Am SP-Eingang (hängt mit dem seriellen Port zusammen) ist ein bestimmter Zustand erreicht
5) An einem Eingang namens FLAG ist ein bestimmter Zustand erreicht.
Die Ereignisse 4 und 5 werden wir ebenfalls im weiteren weitgehend ausklammern.

Nun zum Register 13, dem Unterbrechungs-Kontroll-Register (siehe Bild 5).

Auch hier gehört zu jedem Ereignis ein Bit. Dabei — um Wiederholungen zu vermeiden — ist die Zuordnung schon durch die eben angegebene Ereignisaufzählung gegeben. Ziehen Sie von der vorangestellten Nummer immer eine 1 ab und Sie haben die Bitnummer. Die Bits 5 und 6 sind unbenutzt. Bit 7 hat eine dreifache Funktion, die eng mit den anderen Bitinhalten verknüpft ist. Sehen wir uns das mal der Reihe nach an:

Lesen des Registers

Sind Unterbrechungsereignisse aufgetreten, dann sind die dazugehörigen Bits auf 1 gesetzt. Bit 7 ist gleich 1, wenn mindestens ein solches Ereignis stattgefunden hat und außerdem dieses Ereignis als Unterbrechungsauslöser freigegeben ist. Auf diese Weise kann — ähnlich wie beim VIC-II-Chip-Register 25 — festgestellt werden, ob die Unterbrechung durch einen der beiden CIAs angefordert wurde. Im Unterschied aber zum VIC-II-Register wird Register 13 durch das Lesen **gelöscht**. Braucht man den Inhalt also noch, sollte man ihn irgendwo zwischenspeichern.

Schreiben in das Register

Bit 7 = 0 erzeugt Sperren.
Das erkennt man am besten an einem Beispiel. Nehmen wir an, wir möchten die Unterbrechung sperren, die durch einen Unterlauf von Uhr A erzeugt werden kann. Das betrifft das Bit 0. Wir schreiben in das Register 13 folgende Zahl: 0000 0001

Wie Sie sehen, ist das Bit 7 gleich 0. Die 1 in Bit 1 bewirkt die Sperrung. Durch die Nullen in den anderen Bits wird bewirkt, daß die anderen Unterbrechungs-Ereignisse nicht beeinflußt werden. Wollten wir alle sperren, dann müßten wir einschreiben: 0001 1111

Auf diese Weise können selektiv einzelne Unterbrechungen durch Einschreiben der 1 bei gelöschtem Bit 7 gesperrt werden.

Bit 7 = 1 erzeugt Freigabe.

Auch hier wieder ein Beispiel. Wenn wir ganz gezielt Unterbrechungen durch Unterlauf der Uhr A freigeben wollen, müssen wir die folgende Zahl in Register 13 schreiben: 1000 0001
Bit 7 (gleich 1) zeigt an, daß diejenigen Unterbrechungen freizugeben sind, deren Bits auf 1 gesetzt sind. Alle anderen Unterbrechungen, wo also in der dazugehörigen Bitposition der einzuschreibenden Zahl eine 0 steht, bleiben unverändert.

Ein wichtiger Unterschied zwischen den beiden CIAs ist der, daß der Unterbrechungsausgang von CIA 1 mit dem IRQ-Eingang der CPU verbunden ist, wohingegen der entsprechende Ausgang von CIA2 an den NMI-Eingang unseres Mikroprozessors führt. Daher löst der CIA 1 nur IRQs aus, er wird manchmal deshalb auch IRQ-CIA genannt. Der andere ist dann der NMI-CIA, weil er nur NMIs anfordern kann.

Der IRQ-CIA

Das Register 13 des IRQ-CIA (der die Speicherstellen 56320 bis 56335 belegt), liegt in Zelle 56333 ($DC0D). Die einzelnen Bits sind wie folgt zugeordnet:
Bit 0 Unterlauf Uhr A
Von hier kommt der IRQ, der 60mal pro Sekunde stattfindet zur Tastaturabfrage, zum Weiterstellen der TI$-Uhr etc.
Bit 1 Unterlauf Uhr B
Spielt bei Kassettenoperationen und dem seriellen Port eine Rolle.
Bit 2 ALARM bei interner Uhr.
Spielt beim Zufallszahlengenerator (RND(0)) eine Rolle.
Bit 3 Hier kommen durch den User-Port Unterbrechungs-Anforderungen.
Bit 4 ist verbunden mit dem seriellen Port und der Kassetten-Lese-Leitung.

Der NMI-CIA

Ebenso kurz und schmerzlos wie beim CIA 1 soll auch das besondere am CIA 2, dem NMI-CIA (er belegt den Speicher von 56576 bis 56831) vorgestellt werden. Sein Register 13 findet sich in Speicherstelle 56589 ($DD0D). Die Bits 0 und 1 (Unterläufe der beiden Uhren) spielen beim Senden beziehungsweise Empfangen von Daten über die RS232C-Schnittstelle eine Rolle, Bit 2 (ALARM) wird nicht verwendet, Bit 3 ist direkt mit dem User-Port verbunden ebenso wie Bit 4. Der NMI-CIA wird uns in seiner normalen Funktion nicht mehr beschäftigen.

Die RESTORE-Taste und ein kleines Testprogramm

Die RESTORE-Taste ist direkt mit dem NMI-Eingang unseres Mikroprozessors verbunden. Das ermöglicht es uns, durch einfaches Drücken dieser Taste jederzeit ins Geschehen einzugreifen, ohne uns um Details kümmern zu müssen, ob sich der Computer gerade im Direkt- oder im Programm-Modus befindet und so weiter. Denn NMI hat die höchste Priorität der Unterbrechungen.

Ein kleines Testprogramm soll Ihnen hier noch vorgestellt wer-

den, das Sie vielleicht aber noch nicht ganz verstehen werden, weil wir erst in der nächsten Folge die eingebauten Serviceprogramme kennenlernen werden. Schalten Sie also den SMON ein und geben Sie das Programm 1 ein (ab $6000):

Am besten speichern Sie nun das Programm ab und schalten dann mittels dem SMON-Kommando M 0318 die Anzeige der Bytes ab $0318 ein. Dort steht in den beiden ersten Speicherzellen 47 und FE. Mit dem Cursor fahren Sie in diese Zeile und ändern den Inhalt in 00 und 60, also unsere Programmstartadresse in der LSB/MSB-Form. Nach einem RETURN läuft nun jede NMI-Anforderung über unser Programm. Nun können Sie es ausprobieren, indem Sie mal die RESTORE-Taste drücken. Es genügt völlig, alleine diese Taste zu betätigen. Das wirkt — sichtbar durch die Änderung der Rahmenfarbe — in jedem Modus und jederzeit. Eine kleine Merkwürdigkeit ist, daß man manchmal etwas Geduld aufbringen muß, bis man die Wirkung sieht. Ich vermute, daß der NMI so schnell erledigt wird, daß sich mehrer NMIs pro Tastendruck ereignen. Man müßte sich noch eine kleine Routine überlegen, die die Wirkung etwas verzögert, denn 2 solche EOR-Kommandos nacheinander heben sich gegenseitig auf.

Damit sei's für diesmal genug. In der nächsten Ausgabe werden wir uns die Unterbrechungs-Firmware ansehen und einige Programmbeispiele vorstellen. Zum Schluß noch wie üblich eine Aufstellung (Tabelle 2) mit den besprochenen Befehlen.

(Heimo Ponnath/gk)

```
6000  PHA      mit diesen Befehlen retten wir Akku und Register
6001  TXA      auf den Stapel.
6002  PHA
6003  TYA
6004  PHA

6005  LDA #$7F 0111 1111 ist das in binär.
6007  STA $DD0D Dadurch werden alle NMIs, die vom CIA 2 kommen könnten, gesperrt. Erinnern Sie sich: Bit 7 ist Null beim Schreiben, also Sperrfunktion.
600A  LDY $DD0D Lesen des Registers 13 löscht dieses und zeigt uns, ob die NMI-Anforderung von dort kam.
600D  BMI $601A falls NMI-Anforderung vom CIA 2 kam, wird verzweigt
600F  LDA $D020 ansonsten kommt der NMI von der RESTORE-Taste, und in den Akku wird die Rahmenfarbe eingeladen
6012  EOR #$0E Ausgehend davon, daß als Rahmenfarbe 14 vorliegt, wird diese exklusiv oder verknüpft zu Null. Ist die Rahmenfarbe 0, dann wird sie wieder 14.
6014  STA $D020 Einschreiben des neuen Farbwertes
6017  JMP $FEBC Sprung in den Rest der normalen NMI-Routine
601A  JMP $FE72 Sprung in die normale NMI-Routine im Fall, daß die Anforderung durch den NMI-CIA kam.
```

Programm 1. Ein kleines Testprogramm demonstriert die Wirkung einer Unterbrechung: Durch Drücken der RESTORE-Taste wird die Rahmenfarbe geändert.

Unterbrechungsart	Vektor	Zieladresse
Maskierbare Unterbrechung (IRQ, BRK)	$FFFE/FFFF	65352 $FF48
Reset	$FFFC/FFFD	64738 $FCE2
Nichtmaskierbare Unterbrechung (NMI)	$FFFA/FFFB	65091 $FE43

Tabelle 1. Unterbrechungsvektoren und ihre Inhalte

Befehlswort	Adressierung	Bytezahl	Code Hex	Code Dez	Taktcyclen	Beeinflussung von Flaggen
CLI	implizit	1	58	88	2	I-Flagge
SEI	implizit	1	78	120	2	I-Flagge
RTI	implizit	1	40	64	6	alle Flaggen
BRK	implizit	1	00	0	7	B-Flagge vor dem Schieben auf den Stapel, I-Flagge danach

Tabelle 2. Die Daten zu den letzten Assembler-Befehlen