Werner Stehling

Interrupts bei Prozessoren 680XX

Teil 1: Die Regeln von der Ausnahme

In der guten alten 8-Bit-Welt kam ein Prozessor bequem mit drei verschiedenen Ausnahmebedingungen über die Runden – und schon damit waren die Programme nicht immer leicht zu verstehen. Mit der neuen 16-Bit-Generation ist es wahrhaftig nicht einfacher geworden. Darum zeigt mc, wie die Ausnahmeverarbeitung bei Prozessoren der Serie 68000 gehandhabt wird.

Die erste und wichtigste der Ausnahme-Bedingungen bei 8-Bit-Prozessoren war natürlich der Reset: Hier durfte sich das System mehr oder weniger neu initialisieren. Ein Interrupt, die zweite Ausnahme, sorgte dafür, daß der Prozessor während der Programm-Abarbeitung unterbrochen wurde, um schnell eine andere Tätigkeit auszuführen, und zum dritten konnte so eine Unterbrechung auch per Befehl herbeigeführt werden, ein sogenannter Break.

Die 16-Bit-Welt

Immer getreu der Devise „nicht kleckern sondern klotzen" wird der geneigte Benutzer beim 68000 gebeten, nicht nur drei, sondern 230 verschiedene Ausnahmen zu verwalten. Zum Reset kam der Busfehler dazu, aus einem Interrupt wurden sieben, die man aber auf genau 201 Arten behandeln kann, der Break mauserte sich zu 16 verschiedenen Trap-Befehlen und zusätzlich wurde eine Handvoll Ausnahmen erfunden, auf die man früher bequem verzichten konnte. Bei der ganzen Sache gibt es nur einen Trost: alle Ausnahmen werden im Prinzip auf die gleiche Art behandelt, hat man eine verstanden, dann hat man alle verstanden.
Die Mikroprozessor-Familie der Serie 68000 besteht zur Zeit aus den fünf Mitgliedern 68008, 68000, 68010, 68012 und 68020, und alle Welt behauptet, die Prozessoren seien untereinander aufwärtskompatibel. Nun ist erstens „Kompatibilität" ein sehr dehnbarer (und leider auch arg strapazierter) Begriff und zweitens hat es keinen großen Sinn, die CPU isoliert zu betrachten. Die beiden interessanten Fragen sind deshalb, wie stark sich die Prozessoren in ihrem äußeren Auftreten, also im Zusammenspiel mit der übrigen Hardware, unterscheiden und auf welche Fallstricke der Programmierer gefaßt sein muß, wenn er ein anderes Familienmitglied einsetzen will.
Im Sinne dieser Untersuchung kann man sich auf die Typen 68000, 68010 und 68020 beschränken. Als kleinere Ausgabe des 68000 besitzt der 68008 einen nur acht Bit breiten Datenbus, doch beide verhalten sich bei der Ausnahmeverarbeitung völlig gleich. Die CPU 68012 unterscheidet sich von der 68010 ausschließlich durch einen breiteren Adreßbus und muß daher auch nicht separat betrachtet werden. Wie es sich für eine anständige Familie gehört, sind die Breiten von Adreß- und Datenbus für das Verhalten bei Ausnahmen bedeutungslos, schließlich ist es niemandem verwehrt, den 68020 als 8-Bit-Prozessor zu verwenden.
Im Laufe dieses Beitrages wird, ohne die allgemeine Betrachtung der Ausnahmeverarbeitung zu vernachlässigen, etwas detaillierter auf die Verhältnisse bei Interrupts eingegangen werden. Man kann die Philosophie vertreten, daß alle anderen Ausnahmen nur abgemagerte Interrupts sind, deren Quelle sich meist innerhalb der CPU befindet, eben deshalb ist bei Interrupts nur der Aufwand an Hardware und Verwaltungsarbeit ein bißchen größer. In einem hardwarenahen Teil wird dann auf das Bus-Protokoll eingegangen: wie sieht die Signalfolge auf Adreß-, Daten und Kontrollbus aus, welche CPU-Signale sind überhaupt beteiligt, welche Eigenschaften muß die Peripherie aufweisen. Unter programmtechnischen Aspekten schließlich werden die verschiedenen Stackformate beschrieben, was mit den CPU-Registern geschieht und was man als Programmierer möglichst tun oder lassen sollte. Besonders spannend sind dann die Überlegungen zu Mehrfach-Interrupts mit verschiedenen oder gleichen Prioritäten.

Was ist Ausnahmeverarbeitung

Wie sieht das Leben eines geplagten Mikroprozessors in einem Rechner aus? Er mühlt vor sich hin, holt Befehle und Daten aus dem Speicher, verwurschtelt sie auf irgendeine Weise (oder auch nicht) und speichert das Resultat seiner Bemühungen wieder irgendwo ab. Doch plötzlich passiert etwas Besonderes: Ein Adreß-Strobe tappt ins Leere, unter der gesuchten Adresse meldet sich niemand – wo ist denn nur der Speicher geblieben? Jedenfalls ein völlig neuer Aspekt, wie soll man denn da mit dem normalen Programm fortfahren? Jeder Programmierer hat wohl schon diese trübe Erfahrung gemacht: Die meisten Rechner melden sich dann mit einem häßlichen Piepser und einer Meldung wie: „Ähm – Bus-Fehler – könnten Sie vielleicht mal Reset drücken?"
Wo hier die Ausnahmeverarbeitung steckt? Nun, der Rechner muß irgendwie merken, daß er versucht hat, auf eine nicht existierende Adresse zuzugreifen (Busfehler). Sowie er das festgestellt hat, kann er natürlich nicht im Programm fortfahren, als ob nichts geschehen wäre. Der Vater (oder die Mutter) des Betriebssystems hat aber für diesen Fall vorgesorgt: Das laufende Programm wird abgebrochen, die besagte informative Nachricht erscheint auf dem Bildschirm und die ganze bisherige Rechenzeit ist für die Katz. Das ist Ausnahmeverarbeitung.

Unterschiedliche Arten

Wie man sich denken kann, ist ein Busfehler nur ein möglicher Anlaß, ein Programm nicht normal weiterzuführen. Es können noch andere Fehlerbedingungen auftreten, die vom Programmierer ursprünglich nicht beabsichtigt waren und gesondert behandelt werden müssen. In-

terne Fehler sind aber nicht die einzigen Ursachen für eine Ausnahme. Es gibt durchaus gewollte Unterbrechungen des Programmablaufs, dem Normalverbraucher weithin als Interrupts bekannt. Dabei wird der Prozessor kurzfristig mit einer anderen Aufgabe betraut und führt erst danach das unterbrochene Programm weiter aus. Beispiele sind die Reaktion auf eine kleine Anfrage von einem Peripheriegerät oder -baustein (eine Taste wurde gedrückt, der Drucker kann ein neues Zeichen annehmen) oder ein Timer unterbricht nach einer gewissen Zeit, damit ein anderes Programm auch mal darf (Multitasking).
Die Einteilung in gewollte und nichtgewollte Unterbrechungen ist natürlich völlig willkürlich. Ebenso kann man z. B. zwischen periodischen (Timer, Einzelschritt) und eher zufälligen Ereignissen (Tastatur, Adreßfehler) unterscheiden. Oder wie wäre es mit einer Klassifizierung nach software-generierten (Division durch Null, Einzelschritt) und hardware-korrelierten Unterbrechungen (Busfehler, Timer)? Wie man sieht, sind die Übergänge fließend. Schließen wir uns also lieber Motorolas Standpunkt an und ordnen die verschiedenen Ausnahmebedingungen nach ihrer Wichtigkeit. Aufgrund der Reaktion der CPU auf eine Ausnahmebedingung werden bei 68000 und 68010 drei Gruppen unterschieden:

(0) Die Ausführung des gerade laufenden Befehls ist nicht möglich oder sinnlos (wie etwa beim Busfehler),

(1) der aktuelle Befehl wird fertig abgearbeitet, dann erst beginnt die Ausnahmebehandlung oder

(2) der laufende Befehl selbst führt die Ausnahmebedingung herbei.

Ausnahmen bei 68000 und 68010

Die Ausnahmefälle, die die Prozessoren dabei im einzelnen bearbeiten können, werden im Folgenden nach diesen Gruppen geordnet vorgestellt.

Gruppe 0

0.1 Reset
Ein externes Reset-Signal unterbricht alle laufenden Aktivitäten. Die CPU initialisiert sich neu, indem sie von Adresse $0 des Supervisor-Programm-Bereiches den Startwert des Supervisor-Stackpointers (SSP) und von Adresse $4 den Anfang der Initialisierungs-Routine in den Programmzähler (PC) lädt (alle unverständlichen Ausdrücke werden weiter unten erklärt).

0.2 Address error
Dieser Fehler tritt auf, wenn der Prozessor versucht, einen Befehl oder einen Operanden mit Wort- oder Langwortadressierung auf einer ungeraden Adresse zu erreichen. Die Familien-Philosophie verlangt, daß Worte oder Langworte immer auf geraden Adressen beginnen. In der Ausnahmeverarbeitung kann dann natürlich das Laden von ungeraden Adressen emuliert werden. Der 68020 beschwert sich nur, wenn er einen Befehl auf einer ungeraden Adresse lesen soll, für das Laden von Daten ist er schlau genug.

0.3 Bus error
Ein Busfehler wird außerhalb der CPU durch externe Hardware erkannt. Der Prozessor arbeitet mit allen weiteren Bausteinen des Systems asynchron zusammen, d. h. er legt eine Adresse auf den Adreßbus und erwartet, daß sich innerhalb kürzester Frist irgend jemand angesprochen fühlt. Wenn nicht, erwartet er wenigstens das Bus-Error-Signal.

Gruppe 1

1.1 Trace
Es besteht die Möglichkeit, ein Programm Schritt für Schritt ablaufen zu lassen. Nach jedem Befehl wird zu einer Ausnahmeverarbeitung verzweigt, in der zum Beispiel die Inhalte der Register angezeigt werden können.

1.2...1.8 Interrupt Level 1...7
Externe Geräte oder Bausteine können die Aufmerksamkeit der CPU erheischen und verlangen, daß ihren Bedürfnissen Rechnung getragen wird. So kann etwa ein neuer Meßwert abgeholt oder die Uhrzeit auf dem Bildschirm eingeblendet werden.

1.9 Illegal/unimplemented instruction
Die CPU sieht sich genötigt, einen Befehl ausführen zu sollen, mit dem sie einfach nichts anfangen kann. Von den 65536 möglichen Bitkombinationen gibt es nämlich wahrhaftig noch ein paar, die der Prozessor nicht versteht. Innerhalb der Familie ist die Intelligenz dabei zunehmend: Der 68010 versteht ein bißchen mehr als der 68000, der 68020 noch mehr und was dann noch übrig bleibt, könnte immerhin noch einen Gleitkomma-Coprozessor (Floating Point Unit = FPU) oder eine Speicherverwaltung (Memory Management Unit = MMU) zu hektischer Aktivität veranlassen. Wenn die CPU eine Bitkombination nicht versteht und wenn sich auch kein anderer Baustein im System angesprochen fühlt (etwa weil keine FPU im Sokkel steckt), kann die gewünschte Reaktion immer noch durch die Ausnahmeverarbeitung emuliert werden.

1.10 Privilege violation
Eine Privileg-Verletzung tritt auf, wenn sich der Prozessor in der Anwender-Betriebsart befindet und ein Befehl ausgeführt werden soll, der ausschließlich dem Supervisor (also normalerweise dem Betriebssystem) vorbehalten ist – auch bei Mikroprozessoren funktioniert der Betrieb nur, wenn einer den Daumen drauf hat...

Gruppe 2

2.1 Trap
Die Ausnahmeverarbeitung wird durch einen normalen Befehl eingeleitet. Hauptanwendung ist die Anforderung von Betriebssystem-Aktivitäten durch ein Anwenderprogramm.

2.2 Trapv
Während der Programmausführung kann bei Rechenoperationen ein zu großer Zahlenwert auftreten. Der Test mit einem Trapv-Befehl (neudeutsch: arithmetischer Überfluß) führt zur Ausnahmeverarbeitung.

2.3 CHK
Falls beim Test der Wert eines Registers kleiner als Null oder größer als eine obere Grenze ist, wird eine Ausnahmeverarbeitung eingeleitet.

2.4 Zero divide
Division durch Null gibt halt irgendwie immer Anlaß zum Nachdenken.

2.5 Format error
Dieser Fehler wird dem 68000 nie auffallen. 68010 und 68020 sind da etwas schlauer: sie testen am Ende einer Ausnahmeverarbeitung (beim RTE-Befehl), ob die auf dem Stack abgelegten Daten ihren hohen Ansprüchen genügen. Dieser Test ist zum Beispiel nützlich, wenn der Programmierer in einer Interrupt-Routine schlimme Dinge mit dem Stack anstellt. Der 68020 kennt selbstverständlich noch einige weitere Möglichkeiten, an irgendeinem Datenformat herumzunörgeln, speziell im Zusammenspiel mit Coprozessoren.
Innerhalb der Gruppen 0 und 1 sind die Ausnahmen nach ihrer Wichtigkeit geordnet: Bei gleichzeitigem Auftreten von zwei Ausnahmebedingungen kommt der

zuerst an die Reihe, der weiter oben steht. Die Ausnahmen von Gruppe 2 treten immer im Zusammenhang mit der Ausführung bestimmter Befehle auf und brauchen daher keinerlei Vorrang untereinander. Abschließend ist noch festzuhalten, daß die Ausnahmen der Gruppe 0 die höchste und die der Gruppe 2 die niedrigste Priorität besitzen.

Ausnahmen beim 68020

Der 68020 unterscheidet sogar fünf verschiedene Gruppen. Gegenüber seinen Vorgängern ist die Prioritäts-Reihenfolge etwas abgeändert. Ohne näher darauf einzugehen, hier nur kurz die Gruppeneinteilung:

0.0 Reset
1.0 Adreßfehler
1.1 Busfehler
2.0 Breakpoints, CHK's, CALLM, Traps, Division durch Null u.a.
3.0 illegale und nicht implementierte Befehle, Privileg-Verletzung u.a.
4.0 einige Coprozessor-Befehle
4.1 Trace
4.2 Interrupts

Um es noch einmal ganz klar zu sagen: Die Priorität der Ausnahmebedingung sagt nur etwas darüber aus, welche Ausnahme beim gleichzeitigen Auftreten mehrerer Bedingungen als erste akzeptiert wird. Die Ausnahmeverarbeitungsroutinen werden dann in umgekehrter (!) Reihenfolge abgearbeitet.
Als Beispiel soll hier das Ausführen eines Trap-Befehls im Trace-Modus bei Anliegen eines Interrupts dienen. Als erstes wird in die Trap-Ausnahme verzweigt. Bevor die Verarbeitung jedoch überhaupt losgeht, wird die Trace-Bedingung akzeptiert und in die zugehörige Ausnahmeverarbeitung gesprungen. Doch auch die tut vorläufig einmal nichts, denn nun kommt der Interrupt an die Reihe, der vollständig bearbeitet wird (vorausgesetzt, er schafft das ohne weitere Unterbrechung). Dann erfolgt der Rücksprung in die Trace-Verarbeitung und erst, wenn diese beendet ist, wird die Trap-Bedingung wirklich behandelt. Das gilt natürlich nicht beim Anliegen eines externen Reset-Signals, denn dabei werden alle anderen Ausnahmebedingungen gelöscht. Zwei oder mehr gleichzeitige Interrupt-Anforderungen werden ebenfalls etwas anders behandelt.
Dieses Beispiel funktioniert übrigens nur beim 68020, aber es zeigt das Prinzip. Wie später noch ausführlich beschrieben, stellen die übrigen Prozessoren jedes Tracing erst einmal ab, sobald sie irgend eine Ausnahme akzeptiert haben.

Beginn einer Ausnahmeverarbeitung

Reset, Busfehler und Interrupts (sowie Coprozessoren in 68020-Systemen) sind die einzigen externen Quellen, die eine Ausnahmeverarbeitung anfordern können. Alle anderen Ursachen werden innerhalb der CPU erkannt. Die Behandlung ist jedoch für alle Ausnahmen prinzipiell gleich: Der Prozessor verzweigt sofort in die zugehörige Ausnahmeverarbeitung. Nur Interrupts müssen unter Umständen ein wenig warten, wenn zur Zeit gerade ein anderer Interrupt verarbeitet wird. Bei einer akzeptierten Unterbrechung der Gruppe Null (Gruppe Null oder Eins beim 68020) wird der gerade bearbeitete Befehl abgebrochen, in allen anderen Fällen wird die Instruktion beendet und erst dann in die Ausnahmeverarbeitung verzweigt.
Der Prozessor schaltet dazu in die Supervisor-Betriebsart und rettet gewisse interne Informationen auf dem Stack. Anzahl und Reihenfolge dieser Daten sind einerseits vom vorliegenden Prozessor (68000 oder 68010/20) und andererseits von der Ausnahme selbst abhängig, darauf wird später noch näher eingegangen. In diesem Zusammenhang sollte nochmal daran erinnert werden, daß die Prozessoren zwei Betriebsarten kennen: Den Anwender-Modus, in dem gewisse Befehle verboten sind und sofort zu einer Ausnahmeverarbeitung führen (Privileg-Verletzung), und den Supervisor-Modus, in dem einfach Alles erlaubt ist.
Jeder Ausnahmebedingung ist eine Nummer zugeordnet, die sogenannte Vektornummer. Im allgemeinen wird diese Nummer innerhalb des Prozessors erzeugt, nur bei Interrupts und von Coprozessoren kann sie auch von der anfordernden Peripherie kommen. Die CPU verwendet diese Zahl (nach der Multiplikation mit vier) als Index für eine Tabelle, die die „Ausnahmevektoren" enthält, und lädt die dort gefundene Adresse direkt in den Programmzähler. Dadurch erfolgt ein Sprung zu einer Routine, die dann die eigentliche Ausnahmeverarbeitung vornimmt.

Die Ausnahmevektoren

Der erste Unterschied zwischen 68000 einerseits und 68010/68020 andererseits liegt nun in der Plazierung dieser Vektortabelle im Adreßbereich des Mikroprozessors. Beim 68000 beginnt die Tabelle grundsätzlich bei Adresse $0, die beiden anderen Prozessoren besitzen ein Vektorbasisregister, dessen Inhalt zu der mit vier multiplizierten Vektornummer addiert wird, d. h. bei diesen Prozessoren kann die Vektortabelle (*Tabelle 1*) irgendwo im Adreßbereich untergebracht werden.
Ein Beispiel soll die Verhältnisse verdeutlichen: Nehmen wir wieder den Fall eines Busfehlers. Die zugehörige Vektornummer ist zwei. Der Index für die Tabelle ist somit 4 x 2 = 8 (jeder Adresseintrag belegt 4 Bytes) und der 68000 sucht nun unter der Adresse $8 die Startadresse der Busfehler-Ausnahmeverarbeitungs-Routine (ich liebe diesen Ausdruck!). 68010 und 68020 machen hier nicht so kurzen Prozeß: sie addieren erst noch den Inhalt ihres Vektorbasisregisters dazu, z. B. 8 + 76592 = 76600, und suchen dort nach dem Start der Ausnahmeverarbeitung. Eine Anmerkung am Rande: Der Wert des Vektorbasisregisters muß natürlich gerade sein, sonst hat man zusätzlich auch noch einen Adreßfehler am Hals. Womit auch gleich offensichtlich wird, warum die Adreßfehler-Ausnahme die zweithöchste Priorität besitzt.

Der letzte Befehl einer Ausnahmeverarbeitungs-Routine ist immer ein RTE (Return from Exception). Der Prozessor holt die gleiche Anzahl von Worten vom Stack, die er zu Beginn der Ausnahmeverarbeitung darauf abgelegt hat, und restauriert damit wieder seinen ursprünglichen Zustand. Wenn es sich irgendwie einrichten läßt, sollte er dabei auch einigermaßen sinnvolle Informationen vorfinden. 68010 und 68020 testen diese Daten daher auf Integrität und Plausibilität (wenn man weiß, wie, kann man sie natürlich immer noch hereinlegen).

Interrupts im Detail

Nun soll das spezielle Augenmerk auf die Interrupt-Behandlung gelegt werden. Sie unterscheidet sich von den anderen Ausnahmeverarbeitungen nur in einem Punkt: Die Vektornummer der Ausnahme ist nicht schon von vornherein bekannt, sondern sie wird von der CPU erst in einem zusätzlichen Interrupt-Bestätigungs-Zyklus beschafft.
Eine der wichtigsten Aufgaben eines Mikroprozessors besteht im Reagieren auf äußere Einflüsse. Prinzipiell hat der Prozessor zwei Möglichkeiten, auf die Bedürfnisse der Peripherie einzugehen: Er kann in einer Schleife dauernd ein Gerät nach dem anderen abfragen, ob sich dort etwas tut („polling"), oder er kann dar-

auf vertrauen, daß sich der Baustein schon melden wird, wenn er etwas will – eben per Interrupt. Welche der beiden Methoden im Einzelfall günstiger ist, hängt von der Anwendung und den Umständen ab. Polling ist sicher einfacher zu programmieren und zu überschauen, man wird es einsetzen, wenn der Prozessor ohnehin auf eine Eingabe warten muß und nichts Anderes zu tun hat. Die Verwendung von Interrupts bietet sich vor allem bei seltenen Ereignissen (Tastatur-Eingaben) und bei zeitlich periodischen Vorgängen an. Für ein Multitasking-System ist der Interrupt durch einen Timer-Baustein sogar zwingend notwendig.

Wenn in einem Computer Interrupt-Betrieb möglich ist, dann müssen einige ganz wesentliche Grundregeln beachtet werden:

1. Der Hardware-Entwickler muß dafür sorgen, daß ein Interrupt richtig erkannt und behandelt wird (klingt simpel, ist es aber nicht, wie noch gezeigt wird).

2. Der Programmierer muß dafür sorgen, daß nach dem Interrupt das unterbrochene Programm so weiterläuft, als ob der Interrupt gar nicht stattgefunden hätte. Das bedeutet im Einzelnen, daß alle (!) Register der CPU wieder im ursprünglichen Zustand sein müssen, und es ist allergrößte Vorsicht geboten, wenn eine Interruptroutine in einem Speicherbereich herumfuhrwerkt, der eigentlich einem anderen Programmteil gehört.

3. Wenn ein Interrupt möglich ist, dann ist auch mehr als einer möglich! Diese Aussage klingt so paradox, daß sie der unerfahrener Programmierer gerne übersieht. Die Konsequenz ist jedoch, daß eine Interruptroutine selbst wieder durch eine Ausnahmeverarbeitung mit höherer Priorität unterbrochen werden kann. Und genau hier passieren die häufigsten Fehler, denn es ist (wenn man nicht aufpaßt) relativ einfach, eine Interrupt-Routine zu schreiben, die sich nicht straflos unterbrechen läßt.

Erkennen und Vorbereiten

Im Sinne dieser Beschreibung liegt die Quelle des Interrupts außerhalb der CPU. Deshalb sind (im Gegensatz zu den anderen Ausnahmen) einige zusätzliche Schritte notwendig. Bevor die eigentli-

Tabelle 1:
Vektornummer, relative Adresse (Offset) des Vektors und Ausnahmebedingung

Vektornr. dez.	Vektornr. hex.	Offset dez.	Offset hex.	Ausnahmebedingung
0	00	0	000	Reset, Anfangswert des SSP
	–	4	004	Reset, Anfangswert des PC
2	02	8	008	Busfehler
3	03	12	00C	Adreßfehler
4	04	16	010	ungültiger Befehl
5	05	20	014	Division durch Null
6	06	24	018	CHK-Befehl (Register-Wert testen)
7	07	28	01C	TRAPV-Befehl (auf Overflow testen)
8	08	32	020	Privileg-Verletzung
9	09	36	024	TRACE-Modus (Einzelschritt)
10	0A	40	028	Emulation von $Axxx-Befehlen (%1010...)
11	0B	44	02C	Emulation von $Fxxx-Befehlen (%1111...)
12	0C	48	030	reserviert
13	0D	52	034	Coprozessor-Protokollfehler (68020)
14	0E	56	038	Formatfehler (68010 und 68020)
15	0F	60	03C	nichtinitialisierter Interrupt
16	10	64	040	reserviert
...	...	...	...	reserviert
23	17	92	05C	reserviert
24	18	96	060	Spurious Interrupt (Ausn. + Busfehler)
25	19	100	064	Interrupt-Autovektor Priorität 1
26	1A	104	068	Interrupt-Autovektor Priorität 2
27	1B	108	06C	Interrupt-Autovektor Priorität 3
28	1C	112	070	Interrupt-Autovektor Priorität 4
29	1D	116	074	Interrupt-Autovektor Priorität 5
30	1E	120	078	Interrupt-Autovektor Priorität 6
31	1F	124	07C	Interrupt-Autovektor Priorität 7
32	20	128	080	TRAP-0-Befehl
33	21	132	084	TRAP-1-Befehl
34	22	136	088	TRAP-2-Befehl
35	23	140	08C	TRAP-3-Befehl
36	24	144	090	TRAP-4-Befehl
37	25	148	094	TRAP-5-Befehl
38	26	152	098	TRAP-6-Befehl
39	27	156	09C	TRAP-7-Befehl
40	28	160	0A0	TRAP-8-Befehl
41	29	164	0A4	TRAP-9-Befehl
42	2A	168	0A8	TRAP-10-Befehl
43	2B	172	0AC	TRAP-11-Befehl
44	2C	176	0B0	TRAP-12-Befehl
45	2D	180	0B4	TRAP-13-Befehl
46	2E	184	0B8	TRAP-14-Befehl
47	2F	188	0BC	TRAP-15-Befehl
48	30	192	0C0	reserviert für FPU (68020)
...	...	...	...	reserviert für FPU (68020)
54	36	216	0D8	reserviert für FPU (68020)
55	37	220	0DC	reserviert
56	38	224	0E0	reserviert für MMU (68020)
57	39	228	0E4	reserviert für MMU (68020)
58	3A	232	0E8	reserviert für MMU (68020)
59	3B	236	0EC	reserviert
...	...	...	...	reserviert
63	3F	252	0FC	reserviert
64	40	256	100	192 Interrupt-Vektoren ...
...	...	...	...	...frei ...
255	FF	1020	3FC	... für Anwender.

che Ausnahmeverarbeitung losgeht, läuft im Wechselspiel zwischen Prozessor und beteiligter Hardware folgendes Ritual ab:

- Das anfordernde Gerät teilt der CPU mit, daß es gern bedient werden möchte. Dabei gibt es seine Priorität bekannt, die vom Benutzer vorher festgelegt wurde (in ganz intelligenten Systemen kann diese Einstellung sogar per Software erfolgen). Es kommen die Stufen 1...7 in Frage, wobei 7 die höchste Priorität besitzt. Verschiedene Interrupt-Quellen können durchaus die gleiche Priorität besitzen.

- Der Mikroprozessor vergleicht mit seiner internen Interruptmaske, ob die angemeldete Vorrangstufe hoch genug ist, um das laufende Programm zu unterbrechen. Wenn die Maske etwa die Zahl vier enthält, so werden nur Interrupts der Stufen 5, 6 oder 7 akzeptiert. Die höchste Priorität, also die Stufe sieben, kann nicht maskiert werden, diese Interrupts werden also immer angenommen.

- Die CPU vollzieht dann einen Buszyklus zur Interruptbestätigung, auf dessen Einzelheiten weiter unten eingegangen wird. Das anfordernde Gerät muß nun seinerseits erkennen, daß sein eigener Interrupt behandelt wird und daß die CPU gern wissen möchte, wo die entsprechende Interrupt-Ausnahmebehandlungs-Routine zu finden ist.

- Nachdem der Prozessor nun von der Hardware den heißen Tip bekommen hat, welcher Ausnahmevektor zu verwenden sei, fängt die eigentliche Ausnahmeverarbeitung an.

Beteiligung der CPU

Beim Erkennen und Akzeptieren eines Interrupts sind einige CPU-Register beteiligt: Statusregister, Stackpointer und Vektorbasisregister. Die Außenwelt muß einigen Beinen des Käfers besondere Aufmerksamkeit widmen, speziell den Interrupt-Kontroll-Leitungen, den Prozessor-Status-Signalen, dem Adreßbus und den Kontrollsignalen $\overline{BERR}$ und $\overline{VPA}$. Zwischen den einzelnen Familienmitgliedern bestehen auch hier einige Unterschiede, auf die an den betreffenden Stellen eingegangen wird.
Der Prozessor hat intern die Möglichkeit, einzelne Interrupts zu sperren. Indem er die Interruptmaske auf einen Wert von 0...7 setzt, kann er alle Interrupt-Anforderungen mit einer Priorität kleiner oder gleich der Maske abwimmeln (0 = alles ist erlaubt, 6 oder 7 = nur Priorität 7 kommt durch).
Die Bits 0...7 des Statusregisters (*Bild 1*) enthalten die üblichen Flags für Carry, Overflow, Zero, Negativ und Extend. Sie bilden zusammen das sogenannte „user byte", auf das von allen Anwenderprogrammen aus zugegriffen werden kann. Die hier interessierende Interruptmaske befindet sich in den Bits 8, 9 und 10. Bit 13 (Supervisor-Status) zeigt an, ob sich der Prozessor in der Anwender- (S=0) oder Supervisor-Betriebsart (S=1) befindet. Alle Ausnahmebehandlungen, also auch die Interrupts, werden grundsätzlich im Supervisor-Modus behandelt. Das S-Bit zeigt außerdem noch an, ob der Supervisor- oder der Anwender-Stackpointer in Betrieb ist. Beim 68020 wird zusätzlich im Supervisor-Modus zwischen Master-Stackpointer (M=1) und Interrupt-Stackpointer (M=0) unterschieden.
Bit 15 (Trace-Bit) bei 68000 und 68010 bzw. Bit 15 und 14 bei 68020 steuern die Abarbeitung eines Programms im Einzelschritt-Verfahren. Falls diese Betriebsart eingeschaltet ist, wird nach jedem Befehl eine Einzelschritt-Ausnahmeverarbeitung durchgeführt (unter Verwendung von Vektor Nr. 9, siehe Tabelle 1).

Das obere Byte des Statusregisters (Bit 8...15) ist das sogenannte Systembyte, wie man sieht, haben alle Felder irgend etwas mit Ausnahmeverarbeitung zu tun. Damit nun nicht jedes x-beliebige Anwenderprogramm z. B. die Interruptmaske verändern (wichtig bei Multitasking) oder in den Einzelschritt-Modus schalten kann (wichtig bei mehreren Benutzern), darf das Systembyte nur modifiziert werden, wenn sich der Prozessor in der Supervisor-Betriebsart befindet. Jeder diesbezügliche Versuch im Anwender-Modus führt automatisch zu einer Ausnahmeverarbeitumng über Vektor Nr. 8 (Privileg-Verletzung).

Die Stackpointer

Irgend ein schlauer Mensch hat schon vor langer Zeit durchgesetzt, daß alle gängigen Mikroprozessoren Stackmaschinen zu sein haben. Die Prozessoren der Reihe 680xx sind – noch einen Schritt weiter – typische Hochsprachen-Prozessoren: Ihr Befehlssatz ist an den heute üblichen strukturierten Programmiersprachen orientiert. Darüber hinaus erfolgt eine konsequente Trennung zwischen Supervisor- und Anwender-Betriebsart. Wenn man einmal voraussetzt, daß das Betriebssystem des Computers im Supervisor-Modus läuft und alles andere in der Anwender-Betriebsart, so ist es nur logisch, wenn auch zwei separate Stackpointer vorgesehen sind: Nur dadurch ist es nämlich möglich, ein Betriebssystem zu programmieren, das durch ein Anwenderprogramm unter keinen Umständen zum Absturz gebracht werden kann. Um die ganze Sache noch ein bißchen sicherer (und für den armen Programmierer noch etwas komplexer) zu machen, hat man dem 68020 zusätzlich noch einen eigenen Interrupt-Stackpointer spendiert. Während des Betriebes ist also je nach Zustand der CPU das Register A7 oder A7' (oder A7'' beim 68020) als Stackpointer in Funktion.

Ein Stack ist die einfachste Form dynamischer Speicherplatzverwaltung. Im Zusammenhang mit Interrupts wird er hauptsächlich dazu verwendet, CPU-interne Informationen (Inhalt von Programmzähler, Statusregister usw.) aufzubewahren, damit am Ende der Ausnahmeverarbeitung der ursprüngliche Betriebszustand wieder hergestellt werden kann.

Die Statussignale

Die CPU zeigt der Außenwelt über die drei Funktionscode-Leitungen FC0...FC2 an, von welcher Art der laufende Buszugriff ist (*Tabelle 2*). Die Unterscheidung zwischen Anwenderdaten, -programm, Supervisordaten und Supervisorprogramm kann dazu verwendet werden, den physikalischen Adreßraum des Prozessors zu vervierfachen. Für unsere Betrachungen sind folgende Feststellungen wichtig:

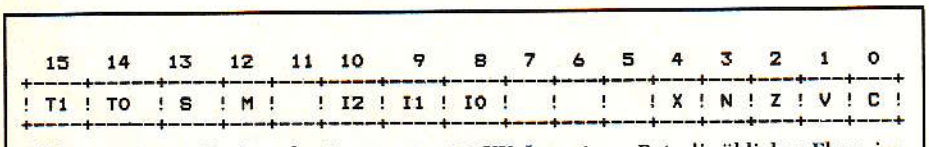

Bild 1. Das Status-Register der Prozessoren 680XX: Im unteren Byte die üblichen Flags, im oberen Byte (Bit 8...15) die Interrupt-Maske, Trace- und Status-Flags. T0 und M sind nur beim 68020 vorhanden

– Alle Ausnahmevektoren (Tabelle 1) außer dem Reset-Vektor (Nr. 0 und 1) liegen im Supervisor-Datenbereich.

– Der Resetvektor liegt im Supervisor-Programmbereich.

– Der Stack für Ausnahmeverarbeitung liegt im Supervisor-Datenbereich.

Von besonderem Interesse für Interrupts ist der CPU-Bereich, der beim 68000 nur „interrupt acknowledge" heißt und auch nur während der Interrupt-Bestätigung auftritt. Bei 68010 und 68020 kommt dieser Code zusätzlich bei der Breakpoint-Bestätigung vor, und der 68020 benutzt diesen Bereich außerdem für die Steuerung einer Speicherverwaltung (MMU) und die Kommunikation mit Coprozessoren. Diese unterschiedlichen Verwendungen des CPU-Bereiches erfordern eine gewisse Aufmerksamkeit beim Hardware-Design eines Mikrocomputers, wenn verschiedene Mitglieder der 680xx-Familie eingesetzt werden sollen.

Interrupt-Ablauf

Wie wird nun die ganze Sache zwischen den beteiligten Parteien abgewickelt – oder mal banal ausgedrückt: Welche Signale geistern durch die Hardware, bis die eigentliche Interrupt-Ausnahmeverarbeitung endlich losgehen kann?
Der Prozessor besitzt drei Eingangsleitungen ($\overline{IPL0}$...$\overline{IPL2}$), über die ein Interrupt angemeldet werden kann. In diesen drei Bit ist gleichzeitig die angeforderte Interrupt-Priorität binär codiert: Liegen alle drei Eingänge auf High-Pegel, so entspricht dies der Priorität 0, es liegt kein Interrupt an. Liegen alle drei Eingänge auf Masse, so enspricht das dem nicht maskierbaren Interrupt (Priorität 7). In den meisten 68000-Mikrocomputern werden diese Eingänge nicht direkt angesteuert, sondern über einen Prioritäts-Encoder (74LS148 oder ähnliche), so daß für die externe Hardware die sieben Prioritätsstufen auf getrennten Leitungen zur Verfügung stehen. So braucht ein Baustein zur Anforderung eines Interrupts nur die seiner Priorität zugeteilte Leitung auf Masse zu ziehen und abzuwarten, was passiert.
Ein beliebter Anlaß für Verwirrung ist übrigens die negative Logik der Interrupt-Eingänge. Um zum Beispiel einen Interrupt der Priorität 4 anzumelden, müssen die drei Leitungen mit „011" beschaltet werden – das sieht aus wie eine binäre Drei. Erst wenn man nun die Bit für Bit invertiert (Einer-Komplement), erhält man „100" und damit die gesuchte Prioritätsstufe 4. Der erwähnte Prioritätsencoder 74LS148 arbeitet ebenfalls in negativer Logik, d. h. wenn etwa sein Eingang Nr. 5 als höchster eine Null aufweist (also die Eingänge 6 und 7 auf Eins liegen, die Eingänge 0...4 sind unwichtig), erscheint am Ausgang „010" und damit wieder genau das Einerkomplement von %101 = 5. Damit sind jetzt hoffentlich alle Klarheiten restlos beseitigt.

Tabelle 2: Bedeutung der Funktions-Codes

Status	FC2	FC1	FC0	Zugriffsart
0	0	0	0	nicht definiert, reserviert
1	0	0	1	Anwender Daten
2	0	1	0	Anwender Programm
3	0	1	1	nicht definiert, reserviert
4	1	0	0	nicht definiert, reserviert
5	1	0	1	Supervisor Daten
6	1	1	0	Supervisor Programm
7	1	1	1	CPU-Bereich

Bei den Interrupteingängen leistet sich der 68008 den einzigen funktionalen Unterschied zum 68000: Da ein Bein eingespart werden mußte, besitzt der 68008 nur zwei „tretbare Schienbeine", nämlich $\overline{IPL0/2}$ und $\overline{IPL1}$. $\overline{IPL0}$ und $\overline{IPL2}$ sind intern miteinander verbunden, und damit versteht der Prozessor nur die Prioritäten 0 (kein Interrupt), 2, 5 und den nicht maskierbaren Interrupt mit Priorität 7.

Interruptbestätigung

Der Prozessor liest periodisch den Zustand seiner Interrupt-Eingangsleitungen, während er sein Mikroprogramm abarbeitet. Wenn der Vergleich mit der Interruptmaske im Statusregister eine erhöhte Priorität oder den Wert 7 anzeigt, so werden von der CPU folgende Schritte ausgeführt:

(1) Der Inhalt des Status-Registers wird intern zwischengespeichert.

(2) Im Statusregister wird das S-Bit gesetzt, d. h. der Prozessor schaltet in die Supervisor-Betriebsart (vgl. Bild 1).

(3) Der Trace-Modus wird abgeschaltet. Während der Ausnahmeverarbeitung gibt es also keine Einzelschritt-Verarbeitung, es sei denn, die Interrupt-Routine selbst schaltet diesen Modus wieder ein.

(4) Die Interrupt-Maske im Statusregister wird auf den aktuellen Wert gebracht, d.h. auf die Prioritätsstufe des gerade akzeptierten Interrupts. Bei einem externen Reset wird die Maske auf 7 gesetzt, womit alle Interrupts außer dem nicht-maskierbaren (NMI) gesperrt werden.

(5) Die CPU bestätigt die Annahme des Interrupts durch eine spezielle Form des Lesezyklus. Die Funktionscode-Leitungen zeigen einen Zugriff auf den CPU-Bereich an (Tabelle 2). Die Adreßleitungen A4...A23 (...A31 beim 68020) sind alle auf 1 gesetzt. Die Adreßleitungen A1...A3 zeigen eine exakte Kopie der neuen Interrupt-Maske des Statusregisters, also das Inverse der Eingangsleitungen $\overline{IPL0}$...$\overline{IPL2}$ und somit die Priorität des akzeptierten Interrupts. Unter dieser Pseudo-Adresse im CPU-Bereich möchte der Prozessor nun dringend die Vektornummer (Tabelle 1) lesen, über die er den Start der Ausnahmeverarbeitungs-Routine finden kann. Der 68020 aktiviert zusätzlich seine $\overline{IPEND}$-Leitung (logisch Null) um anzuzeigen, daß als nächstes eine Interrupt-Bestätigung abläuft.

(6) Nun ist es an der Hardware, dem Informationsdefizit der CPU abzuhelfen. Zuerst einmal muß ihr dazu auffallen, daß ihr eigener Interrupt jetzt behandelt wird, was ja am Zustand der Funktionscode- und Adreßleitungen zu erkennen ist. Hat die Außenstelle gemerkt (oder auch nicht), daß sie gemeint ist, so hat sie drei Möglichkeiten, zu reagieren:

(6.1) Beispielsweise gar nicht, was zweifellos das Bequemste ist. In diesem Fall meldet sich nach einer gewissen Zeit der sowieso (hoffentlich) vorhandene Zeitgeber, z. B. ein Monoflop, und signalisiert einen Busfehler. Im Normalfall reagiert der

Prozessor auf eine Busfehler-Ausnahme mit einem Sprung über Vektor Nr. 2 (Tabelle 1). Tritt der Busfehler jedoch während einer Interrupt-Bestätigung auf, so verwendet die CPU den Vektor Nr. 24 (Spurious Interrupt). Da es jetzt plötzlich niemand gewesen sein will, war der Interrupt offenbar ungewollt (Langenscheidt: spurious = unehelich).

(6.2) Eine etwas höflichere Hardware sollte allerdings der CPU den Gefallen tun, den Pseudo-Lesezyklus ganz normal zu beenden: Sie legt eine Zahl (0...255) auf den Datenbus und aktiviert die $\overline{\text{DTACK}}$-Leitung (oder entsprechend $\overline{\text{DSACK0}}$ und $\overline{\text{DSACK1}}$ beim 68020). Der Prozessor bekommt also seine Vektornummer und alle sind glücklich und zufrieden.

Anmerkung 1:
Es ist unmoralisch, wenn in diesem Fall die Vektornummer kleiner als 64 ist, da die Vektoren 0...63 ja bereits für andere Aufgaben reserviert sind (siehe Tabelle 1), aber man kann natürlich...

Anmerkung 2:
Fragen der Moral verlieren schlagartig jede Bedeutung, wenn es um Geld geht. Eine kleine (selbstverständlich frei erfundene) Geschichte mag das verdeutlichen: Es war einmal eine große Firma, die eine Internationale BüroMaschine unter Verwendung eines INTELligenten Mikroprozessors eines großen Halbleiterherstellers gebaut hatte. Dazu wurde bei einem berühmten Programmschreiber, nennen wir ihn mal MikroWeich, das stolze und erfolgreiche Betriebssystem MW-DOS bestellt. Viele, viele Einheiten wurden an den Mann gebracht, und alle lebten angenehm und sorgenfrei. Nun begab es sich aber, daß der INTELligente Baustein auch einige für zukünftige Entwicklungen reservierte Vektornummern besaß, aber dummerweise machte MW-DOS fleißig Gebrauch von diesen verbotenen Früchten. Als nun einige Zeit später der noch INTELligentere Nachfolgebaustein in großen Stückzahlen auf den Markt geworfen wurde, kam plötzlich Erstaunen auf: Die vom Betriebssystem so eifrig benutzten Vektoren hatten plötzlich andere Aufgaben und waren somit für MW-DOS absolut unverdaulich. Jetzt war guter Rat teuer. Auf der einen Seite eine riesige Anzahl installierter Betriebssysteme, auf der anderen ein entnervter Halbleiter-Hersteller, der andauernd nur so etwas wie „aber das war doch reserviert" vor sich hin murmelte. Das Ende der Geschichte: Ein paar tausend hochINTELligente Bausteine wurden zurückgerufen, denn schließlich kostet die Änderung einer Mikroprozessor-Maske nur ein paar Millionen Dollar, und dessen Kurs sinkt ja ohnehin immer tiefer. Verglichen mit den Kosten für eine Revision von MW-DOS ist das jedenfalls ein Klacks. Was lernen wir daraus? Jeder darf ruhig einen reservierten Vektor verwenden, denn Motorola wird ganz sicher beim Redesign ihres 68030 darauf Rücksicht nehmen. Vorausgesetzt natürlich, man hat schon genügend Kasse mit dem Vektor gemacht...
Doch nun zur dritten Möglichkeit, wie die CPU endlich Ihre Vektornummer finden kann:

(6.3) 68000 und 68010 besitzen einen $\overline{\text{VPA}}$-Eingang (valid peripheral address), der zusammen mit dem Enable- und dem $\overline{\text{VMA}}$-Ausgang die Synchronisation mit den altbewährten (und recht betagten) Peripheriebausteinen der Serie 6800 ermöglicht. Während der Interrupt-Bestätigung jedoch besitzt dieser Eingang eine völlig andere Bedeutung: Wenn die Außenstelle die $\overline{\text{VPA}}$-Leitung aktiviert (auf logisch Null), dann merkt die CPU, daß sie vergeblich auf eine Vektornummer von der Peripherie wartet. Sie beendet Ihren Pseudo-Lesezyklus und berechnet die gesuchte Vektornummer direkt aus der Priorität des Interrupts, indem sie zu dieser Priorität 24 addiert (vergleiche Tabelle 1). Der zugehörige Vektor ist dann der sogenannte Interrupt-Autovektor.

Da niemand reinen Gewissens einem 68020 zumuten wollte, mit der guten alten 6800-Peripherie zusammenzuarbeiten, sind bei ihm die entsprechenden Extremitäten im Sinne der Evolution völlig verkümmert: als Ersatz für $\overline{\text{VPA}}$ gibt es nun nur noch einen Autovektor-Eingang $\overline{\text{AVEC}}$, womit wir wieder kompatibel wären.

Zusammenfassung

Reagiert die Außenwelt nicht auf eine Interruptbestätigung der CPU, so erfolgt die Ausnahmeverarbeitung über den Spurious-Interrupt-Vektor. In einem normalen Pseudo-Lesezyklus liest der Prozessor die Vektornummer über den Datenbus, wobei nur die unteren 8 Bit gültig sind. Wird dagegen die $\overline{\text{VPA}}$-Leitung ($\overline{\text{AVEC}}$ beim 68020) aktiv (low), dann startet die Ausnahmeverarbeitung über den Autovektor, der zur Prioritätsstufe des Interrupts gehört.
Die Schritte 5...6 dienen nur dazu, die Interruptquelle zu lokalisieren; sie kommen daher nur bei Interrupts vor. Die vorbereitenden Schritte 1...3 und alle sonstigen Schritte sind nicht interruptspezifisch, sondern werden grundsätzlich bei jeder Ausnahmebehandlung durchlaufen. *(wird fortgesetzt)*

Der Mensch im CIM-Mittelpunkt

Kreativität, Erfindergeist und Engagement des Werkstattmanns können für ein CIM-Konzept wichtiger sein als Maschinenauslastung und Materialausnutzung. Dies ist der Kern des neuen Esprit-Projektes Nr. 1217, das unter dem Namen „Human Centered CIM-System" im Mai 1986 gestartet wurde. Innerhalb von drei Jahren entwickeln und erproben drei nationale Arbeitsgruppen je ein Softwarepaket für CAP, CAM und CAD mit einem Investitionsvolumen von insgesamt 13,2 Millionen DM. In der deutschen Arbeitsgruppe beteiligt sich Krupp Atlas neben der Universität Bremen am CAP-System. Ziel des Vorhabens ist ein flexibles Fertigungssystem für die auftragsorientierte Einzel- und Kleinserienfertigung. Dabei sollen die menschlichen Interventionsmöglichkeiten für Disposition, Überwachung und Steuerung auf Werkstattebene verstärkt genutzt werden. Das Wissen und die Erfahrung des Maschinenbedieners soll in ein CIM-System einfließen, daß auf geänderte Anforderungen und Störfälle schneller reagiert und damit robuster, effizienter und wirtschaftlicher arbeitet. Erstmals kommen in einem Esprit-Projekt Sozialwissenschaftler und Arbeitspsychologen zum Zuge. Neben den fertigungstechnischen Aspekten werden auch arbeitswissenschaftliche Aspekte berücksichtigt. Daraus werden sich Konsequenzen auch für die Berufsausbildung ergeben. An dem Projekt arbeiten neben den deutschen Arbeitsgruppen dänische (CAD) und englische (CAM) mit.

Nach Unterlagen von Krupp