2. Komponenten eines Mikroprozessors

In diesem Kapitel werden wir nun hauptsächlich die internen Komponenten eines einfachen Mikroprozessors (nach Bild 1.3-6), so wie er auch heute noch z.B. als Kern in 8- und 16-bit-Mikrocontrollern implementiert wird, ausführlich beschreiben. Dabei werden wir jedoch häufiger auch schon auf Erweiterungen eingehen, die für die modernen Hochleistungsprozessoren typisch sind.

2.1 Steuerwerk

2.1.1 Funktion und Aufbau

Das Steuerwerk (*Control Unit* – CU) eines Mikroprozessors ist ein synchrones Schaltwerk. Bei einigen, insbesondere den älteren Mikroprozessortypen, wird die Erzeugung des benötigten Takts extern in einem besonderen Baustein vorgenommen. Moderne Mikroprozessoren haben den Taktgenerator *(Clock Generator)* meist auf dem Chip integriert, so daß zur Stabilisierung der Frequenz extern nur noch ein Quarz zwischen zwei Anschlüsse des Prozessors gelegt werden muß. Im Taktgenerator wird durch einen freischwingenden Oszillator eine Rechteckschwingung mit fester Frequenz erzeugt. Heute sind Taktraten zwischen 10 und 2000 MHz üblich; die 1-GHz-Grenze wurde im Jahr 2000, die 2 GHz im Jahr 2001 erreicht. Da einige Mikroprozessoren dynamische Schaltwerke sind, bei denen die Information in Kondensatoren gespeichert ist, darf bei ihnen die Taktfrequenz nicht unter einen Minimalwert fallen, sonst werden die Inhalte der internen Register nicht rechtzeitig aufgefrischt und gehen durch Leckströme verloren. Der Prozessortakt wird über einen Ausgang meist allen externen Komponenten als Systemtakt zur Verfügung gestellt, gegebenenfalls auf eine niedrigere Frequenz heruntergeteilt.

Häufig werden im Taktgenerator noch weitere Zeitfunktionen abgeleitet. Dazu gehört beispielsweise die Erzeugung eines mit dem Prozessortakt synchronisierten Rücksetzsignals. Dieses Signal wird durch einen asynchron am RESET-Eingang angelegten Impuls angestoßen und muß genaue zeitliche Spezifikationen erfüllen, um dem Prozessor eine geordnete Initialisierung zu ermöglichen. Diese wird durch ein (Mikro-) Programm des Steuerwerks durchgeführt und besteht vor allem darin, die internen Register mit bestimmten Anfangswerten zu laden. Das Rücksetzsignal wird bei einigen µP-Typen über einen RESET-Ausgang auch allen anderen Komponenten des Systems zugeführt.

2.1.1.1 Exkurs: Erzeugung eines RESET-Signals

Im Bild 2.1-1a ist eine kombinierte Schaltung dargestellt, die sowohl ein automatisches Rücksetzen des Prozessors nach dem Einschalten der Betriebsspannung (*Power on Reset* – POR) als auch ein manuelles Rücksetzen über einen Taster ermöglicht.

Zur Funktion: Nach dem Einschalten der Spannungsversorgung (im Zeitpunkt t=0) steigt die Betriebsspannung $+U_B$ sehr rasch auf ihren Endwert (z.B. +5 V, s. Bild 2.1-1b). Die dabei auftretende Verzögerung ist hauptsächlich dadurch bedingt, daß zunächst alle Kapazitäten des Systems aufgeladen werden müssen. Dazu gehören insbesondere die Kondensatoren mit unterschiedlichsten Aufgaben, aber auch die Störkapazitäten der Leiterbahnen, der Gattereingänge und der Transistoren. Nachdem die Betriebsspannung ihren Endwert erreicht hat, beginnt der Oszillator mit der Ausgabe des Systemtakts. Mit der durch die Widerstands-Kondensator-Kombination vorgegebenen Zeitkonstanten $\tau = R \cdot C$, im Beispiel $\tau = 200$ ms ($1\ ms = 10^{-3}$ Sekunden), wird der Kondensator C allmählich (exponentiell) aufgeladen.

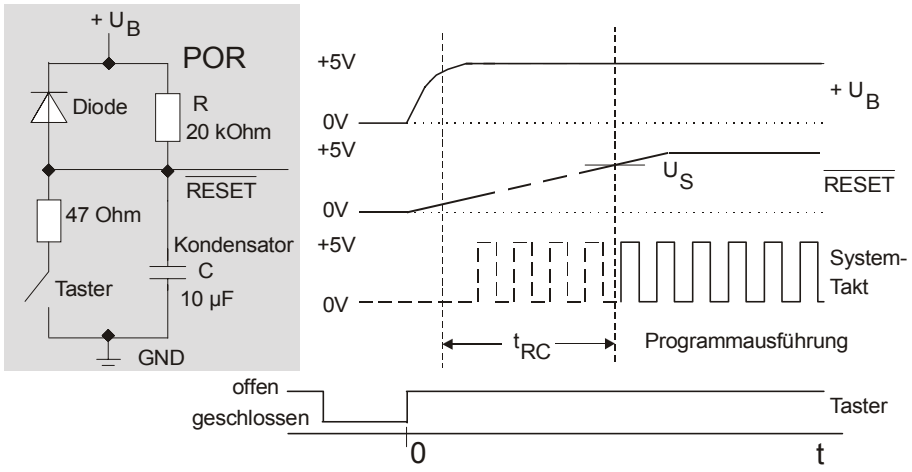

Bild 2.1-1: Schaltung zum Rücksetzen des Prozessors[1]

Diese Zeitkonstante τ muß bei einigen µP-Typen so groß gewählt werden, daß die eben beschriebene Initialisierungsroutine im Prozessor vollständig ausgeführt werden kann. Vom Hersteller wird die benötigte Zeit t_{RC} durch eine Mindestanzahl von Taktzyklen angegeben. Erst wenn die Kondensatorspannung (am Ausgang RESET) einen bestimmten Schwellwert U_S überschreitet, wird mit der nächsten Schwingung des Systemtakts (bzw. eine fest vorgegebene Anzahl von Schwingungen später) der Programmzähler (s. Bild 1.3-6) mit der Adresse des ersten auszuführenden Maschinenbefehls geladen und dadurch die Programmausführung gestartet. (Bei anderen Prozessor-

[1] 1 µF (Mikrofarad) = 10^{-6} Farad, 1 Farad = 1 As/V (Ampere·Sekunde/Volt)

typen wird die Initialisierung erst nach dem Erreichen des Schwellwerts US und vor der Ausführung des ersten Maschinenbefehls durchgeführt.)

Wird während des laufenden Betriebs der Rücksetz-Taster betätigt, also geschlossen, so kann sich über ihn der Kondensator C gegen Masse entladen. Der Reihenwiderstand (47 Ω) sorgt dabei für eine Begrenzung des Entladestroms. Nach Lösen des Tasters können nun wiederum – wie oben beschrieben – die Initialisierungsroutine ablaufen und der Kondensator C sich über den Widerstand R erneut aufladen.

Der Vollständigkeit halber sei angemerkt, daß die Diode als Stromventil wirkt, das einen Strom nur dann zum UB-Anschluß fließen läßt, wenn dort ein niedrigeres Potential liegt als am RESET-Anschluß. Sie sorgt dafür, daß nach dem Abschalten der Betriebsspannung (UB=0 V) der Kondensator schnell entladen wird, und damit der Eingang RESET des Prozessors nicht auf hohem Potential liegen bleibt.

2.1.1.2 Ablauf der Befehlsbearbeitung

Der Aufbau des Steuerwerks wurde bereits kurz im Unterabschnitt 1.3.3 skizziert. Darin kann man das Befehlsregister, den Befehlsdecod(ier)er, die eigentliche Steuerlogik sowie das Steuerregister unterscheiden. Das Steuerwerk des Mikroprozessors liefert nach einem durch ein Programm vorgegebenen Ablauf Steuersignale in festgelegter zeitlicher Reihenfolge an das Rechenwerk, an die anderen Komponenten des Prozessors und über die Prozessoranschlüsse auch an alle externen Komponenten des Mikroprozessor-Systems. Von einigen internen und externen Komponenten treffen Meldesignale ein, die vom Steuerwerk zu Entscheidungen über den weiteren Ablauf herangezogen werden. Dazu gehören insbesondere Programmverzweigungen und -unterbrechungen[2]. Ein Programm für den Mikroprozessor besteht aus einer Folge von Maschinenbefehlen (Makrobefehle). Diese werden vom Steuerwerk im Maschinencode sukzessiv in den Prozessor geladen. (Auf den Aufbau der Makrobefehle gehen wir in Abschnitt 3.2 ausführlich ein.) Bild 2.1-2 zeigt die Phaseneinteilung der Befehlsbearbeitung[3] im Prozessor.

- In der **Holphase** *(Fetch Cycle)* werden die Maschinenbefehle vom Steuerwerk ins Befehlsregister geladen. Genauer gesagt, wird von jedem Maschinenbefehl der Teil übernommen, der die gewünschte Operation und die dazu benötigten Prozessorkomponenten spezifiziert. Dieser Befehlsteil wird Operationscode *(OpCode)* genannt. Das Befehlsregister ist häufig als Block aus mehreren Registern realisiert, da einige Prozessoren Operationscodes besitzen, die sich in der Anzahl der benutzten Wörter unterscheiden. In diesem Fall werden befehlsabhängig ein oder mehrere Register des Blocks belegt.

[2] Genau genommen, liegt hier also keine Steuerung, sondern eine Regelung vor.
[3] Wir unterscheiden streng zwischen der (vollständigen) Befehlsbearbeitung und der Befehlsausführung, mit der nur die Bearbeitung in den Operationswerken gemeint ist.

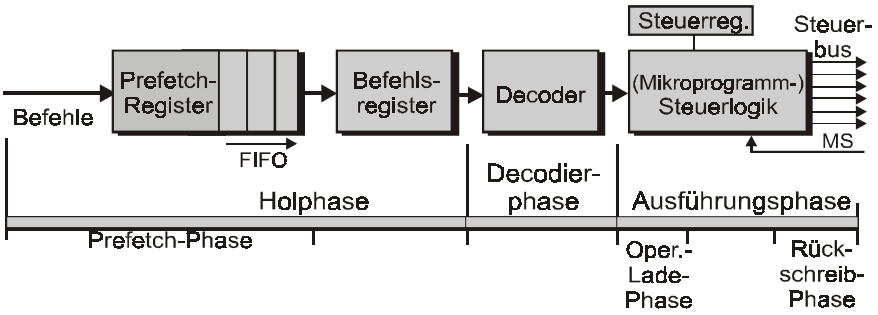

Bild 2.1-2: Phasenunterteilung der Befehlsbearbeitung

- Moderne Mikroprozessoren führen eine mit *Opcode Prefetching* bezeichnete Maßnahme zur Steigerung der Verarbeitungsgeschwindigkeit aus, für die eine eigene *Prefetch*-Phase eingeführt wird: Während der aktuelle Befehl decodiert oder in anderen Komponenten des Prozessors bearbeitet wird, wird schon einer (oder mehrere) der folgenden Befehle aus dem Speicher in den Prefetch-Registerblock geladen. Im Bild 2.1-3 ist im oberen Teil der zeitliche Signalverlauf *(Timing)* ohne Prefetching, im unteren mit Prefetching dargestellt. Deutlich wird, daß in diesem Fall die Bandbreite des Systembusses besser genutzt wird. Nachteilig ist jedoch, daß vorzeitig geladene Befehle wieder aus dem Prefetch-Register(-block) entfernt werden müssen, wenn sich herausstellt, daß der aktuell ausgeführte Befehl zu einer Programmverzweigung führt.

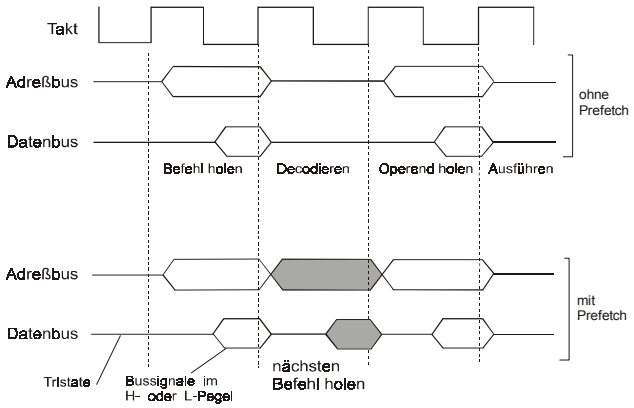

Bild 2.1-3: Zeitlicher Ablauf des Befehls-Prefetching

Im einfachsten Fall besitzt der Prefetch-Registerblock genau einen Eintrag. Viele Prozessoren besitzen einen ganzen Block von Registern, der als Warteschlange *(Opcode Prefetch Queue)* realisiert ist, bei welcher der aktuell zu bearbeitende Befehl am Ende ausgelesen wird, jeder andere Befehl darin um eine Position weiter-

rückt und am Anfang ein neuer Befehl eingetragen wird[4]. Die Befehle werden also immer in der Reihenfolge abgearbeitet, mit der sie in die Schlange eintreten. Deshalb spricht man von einem FIFO-Speicher *(First in, First out)*. (Dieser ist aus Schieberegistern relativ einfach aufzubauen.)

- In der folgenden **Decodierphase** *(Decode Cycle)* wird durch den Befehlsdecod(ier)er (*Instruction Unit* – IU) der Befehl entschlüsselt („interpretiert"), d.h. es wird insbesondere festgestellt, um welche Befehlsgruppe es sich handelt, welche Prozessorkomponente für die Ausführung zuständig ist, ob und welche Operanden benötigt werden. (Häufig wird die Decodierphase noch zur Holphase gezählt.)
- Je nach Implementierung können in der Decodierphase auch bereits die Operanden geladen werden. In diesem Fall ist die **Operanden-Ladephase** *(Operand Fetch Cycle)* nach Bild 2.1-2 eine Teilphase der Decodierphase und nicht der folgenden Ausführungsphase.
- In der **Ausführungsphase** *(Execution Cycle)* wird die eigentliche „Nutzleistung" erbracht, wie sie von den Befehlen verlangt wird. Überwacht und angeleitet durch die Steuerlogik, werden die Befehle durch die vom Decoder bestimmte Prozessorkomponenten „ausgeführt". Dazu generiert die Steuerlogik die erforderlichen Steuersignale, die über den Steuerbus alle Komponenten erreichen. Durch Meldesignale (MS, s. Bild 2.1-2) können die Komponenten dem Steuerwerk Zustandsinformationen liefern und dadurch Einfluß auf die Befehlsausführung nehmen. Durch die Ablage bestimmter Informationen („Programmierung") im Steuerregister kann ebenfalls die grundlegende Arbeitsweise der Steuerlogik beeinflußt werden. Typische Vorgaben sind z.B., ob Unterbrechungsanforderungen zugelassen werden sollen (*Interrupt Enable Flag*, s. Abschnitt 2.2), ob die virtuelle Speicherverwaltung *(Memory Management)* aktiviert ist oder ob der Prozessor im Benutzer- oder Betriebssystemmodus *(User/Supervisor Mode)* arbeitet. Im einfachen Prozessor nach Bild 1.3-6 geschieht die Befehlsausführung durch das Operationswerk (oder die ALU) und das Rechenwerk. Moderne Hochleistungsprozessoren verfügen darüber hinaus über weitere Ausführungseinheiten (*Execution Units*, s. Abschnitt 2.4).
- Zum Ende der Ausführungsphase werden die berechneten Ergebnisse in die Register oder in den Arbeitsspeicher zurückgeschrieben. Zur Erleichterung der Fließbandverarbeitung *(Pipelining)* wird bei modernen Mikroprozessoren die Abspeicherung der Ergebnisse in einer getrennten Phase, der **Rückschreibphase** *(Write-Back Phase),* durchgeführt.

Wie im Abschnitt 1.1 erwähnt, ist die Steuerung moderner Hochleistungs-Prozessoren überwiegend aus fest verdrahteter Logik *(Hard-wired Logic)* realisiert; die Steuerlogik einfacher Mikroprozessoren, insbesondere auch von Mikrocontrollern, besteht zum großen Teil aber (immer noch) aus Mikroprogramm-Steuerwerken.

---

[4] Der *Prefetch*-Zugriff auf den Systembus zum Laden neuer Befehle hatte bei den älteren µP-Typen stets eine geringere Priorität als die zur Ausführung des aktuellen Befehls erforderlichen Buszugriffe. Erst bei den neueren Typen, beginnend mit dem Intel 80486, bekam er eine höhere Priorität. Auf diese Weise wird dafür gesorgt, daß die Warteschlange stets gefüllt ist und der µP somit nicht unbeschäftigt (*idle*) ist.

2.1.1.3 Exkurs: Grundlagen der Mikroprogramm-Steuerwerke

Wir betrachten hier einen einfachen Standardprozessor, dessen Steuerung als Mikroprogramm-Steuerwerke ausgelegt ist. Im Bild 2.1-4 ist sein MPStW skizziert. In einem Festwertspeicher des Steuerwerks (Mikroprogramm-Speicher, *Control Memory/ Store*) liegt für jeden Makrobefehl ein Mikroprogramm vor. (Da die Mikroprogramme vom Benutzer nicht geändert werden können, ist dieser Prozessor somit zwar mikroprogrammiert aber nicht mikroprogrammierbar; vgl. Abschnitt 1.1). Der Operationscode (OpCode) des Makrobefehls, der im Befehlsregister gespeichert ist, wird vom Befehlsdecod(ier)er interpretiert („entschlüsselt"). Als Ergebnis liefert der Decoder dem Mikroprogramm-Steuerwerk die Anfangsadresse eines Mikroprogramms zur Ausführung des anstehenden Befehls. Außerdem stellt der Decoder fest, wie viele Operanden der Befehl benötigt, wo sie zu finden sind und welche Register dazu benötigt werden.

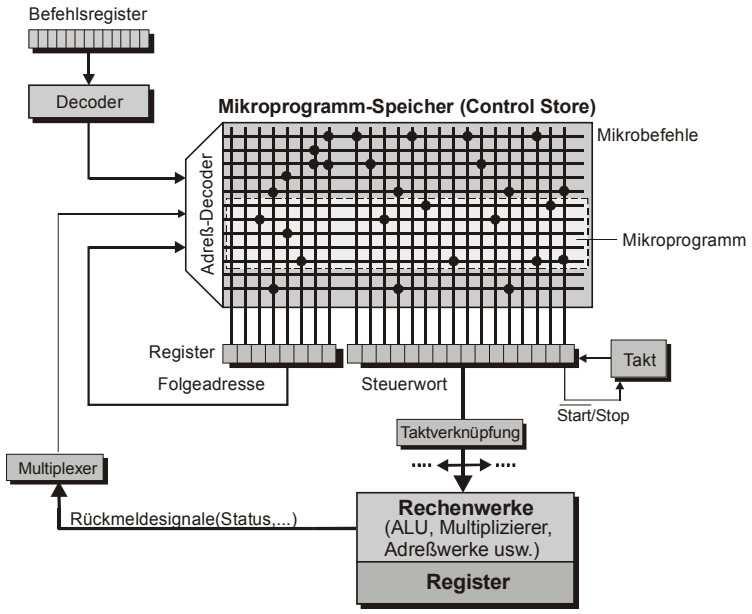

Bild 2.1-4: Aufbau eines einfachen Mikroprogramm-Steuerwerks

Ein Mikroprogramm besteht aus einer Folge von **Mikrobefehlen** (*Micro Instructions, s.* Bild 2.1-5). Die Abfolge dieser Befehle wird durch einen Takt gesteuert, der (z.B.) durch das letzte Bit der Mikrobefehle beeinflußt, d.h. gestartet und gestoppt wird. Den einzelnen Bits eines Mikrobefehls entsprechen **Mikrooperationen**, durch welche die Auswahl- und Freigabesignale *(Select, Enable)* für die benötigten Komponenten bestimmt und die Datenwege im Prozessor und zu den externen Komponenten geschaltet werden. Das Steuerwort im Mikrobefehl kann in einzelne Felder unterteilt werden, deren Bits jeweils bestimmte Komponenten des Prozessors steuern, wie z.B. das Re-

chenwerk (s. Bild 2.1-5). Durch Verknüpfung der Signale mit dem Prozessortakt werden spezifische Steuertakte erzeugt, deren Flanken die Schaltzeitpunkte festlegen. Dadurch wird für die erforderliche zeitliche Abfolge der Komponentensteuerung gesorgt. Die Komponenten ihrerseits liefern Rückmeldesignale, aus denen über einen Multiplexer vom Steuerwort ein Signal ausgewählt und als Teiladresse an den Speicher gelegt werden kann. Auf diese Weise können, abhängig von einem bestimmten Zustand, Verzweigungen im Mikroprogramm durchgeführt werden. Schließlich meldet das Mikroprogramm-Steuerwerk dem Decoder das Ende der Befehlsausführung und fordert dadurch den nächsten Befehl an. Gleichzeitig wird der Takt des MPStWs angehalten.

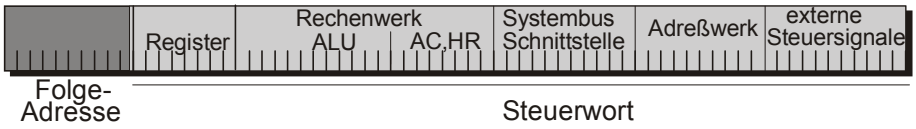

Bild 2.1-5: Prinzipieller Aufbau eines Mikrobefehls

Das letzte Feld des im Bild 2.1-5 dargestellten Mikrobefehls erzeugt die externen Signale, die an den Anschlüssen des Prozessors zur Steuerung der Peripheriebausteine abgegriffen werden müssen. Andere Signale werden von diesen Bausteinen zum Steuerwerk des Prozessors geführt und dort ausgewertet. Dazu gehören typischerweise die im folgenden Unterabschnitt beschriebenen Signalgruppen.

2.1.2 Ein-/Ausgangssignale

2.1.2.1 Zuteilung des Systembusses

Können in einem Mikroprozessor-System mehrere Komponenten aktiv auf den Systembus zugreifen, so benötigt man eine Schaltung zur Anforderung des Systembusses vom µP und zur Gewährung des Zugriffs durch den Prozessor *(Bus Arbitration Control)*. Bei diesen Komponenten kann es sich um einen weiteren (Co-)Prozessor oder aber auch um einen Peripheriebaustein handeln (z.B. um den im Abschnitt II.3.3 beschriebene DMA-Controller – *Direct Memory Access*). Die folgenden drei Signale dienen zur Regelung des Zugriffs auf den Systembus. Die beschriebene Form der Buszuteilung wird als 3-Leitungs-*Handshake* oder Quittungsbetrieb bezeichnet.

- Durch das HOLD-Eingangssignal, auch mit BR oder BREQ *(Bus Request)* bezeichnet, wird dem Prozessor mitgeteilt, daß (wenigstens) eine andere Komponente des Mikroprozessor-Systems den Systembus belegen will. Der Prozessor schließt daraufhin noch die augenblicklich durchgeführte Systembusoperation (Schreib- oder Lesezugriff) ab.
- Sobald der Prozessor den Systembus nicht mehr benötigt, schaltet er alle seine Systembus-Ausgänge in einen hochohmigen Zustand (Tristate, vgl. Unterabschnitt

2.1.2.4). Davon unterrichtet er jede anfordernde Komponente über das Ausgangssignal HOLDA *(Hold Acknowledge),* auch BG oder BGRT *(Bus Grant)* genannt. Während der Freigabe des Systembusses kann der Prozessor gegebenenfalls weitere Befehle in seinem Befehls-Prefetch-Block (s.o.) oder dem internen Cache-Speicher bearbeiten, solange dazu nicht der Systembus benötigt wird.

- Bewerben sich gleichzeitig mehrere Komponenten um das Bussystem, so muß nach einem geeigneten Verfahren durch einen „Schiedsrichter“ *((Bus Hold) Arbiter)* genau eine von ihnen zum neuen *Bus Master* bestimmt werden. Diese kann nun z.B. über das Signal BGA *(Bus Grant Acknowledge)* allen anderen Mitbewerbern mitteilen, daß sie die Buskontrolle übernommen hat. (Im Unterabschnitt 2.6.8 werden Sie mit dem *Independent Request*-Verfahren eines dieser Entscheidungsverfahren kennenlernen.)

2.1.2.2 Unterbrechungsanforderungen

- Durch die Interrupteingänge IRQ (*Interrupt Request;* auch mit INT, INTR bezeichnet) und NMI *(Non-Maskable Interrupt)* kann ein laufendes Programm durch eine Systemkomponente unterbrochen und die Abarbeitung einer „Interruptroutine“ im Prozessor angestoßen werden. (Wegen der großen Bedeutung der Programmunterbrechungen wird in einem eigenen Abschnitt 2.2 ausführlich darauf eingegangen. Dort wird der im Bild 1.3-6 mit Unterbrechungslogik bezeichnete Teil des Steuerwerks beschrieben.)
- Mit Hilfe der oft bidirektional ausgeführten HALT-Leitung – benutzt als Eingang – kann der Prozessor aufgefordert werden, nach der Beendigung der momentan auf dem Systembus laufenden Transaktion zu stoppen, d.h. er stellt jegliche Programmbearbeitung ein – insbesondere auch die interne Bearbeitung seiner Befehlswarteschlange. Er signalisiert den angeschlossenen Komponenten über die gleiche HALT-Leitung, benutzt als Ausgang, oder seine Statussignale (s. Unterabschnitt 2.1.2.3), daß er im Halt-Zustand ist. Dabei schaltet er alle seine Ausgänge, also die Systembus-Ausgänge und alle anderen Steuerausgänge, in den hochohmigen Zustand. Aus diesem Zustand kann er meist nur durch einen Interrupt oder ein Rücksetzen (RESET) „aufgeweckt“ werden. Von dieser Möglichkeit des Abbruchs jeder Programmbearbeitung wird insbesondere dann Gebrauch gemacht, wenn ein Fehler aufgetreten ist, der nicht vom µP selbst behoben werden kann.

2.1.2.3 Weitere Signale

Fehlermeldungen

Durch die Fehlermeldung BERR *(Bus Error)* wird dem Prozessor von bestimmten Komponenten mitgeteilt, daß bei der augenblicklich auf dem Systembus ablaufenden Transaktion ein Fehler festgestellt wurde. Beispiele dafür sind ein Zugriff auf einen

geschützten Speicherbereich, das Ausbleiben eines Quittungssignals von einer Systemkomponente oder aber ein festgestellter Übertragungsfehler.

Statusinformationen

Durch eine Gruppe von **Prozessor-Statussignalen**, die beispielsweise mit FC_n,..,FC_0 *(Function Code)* oder ST_n,..,ST_0 (*Status*) bezeichnet sind, kann der Prozessor den angeschlossenen Komponenten seinen Zustand bzw. den Typ der augenblicklich ausgeführten Operation mitteilen. Dazu gehören z.B., ob der Prozessor:

- ein Benutzerprogramm, eine Betriebssystemroutine oder eine Interruptroutine ausführt,
- im Halt-Zustand ist bzw. auf eine Unterbrechung wartet,
- Daten von einem Coprozessor erwartet oder zu diesem übertragen will,
- eine Lese- oder Schreiboperation ausführt,
- Daten oder OpCodes überträgt,
- auf den Arbeitsspeicher oder einen Peripheriebaustein zugreift.

Kommunikation mit einem Coprozessor

Über einige weitere Signale korrespondiert der Prozessor mit einem eventuell im System vorhandenen Coprozessor. Diese Signale werden im Rahmen dieses Buches nicht weiter behandelt.

Systembus-Steuersignale

Durch diese Signale werden insbesondere die Richtung des Datentransfers auf dem Systembus, das Einfügen von Wartezyklen beim Buszugriff sowie die Art der selektierten Komponente (Arbeitsspeicher oder Peripheriebaustein) festgelegt. Sie werden neben weiteren Signalen im Abschnitt 2.6 ausführlich beschrieben.

2.1.2.4 Exkurs: Open-Collector- und Open-Drain-Eigenschaft

Zum Abschluß dieses Abschnitts soll noch einmal auf die Eingangssignale des Steuerwerks eingegangen werden. Als Beispiel werde das oben beschriebene HOLD-Signal herangezogen, das typischerweise im L-Pegel aktiv ist – also korrekter mit HOLD# bezeichnet wird. In der Regel gibt es in einem komplexen Mikrorechner-System mehr als eine Komponente, die vom µP den Zugriff zum Systembus verlangen kann. Nun ist es nicht ohne weiteres möglich, die HOLD-Ausgänge aller dieser Komponenten einfach zu verbinden und zusammen auf den HOLD-Eingang des Prozessors zu legen, da dadurch ein elektrischer Kurzschluß entstehen würde. Diese direkte Verbindung ist nur bei Gattern möglich, die sogenannte *Open-Collector*-Ausgänge (bzw. *Open-Drain*-Ausgänge) besitzen. Im Bild 2.1-6 ist das Prinzip skizziert, nach dem mehrere *Open-Collector*-Ausgänge mit einem Eingang verbunden werden können.

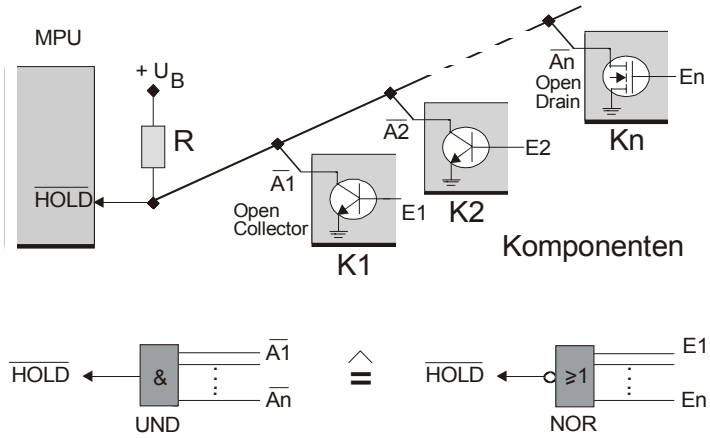

Bild 2.1-6: Anschluß mehrerer Komponenten am *HOLD*-Eingang des Prozessors

Das kreisförmige Symbol in den Komponenten K1 und K2 stellt einen bipolaren Transistor, in der Komponente Kn einen MOS-Transistor *(Metal-Oxid Semiconductor)* dar. Ihre Funktion werden wir im Abschnitt II.2.1 genauer beschreiben. Hier reicht es zu wissen, daß diese Transistoren „leiten", wenn am zugehörigen Eingang Ei ein H-Pegel liegt. In diesem Fall wird der Komponentenausgang Ai gegen Masse kurzgeschlossen.

Liegt hingegen an Ei ein L-Pegel, so „sperrt" der Transistor, der Ausgang ist dann hochohmig gegen Masse. Alle Ausgänge Ai sind über einen gemeinsamen Widerstand R mit der positiven Betriebsspannung +UB verbunden. Dieser sorgt hauptsächlich dafür, daß der HOLD-Eingang auf H-Potential liegt, wenn alle Ausgänge Ai hochohmig gegen Masse geschaltet sind. Um den HOLD-Eingang des Prozessors auf L-Potential herunterzuziehen und dadurch den Systembus anzufordern, reicht es nun stets, daß wenigstens ein Ausgang Ai auf Masse heruntergezogen wird oder – gleichbedeutend damit –, daß wenigstens ein Steuereingang Ei auf H-Pegel liegt.

Wie im Bild angedeutet, stellt daher die direkte Verbindung der *Open-Collector*-Ausgänge eine logische UND-Verknüpfung der negierten Signale Ai bzw. eine NOR-Verknüpfung der Signale Ei dar. In der Regel sind alle Ausgänge von Systembausteinen, die zur Ansteuerung gemeinsam benutzter Prozessoreingänge dienen, als *Open-Collector*- bzw. *Open-Drain*-Ausgänge realisiert.

2.1.3 Das Steuerregister

Mit Hilfe des Steuerregisters *(Control Register)* kann die momentane Arbeitsweise des Prozessors beeinflußt werden. Dazu wird während des Programmlaufs ein spezifisches Steuerwort in dieses Register geschrieben. Die einzelnen Bits des Steuerregisters besitzen eine von Prozessor zu Prozessor verschiedene Funktion. Im folgenden werden nur einige Beispiele für diese Steuerbits angegeben.

- Das *Interrupt (Enable) Bit/Flag* (IE, IF) bestimmt, ob einer Unterbrechungsanforderung am INT-Eingang (s.o.) des Prozessors stattgegeben werden soll oder nicht. Auf dieses Bit wird im Abschnitt 2.2 ausführlich eingegangen. Durch weitere Bits, die z.B. mit IPLi (*Interrupt Privilege Level*, i=0–3) bzw. IOPLi (*Input/Output Privilege Level*, i=0–2) bezeichnet werden, kann festgelegt werden, welche Priorität eine Unterbrechungsanforderung bzw. eine Ein-/Ausgaberoutine momentan besitzen muß, um ausgeführt zu werden.
- Mit dem *User/System Bit* kann zwischen zwei Betriebsarten des Prozessors gewählt werden. In der Betriebsart Benutzermodus *(User Mode)* ist nur eine eingeschränkte Teilmenge der Prozessorbefehle ausführbar. In der Betriebsart Systemmodus *(System Mode)* können dagegen alle Befehle benutzt werden. Diese Betriebsart ist in der Regel für das Betriebssystem reserviert (deshalb auch: *Supervisor Mode*).
- Mit den Bits *Paging Enable* (PG) bzw. *Protection Enable* (PE) können eine bestimmte Art der virtuellen Speicherverwaltung bzw. Mechanismen zum Speicherschutz gegen unerlaubte Zugriffe aktiviert oder deaktiviert werden (vgl. Kapitel 5).
- Das *Trace Bit* (auch *Trap Bit* genannt) erlaubt eine Abarbeitung des Programms im Einzelschritt-Modus *(Single Step Mode),* bei dem nach jeder Ausführung eines Befehls eine Unterbrechungsroutine aufgerufen wird. Diese Betriebsart dient zum Testen von Programmen in der Entwurfsphase und zur Überprüfung fehlerhaft ablaufender Programme *(Debugging).*
- Das *Decimal Bit* veranlaßt die ALU, im Dezimalmodus zu arbeiten, d.h. die Operanden als BCD-Zahlen *(Binary Coded Decimals)* aufzufassen und als solche zu verarbeiten.

In den nun folgenden Fallstudien zu den Steuerwerken realer Mikroprozessoren werden einige weitere, sehr spezielle Steuerbits beschrieben. Leider wird in den Datenblättern der Mikroprozessor-Hersteller das Steuerwerk meist sehr knapp abgehandelt. Deshalb können einige Punkte bei der folgenden Beschreibung zweier Mikroprogramm-Steuerwerke nur sehr pauschal dargestellt werden.

2.1.4 Fallstudie: Das Steuerwerk des Motorola MC68000

In der Literatur konnten wir eine genauere Beschreibung des MPStWs nur für den Prozessor MC68000 der Firma Motorola finden, die glücklicherweise auch Aussagen über die exakte Organisation des Mikroprogrammspeichers macht. Das Steuerwerk ist in Bild 2.1-7 skizziert. (Interne Steuersignale des Steuerwerks sind darin zur Vereinfachung nicht eingezeichnet.) Über den internen Bus gelangen die Operationscodes (OpCodes) in das Befehlsregister. Dort werden sie vom Befehlsdecoder ausgewertet, der daraus eine 10-bit-Startadresse für das Mikroprogramm ermittelt und diese an den Mikrobefehlszähler des MPStWs übergibt. Außerdem kann der Decoder bereits die Steuersignale aktivieren, die während der Ausführungsphase konstant bleiben. Dazu gehören z.B. die Auswahlsignale eines bestimmten Registers oder der ALU.

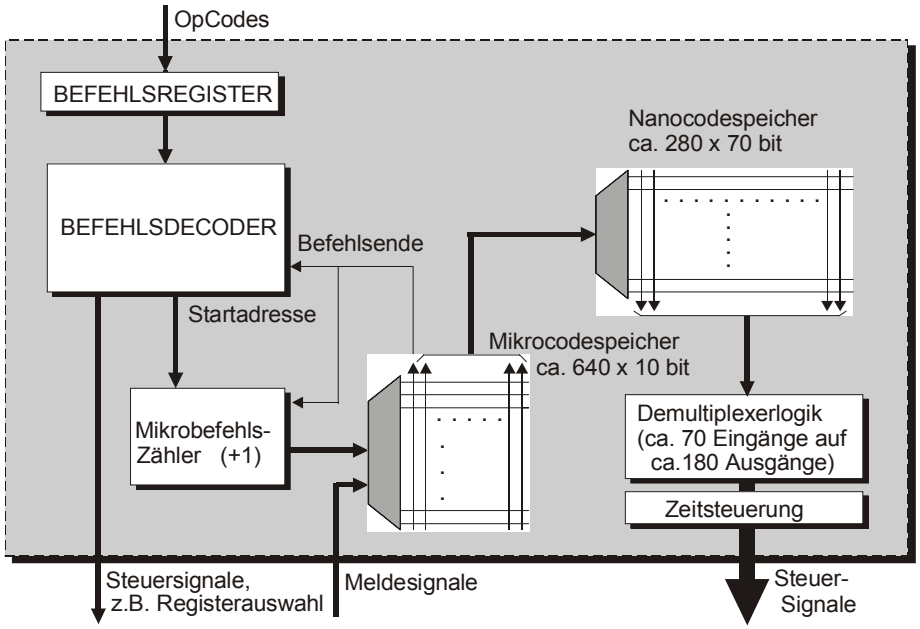

Bild 2.1-7: Das Steuerwerk des MC68000 von Motorola

Alle anderen Steuersignale, die zur zeitgerechten Aktivierung der Prozessorkomponenten oder der anderen Systembausteine dienen, werden dann vom MPStW erzeugt. Ein *OpCode-Prefetch*-Verfahren findet in der oben beschriebenen einfachen Form statt, derart daß während der Decodier- oder Ausführungsphase eines Befehls schon ein weiterer OpCode in das Befehlsregister geladen werden kann.

Das MPStW ist zweistufig mikroprogrammiert, d.h. der Mikrocodespeicher enthält nicht die Steuerwörter selbst, sondern nur deren Adresse im Steuerwortspeicher (Nanocodespeicher). Dadurch wird sehr viel Speicherplatz gespart, denn sich häufig wiederholende Steuerwörter brauchen jeweils nur ein einziges Mal im Speicher abgelegt zu werden. So sind für den gesamten Befehlsvorrat des Prozessors ca. 640 Mikrobefehle im Mikrocodespeicher nötig, die jeweils nur 10 bit lang sind. Eines dieser 10 Bits liefert ein Rückmeldesignal zum Befehlsdecoder und meldet z.B. das Ende eines Mikroprogramms. Der Nanocodespeicher enthält ungefähr 280 Steuerwörter mit jeweils ca. 70 Bits. Insgesamt umfaßt der Mikroprogrammspeicher also ungefähr 26 kbit. Berechnungen haben gezeigt, daß dieser bei einer einstufigen Organisation fast doppelt so groß geworden wäre. Die Komponenten des Mikroprozessors benötigen näherungsweise 200 Steuersignale, die hauptsächlich vom Nanocodespeicher erzeugt werden müssen. Durch die nachfolgende Demultiplexerlogik werden die 70 Signale des Nanocodespeichers selektiv auf ca. 180 Steuerleitungen der Komponenten geschaltet. Es reicht dabei, jedes Bit des Steuerworts über den Demultiplexer mit maximal 2 bis 3 Steuerleitungen zu verbinden.

Durch die mit Zeitsteuerung bezeichnete Komponente werden die Signale des Steuerworts mit dem Systemtakt so verknüpft, daß sie die einzelnen Komponenten zeitgerecht ansteuern.

2.1.5 Fallstudie: Das Steuerwerk des Intel 80486

Bild 2.1-8 zeigt grob den Aufbau des Steuerwerks beim Intel 80486.

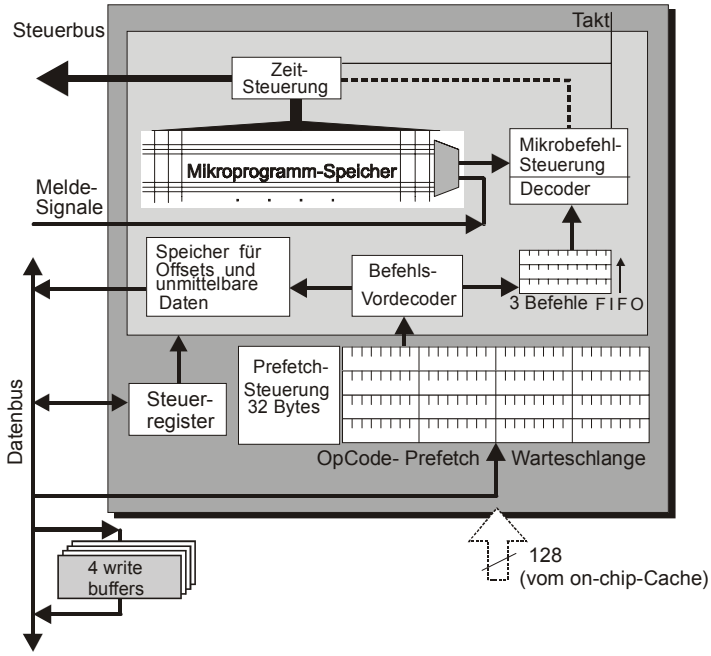

Bild 2.1-8: Das Steuerwerk des 80486 von Intel

Der 80486 besitzt einen Befehls-Registerblock, der in Form eines FIFO-Speichers mit 32 byte ausgelegt ist. Die *Prefetch*-Steuerung sorgt dafür, daß dieser Speicher immer so weit wie möglich gefüllt ist. Das Füllen geschieht dabei mit großer Wahrscheinlichkeit aus dem chipinternen Cache über einen 128 bit breiten Datenpfad, über den der FIFO-Speicher in nur zwei Taktzyklen vollständig neu geladen werden kann. Müssen jedoch Daten oder Befehle aus dem Arbeitsspeicher geholt werden, so haben diese Zugriffe höhere Priorität als eventuelle Schreibzugriffe des Prozessors auf den externen Datenbus. Die auszugebenden Daten (und ihre Zieladressen) werden in diesem Fall in den vier Schreibpuffern *(Write Buffer)* zwischengelagert. Liegt in einem der Schreibpuffer eine neuere Version des zu ladenden Datums, so wird dies von der Steuerung erkannt und das Datum nicht aus dem Speicher, sondern aus dem Puffer geladen.

Es werden ganze Befehle, bestehend aus OpCodes und Operanden in die Warteschlange übertragen. Dabei verhindert die FIFO-Steuerung, daß Befehle aus Speicherbereichen entnommen werden, die durch die Speicherverwaltung nicht dem aktuell ausgeführten Programm zugeteilt sind. Aus der Warteschlange gelangt jeder Befehl zum Befehls-Vordecoder *(Instruction Predecode)*. Hier werden einerseits die unmittelbar im Befehl angegebenen Operanden („unmittelbare Daten“) oder ihre Adreßdistanzen *(Offset)* zu bestimmten Basisadressen abgezweigt und in einem separaten Speicher abgelegt. Andererseits wird der OpCode so weit vorverarbeitet, daß dem nachfolgenden Decoder alle wesentlichen Informationen über die Art der Operation, die benutzten Register usw. zur Verfügung gestellt werden. Die vordecodierten Befehle werden in eine weitere FIFO-Warteschlange eingefügt, die maximal drei Befehle aufnehmen kann. Nach jeder Befehlsausführung entnimmt der Decoder den nächsten vordecodierten Befehl aus dieser Warteschlange und ermittelt für die Mikrobefehlssteuerung die Startadresse des erforderlichen Mikroprogramms. Die Signale des Mikroprogrammspeichers werden in der Komponente ‚Zeitsteuerung’ mit Hilfe des Takts zu den Steuersignalen der Komponenten verknüpft. Die gestrichelte Linie soll auch hier andeuten, daß einige der Steuersignale auch direkt vom Decoder geliefert werden können.

2.2 Ausnahmebehandlung

Während der normalen Bearbeitung eines Programms kann es in einem Mikroprozessor-System immer wieder zu Ausnahmesituationen *(Exceptions)* kommen, die vom Prozessor eine vorübergehende Unterbrechung oder aber einen endgültigen Abbruch des Programms, also in jedem Fall eine Änderung des normalen Programmablaufs verlangen. Diese Ausnahmesituationen können einerseits durch das aktuell ausgeführte Programm selbst herbeigeführt werden, mit diesem also in einem logischen Zusammenhang stehen, andererseits aber auch ohne jegliche Beziehung zum Programm sein. Ursachen sind im ersten Fall das Vorliegen von Fehlern im System oder im Prozessor selbst, im zweiten Fall der Wunsch externer Systemkomponenten, vom Prozessor „bedient“ zu werden. Durch einen Abbruch reagiert das System stets auf schwere Fehler, die eine Fortsetzung der Programmausführung verhindern, also z.B. auf Hardwarefehler oder falsche Werte in einer Systemtabelle.

Im folgenden werden nun die erwähnten Möglichkeiten im einzelnen dargestellt. Zu jeder wird beschrieben, welche Maßnahmen ergriffen werden, um die Ausnahmesituation zu beenden. Man spricht allgemein von **Ausnahmebehandlung** *(Exception Handling)* oder Ausnahmebearbeitung *(Exception Processing)*. Die Ausnahmebehandlung wird von einer Komponente des Steuerwerks ausgeführt bzw. veranlaßt, die wir mit dem Begriff Unterbrechungslogik *(Interrupt System)* bezeichnen (vgl. Bild 1.3-6).

Einige Ausnahmesituationen werden nur durch die Hardware des Prozessors behoben. Dazu zählen beispielsweise das unten beschriebene Rücksetzen des Prozessors oder das Wiederholen einer Systembusoperation im Fehlerfall. Die meisten Ausnah-

mesituationen – insbesondere die, bei denen das Programm lediglich kurzzeitig unterbrochen, nicht aber abgebrochen werden soll – werden durch die Ausführung spezieller Routinen bereinigt. Wir nennen sie Ausnahmebehandlungsroutinen, kurz **Ausnahmeroutinen**. Diese Routinen müssen i.d.R. – wie alle anderen Programme auch – vom Betriebssystem oder dem Anwender selbst zur Verfügung gestellt werden. Die Auswahl und Aktivierung einer bestimmten Routine muß durch die bereits erwähnte Hardwarekomponente im Steuerwerk des Prozessors, die Unterbrechungslogik, wirksam unterstützt werden. Insbesondere müssen alle Aktivitäten aus Zeitgründen möglichst parallel zu den Operationen der übrigen Prozessorkomponenten ablaufen.

2.2.1 Ausnahmeroutinen

Der Aufbau einer Ausnahmeroutine ähnelt stark dem Aufbau eines Unterprogramms. In einem Unterprogramm bringt man gewöhnlich Programmteile unter, die häufiger benutzt werden, aber nur einmal im Speicher abgelegt werden sollen. Ein Unterprogramm kann nur an zur Programmierzeit festgelegten Stellen im Hauptprogramm (oder einem anderen Unterprogramm) durch spezielle Befehle *(Call Subroutine*, *Jump to Subroutine)* aufgerufen werden (vgl. Unterabschnitt 3.2.3). In diesen Befehlen muß die Startadresse des Unterprogramms angegeben werden. Durch das Steuerwerk wird zunächst der aktuelle Inhalt des Programmzählers und bei einigen Prozessortypen auch der des Statusregisters auf dem Stack[1] abgelegt. Danach wird die im Befehl angegebene Startadresse in den Programmzähler geladen und der Programmablauf an der dadurch bezeichneten Speicherstelle fortgesetzt. Die Beendigung des Unterprogramms geschieht durch einen besonderen Befehl *(Return from Subroutine),* durch den der Programmzähler und gegebenenfalls auch das Statusregister mit ihren alten Werten aus dem Stack restauriert und das Programm an der Unterbrechungsstelle fortgesetzt werden.

Demgegenüber tritt der Aufruf einer Ausnahmeroutine in einem Programm nicht explizit in Erscheinung. Stattdessen wird er durch ein externes Signal oder das Auftreten einer Fehlerbedingung im Prozessor veranlaßt. Dies kann prinzipiell an beliebigen Stellen des Programms, z.B. nach jeder Befehlsausführung geschehen. Dabei wird die Startadresse der Ausnahmeroutine durch das Steuerwerk vorgegebenen Speicherzellen entnommen. Wie bei einem Unterprogrammaufruf wird zunächst der aktuelle Inhalt des Programmzählers auf den Stack gerettet und dann an der durch die Startadresse bezeichneten Stelle im Speicher fortgefahren. Um möglichst rasch auf Unterbrechungsanforderungen reagieren zu können, wird manchmal auf die Sicherung des Statusregister-Inhalts verzichtet. Die Beendigung der Ausnahmeroutine geschieht mit Hilfe eines speziellen Befehls *(Return from Interrupt/Exception),* durch den der Programmzähler mit seinem alten, auf dem Stack geretteten Inhalt restauriert und die Programmausführung an der Unterbrechungsstelle fortgesetzt wird.

---

[1] Mit Stack bezeichnet man einen besonderen Bereich im Speicher, der zur kurzfristigen Ablage wichtiger Daten benutzt wird. Er wird im Unterabschnitt 2.5.2 genauer beschrieben.

Genauer betrachtet, kann man bei der Behandlung von vorübergehenden Programmunterbrechungen die folgenden Schritte unterscheiden:

- Feststellen, ob eine interne oder externe Unterbrechungsanforderung vorliegt; bei externen Anforderungen weiterhin feststellen, ob durch das *Interrupt Enable Bit* (IE, s. Unterabschnitt 2.2.2) im Steuerregister eine Unterbrechung zugelassen ist[2];
- Zuendeführen der augenblicklich bearbeiteten Operation, sofern dies sinnvoll ist; Dabei kann es sich je nach Situation um einen vollständigen Maschinenbefehl oder aber nur um einen Systembuszugriff handeln;
- Übertragen des Prozessorzustands, also des Inhalts von Programmzähler, Statusregister und eventuell weiterer Register in den Stack;
- Feststellen der Quelle der Unterbrechungsanforderung;
- Informieren der Systemkomponenten über spezielle Leitungen (*Interrupt Acknowledge* – INTA) oder die Prozessor-Statussignale, daß eine Unterbrechungsanforderung akzeptiert wurde;
- Deaktivieren bestimmter anderer momentaner Unterbrechungsanforderungen;
- Ermitteln der Startadresse der Ausnahmeroutine für die festgestellte Quelle der Unterbrechungsanforderung und Laden dieser Adresse in den Programmzähler;
- Ausführen der Ausnahmeroutine;
- Beenden der Ausnahmeroutine durch einen speziellen Befehl (*Return from Interrupt* – RTI, auch: *IRET*);
- Restaurieren der anfänglich gesicherten Registerinhalte, Reaktivierung der zeitweise gesperrten Unterbrechungsanforderungen und Fortsetzung des Programms an der Unterbrechungsstelle, also mit dem unmittelbar auf die Unterbrechungsstelle folgenden Maschinenbefehl.

Bis auf die (grau unterlegten) Schritte zur Ausführung und Beendigung der Ausnahmeroutine werden alle diese Schritte durch die Hardware der Interruptlogik – und das so weit wie möglich – parallel ausgeführt. Wie die Startadresse der Ausnahmeroutine genau ermittelt wird, wird erst im Anschluß an die folgende Beschreibung der verschiedenen Ausnahmesituationen gezeigt. Außerdem wird zunächst vorausgesetzt, daß zu jedem Zeitpunkt höchstens eine Unterbrechungsanforderung vorliegt. Erst zum Schluß dieses Abschnitts wird gezeigt, wie mehrere gleichzeitig vorliegende Anforderungen bearbeitet werden.

Für die Ausnahmesituationen können sowohl prozessorinterne als auch prozessorexterne Ursachen vorliegen. Prozessorexterne Ursachen treten in der Regel asynchron zum Programmablauf auf und werden durch die Hardware des Systems erzeugt. Hingegen werden fast alle Ausnahmesituationen mit prozessorinternen Ursachen durch das Programm selbst an fest vorgegebenen Stellen ausgelöst. Sie treten also synchron zum Programmablauf auf.

[2] Diese Abfrage entfällt bei den unten beschriebenen „nicht maskierbaren“ Unterbrechungen!

2.2.2 Prozessorexterne Ursachen für Ausnahmesituationen

2.2.2.1 Rücksetzsignale und Fehlermeldungen

RESET

Die einfachste und „wirksamste“ Ausnahmesituation liegt vor, wenn der Prozessor in einen definierten Anfangszustand zurückgesetzt wird („Initialisierung“). Dies geschieht durch ein Signal am RESET-Eingang des Steuerwerks, wie es im Unterabschnitt 2.1.1 beschrieben wurde. Dort wurden bereits die Möglichkeiten erwähnt, das Zurücksetzen automatisch beim Einschalten der Betriebsspannung *(POR)* oder durch eine Taste durchzuführen. In beiden Fällen liegt gewöhnlich kein Bezug zu einem laufenden Programm vor.

Andererseits werden in Mikroprozessor-Systemen mit hohen Verfügbarkeitsanforderungen häufig Überwachungsschaltungen *(Watch Dogs)* eingesetzt, die immer dann den Prozessor über den RESET-Eingang neu initialisieren, wenn dieser nicht innerhalb einer maximalen Zeitschranke durch eine genau definierte Aktion gezeigt hat, daß er noch korrekt arbeitet. Durch diese Schaltungen wird also insbesondere erkannt, wenn der Prozessor in einer Endlosschleife festhängt oder – durch eine Störung verursacht – den normalen Programmbereich verlassen hat. In diesem Fall besteht also ein Zusammenhang zwischen dem auszuführenden Programm und dem Rücksetzen des Prozessors. (Eine einfache Realisierung eines *Watch Dogs* ist ein Monoflop, dessen Ausgang am RESET-Eingang anliegt und das in regelmäßigen Abständen durch ein Ausgangssignal des Prozessors ‚nachgetriggert‘ werden muß.)

Sobald der RESET-Eingang aktiviert wird, bricht der Prozessor, wie bereits in Unterabschnitt 2.1.1 beschrieben, alle augenblicklich ausgeführten Operationen und Busaktivitäten ab. Danach setzt er (z.B. durch ein Mikroprogramm) alle bzw. einige Register auf vordefinierte Werte (häufig den Wert 0). Anschließend holt er aus einer vorbestimmten Stelle im (Festwert-)Speicher die Startadresse für das nach dem Rücksetzen zuerst auszuführende Programm. Diese wird in den Programmzähler geladen. Bei diesem Programm handelt es sich meist um eine Betriebssystemroutine, die nun alle Hardware- und Software-Systemkomponenten in die erforderlichen Anfangszustände versetzt. Abschließend wird der Teil des Steuerwerks aktiviert, der für die sequentielle Ausführung der Maschinenbefehle sorgt.

HALT, HOLD

Im Abschnitt 2.1 wurde gezeigt, daß einige Mikroprozessoren spezielle Eingänge (HALT, HOLD) besitzen, über die sie in einen Haltezustand versetzt werden können. Das bedeutet, daß der Prozessor nach Abschluß des laufenden Zugriffs auf den Systembus seine Programmbearbeitung unterbricht. Während dieser Zeit werden die Systembus-Ausgänge des Prozessors in den hochohmigen Zustand *(Tristate, Three State)* versetzt. Die beiden Signale HALT bzw. HOLD unterscheiden sich darin, wie lange der Haltezustand des Prozessors andauert: Im ersten Fall dauert der Zustand meist solange, bis er durch ein Rücksetzen des µPs oder eine Unterbrechungsanforde-

rung beendet wird. Im zweiten Fall kann der Prozessor intern vorliegende Befehle und Daten weiter bearbeiten und wird nur dann und solange angehalten, wenn das HOLD-Signal aktiviert ist und der Prozessor auf den Bus zugreifen will[3]. Während dieser Zeit kann eine andere Systemkomponente den Systembus benutzen. Im Abschnitt II.3.3 wird das am Beispiel des direkten Speicherzugriffs (*Direct Memory Access* – DMA), einer Methode zum schnellen Datentransfer zwischen Speicher und Peripheriegerät, beschrieben.

ERROR

Viele Prozessoren[4] besitzen einen Eingang, z.B. BERR *(Bus Error)* genannt, der die Reaktion auf verschiedene prozessorexterne Fehler des Systems erlaubt, die mit dem augenblicklich durchgeführten Zugriff auf den Systembus im Zusammenhang stehen (s. Abschnitt 2.1). Dazu gehören insbesondere Paritätsfehler auf dem Datenbus, Zugriffe auf geschützte Datenbereiche oder das Ausbleiben von Quittungssignalen bestimmter Komponenten. Das Vorliegen einer solchen Fehlersituation muß durch eine spezielle externe Schaltung festgestellt und in ein BERR-Signal umgesetzt werden. Solange dieses Signal aktiv ist, geht der Prozessor in den oben beschriebenen Haltezustand. Sobald das Signal zurückgesetzt wird, wird z.B. in Abhängigkeit vom aktuellen Zustand des HALT-Eingangs eine von zwei Ausnahmebehandlungen durchgeführt:

- Die Operation, die zum Fehler führte, wird wiederholt;
- Es wird eine spezielle Routine zur Fehlerbehandlung aufgerufen.

Weitere Beispiele für spezielle Error-Signale:

- Prozessoren der 32X32-Familie von National Semiconductor besaßen einen speziellen Eingang BRT *(Bus Retry),* über den sie zur Wiederholung des letzten Buszyklus aufgefordert werden konnten.
- Die älteren Intel-Prozessoren (80286, 80386) verfügten über einen Eingang ERROR, über den angeschlossene Coprozessoren das Auftreten eines Fehlers melden konnten (beim 80486: FERR#). Während der Prozessor dem Coprozessor Befehle übermittelte oder auf dessen Ergebnisse wartete, führte ein Signal am ERROR-Eingang zum Aufrufen einer speziellen Behandlungsroutine.

2.2.2.2 Interrupts

Durch periphere Geräte oder Systemsteuerbausteine können – über spezielle Eingänge – asynchron zum Programmablauf Unterbrechungssignale *((Hardware*) *Interrupts)* an den Prozessor abgegeben werden. Gründe dafür können zum Beispiel das Vorliegen eines Datums in einem Eingabegerät oder eines bestimmten Zustands (‚Operation ausgeführt', Fehler usw.) sein. Einer Interruptanforderung wird vom Prozessor frühestens nach der vollständigen Beendigung des in Bearbeitung befindlichen Befehls stattgege-

[3] Einfache Prozessoren ohne internen Programmspeicher werden sofort angehalten.
[4] wie z.B. diejenigen der Familie MC680X0 der Firma Motorola.

ben. Die vom Prozessor als Reaktion gestarteten Ausnahmeroutinen werden **Interruptroutinen** genannt. Die meisten Mikroprozessoren können über die im Unterabschnitt 2.1.2 beschriebenen Statusinformationen den Systemkomponenten mitteilen, daß sie im Augenblick eine Interruptroutine durchführen. Dazu muß der Zustand der Prozessor-Statussignale (z.B. FC_2,..,FC_0 – *Function Code*) von den Komponenten decodiert werden. Diese Information ist insbesondere für diejenige Komponente wichtig, welche die Unterbrechung angefordert hat.

Das oben beschriebene Retten des Prozessorzustands vor der Ausführung eines Interrupts dauert bei zeitkritischen Anwendungen oft unvertretbar lange. Auch bei modernen µPs liegt die Zeit im Bereich einiger Mikrosekunden, z.B. 20 µs. Deshalb bieten einige Prozessoren spezielle Befehle an, mit denen sie in einen Zustand versetzt werden können, in dem sie auf den nächsten Interrupt warten. Diese Befehle, meist mit WAI, WAIT *(Wait for Interrupt)* oder SYNC *(Synchronize)* bezeichnet, sorgen teilweise selbst für das vorzeitige Retten des Prozessorzustands im Stack. Bei anderen Prozessortypen muß der Programmierer dies durch Speicherbefehle explizit vornehmen. Nach dem Auftreten des Interrupts muß dann nur noch der Inhalt des Programmzählers ausgetauscht werden. Oft kann der Wartezustand nicht nur durch einen Interrupt verlassen werden, sondern es besteht die Möglichkeit, durch einen weiteren Prozessoreingang oder durch ein kurzes Triggersignal am Interrupteingang das Programm mit dem auf WAIT folgenden Befehl fortzusetzen, ohne eine Interruptroutine durchzuführen. (Natürlich darf in dieser Realisierung der Prozessorzustand nicht durch den WAIT-Befehl vorher auf den Stack gerettet werden.)

Interrupts können maskierbar oder nicht maskierbar sein. Für beide Typen stehen (meist) getrennte Eingänge zur Verfügung. Der Prozessoreingang für maskierbare Interrupts wird häufig mit INT, INTR bzw. IRQ *(Interrupt Request),* der für nicht maskierbare fast einheitlich mit NMI *(Non-maskable Interrupt)* bezeichnet.

- **Nicht maskierbare Interrupts** *(NMI)* werden nach Abschluß des gerade ausgeführten Befehls unbedingt durchgeführt. Diese Tatsache zeigt schon, daß diese Interrupts nur für wirklich wichtige Ausnahmesituationen reserviert sein sollten. Dazu gehören insbesondere Fehler, welche die Funktionsfähigkeit des Systems ernsthaft gefährden, wie z.B. der Zusammenbruch der Betriebsspannung. In diesem Fall kann die Interruptroutine die Aufgabe haben, die aktuellen Registerinhalte in einen (permanenten) Speicher zu retten, eine Not-Spannungsversorgung einzuschalten oder einen Alarm auszulösen. Während der Behandlung einer NMI-Routine wird zunächst durch die Hardware kein weiterer (maskierbarer oder nicht maskierbarer) Interrupt zugelassen. Einige Prozessoren speichern jedoch diese zwischenzeitlich eintreffenden Anforderungen und bearbeiten sie unmittelbar nach Abschluß der aktuellen Interruptroutine. Diese Anforderungen können aber durch den Programmierer auch während der NMI-Bearbeitung gezielt zugelassen werden.
- **Maskierbare Interrupts** (IRQ) werden nur dann ausgeführt, wenn im Steuerregister des Prozessors das spezielle IE-Bit *(Interrupt Enable Flag)* gesetzt ist (vgl. Unterabschnitt 2.1.3). Beim Initialisieren des Systems (RESET) wird dieses Bit in der Regel zurückgesetzt. Es kann danach vom Entwickler eines Programms aufga-

bengemäß gesetzt bzw. zurückgesetzt werden. (Die Zuordnung der Zustände ‚gesetzt' bzw. ‚zurückgesetzt' zu den logischen Werten ‚0' bzw. ‚1' hängt nur vom Typ des Prozessors ab.) Sobald die Bearbeitung einer maskierbaren Unterbrechungsanforderung beginnt, wird in der Regel das IE-Bit automatisch zurückgesetzt, um weitere maskierbare Interrupts während der aktuellen Ausnahmebehandlung zu verhindern. (NMIs werden natürlich stets akzeptiert.) Jedoch kann der Benutzer durch gezieltes Setzen des Bits selbst bestimmen, in welchem Programmbereich er eine weitere Unterbrechung zulassen will, und dadurch eine Verschachtelung der Unterbrechungsbearbeitungen erreichen *(Interrupt Nesting)*. Zum Setzen bzw. Rücksetzen des IE-Bits werden ihm beispielsweise die Befehle SEI *(Set Interrupt Flag)* bzw. CLI *(Clear Interrupt Flag)* zur Verfügung gestellt (vgl. Unterabschnitt 3.2.3). Beim Verlassen der Interruptroutine wird das IE-Bit wieder automatisch gesetzt und ermöglicht so die Ausführung der nächsten, eventuell schon anstehenden Unterbrechungsanforderungen.

2.2.3 Prozessorinterne Ursachen für Ausnahmesituationen

Wie bereits gesagt, treten diese Ursachen fast immer synchron zum Programmablauf auf. Ausnahmen liegen z.B. bei den Prozessoren vor, die auf dem Chip selbst interruptfähige Komponenten besitzen, wie z.B. serielle oder parallele Schnittstellen, Zeitgeber usw. (s. Kapitel II.3). Für diese Interrupts gelten die oben angestellten Betrachtungen, so daß sie hier nicht wiederholt werden müssen. Die programmsynchronen, internen Unterbrechungen werden durch Software-Interrupts und *Traps* hervorgerufen.

2.2.3.1 Software-Interrupts

Software-Interrupts werden durch besondere Befehle aufgerufen. Diese Befehle tragen oft die Bezeichnungen SWI n *(Software Interrupt)* oder INT n, wobei die Nummer n die Auswahl unter einer vorgegebenen Menge von Ausnahmeroutinen erlaubt. In ihrer Wirkung entsprechen Software-Interrupts dem Aufruf eines Unterprogramms, dessen Startadresse jedoch an einer bestimmten Speicherstelle festgelegt ist und deshalb nicht im Befehl angegeben werden muß. Gegenüber einem Unterprogrammaufruf besitzen sie daher die Vorteile, daß sie mit weniger Speicherzellen für den Maschinencode auskommen (1- oder 2-Byte-Befehle) und in der Regel den gesamten Prozessorzustand automatisch im Stack (s. Unterabschnitt 2.5.2) ablegen. In Betriebssystemen für Mikrorechner werden sie häufig benutzt, um die peripheren Komponenten des Systems anzusprechen. Die von diesen Komponenten benötigten oder ermittelten Parameter werden als Registerinhalte im Stack übergeben.

Software-Interrupts bieten darüber hinaus die Möglichkeit, die Ausnahmeroutinen der Hardware-Interrupts zu Testzwecken ohne externe Stimulierung durchzuführen. Da aber dieser Aufruf synchron zum Programm geschieht, können natürlich nicht alle Situationen getestet werden, die beim entsprechenden Hardwareaufruf vorliegen können.

2.2.3.2 Traps und Faults

Traps („Fallen“) sind Ausnahmesituationen, in die der Prozessor nach dem Auftreten bestimmter Ereignisse kommt, die synchron zum Programmablauf auftreten. Diese Ereignisse stehen in direkter Beziehung zu den aktuell ausgeführten Befehlen bzw. Operationen. Sie werden deshalb auch *Instruction Exceptions* genannt. Traps haben in der Mehrzahl prozessorinterne Ursachen; sie können aber auch durch externe Ereignisse hervorgerufen werden, die dem Steuerwerk durch spezielle Signale mitgeteilt werden. Bei ihnen kann es sich um einen Fehler oder einen anderen „anomalen“ Prozessorzustand handeln. Sie können einerseits vor, andererseits aber auch nach Ausführung einer Operation eintreten. Ereignisse, die vor der Ausführung auftreten, werden häufig auch *Faults* (Fehler) genannt[5]. In Abhängigkeit von der Art des Ereignisses wird eine spezielle Ausnahmeroutine des Betriebssystems ausgeführt.

- *Traps* (im engeren Sinne) treten nach der Ausführung eines Befehls auf und führen in der Regel zum Abbruch *(Abort)* dieses Befehls, ohne daß eine Wiederholung versucht wird. In diesem Fall zeigt die Rücksprungadresse aus der Ausnahmeroutine auf den ersten Befehl nach dem Befehl, der die Ausnahmesituation hervorgerufen hat.
- Bei einem *Fault* wird in der Regel die Operation wiederholt, die zur Ausnahmesituation führte. Dazu muß zunächst durch die Ausnahmeroutine die Bedingung beseitigt werden, die zum Auslösen des Faults führte. Die Rücksprungadresse aus der Ausnahmeroutine zeigt in diesem Fall auf den Befehl, der die Ausnahmesituation hervorgerufen hat. Ein wichtiges Beispiel dafür ist ein sog. Seitenfehler[6] *(Page Fault),* d.h. ein Zugriff auf eine bestimmte Speicherseite, die noch nicht im Hauptspeicher vorliegt. Durch die Ausnahmeroutine des Betriebssystems wird die gewünschte Seite vom Peripheriespeicher in den Arbeitsspeicher geladen.

Moderne Hochleistungsprozessoren besitzen wenigstens zwei verschiedene Arbeitswiesen: den Benutzermodus und den geschützten, privilegierten Betriebssystemmodus. Bei diesen Prozessoren werden Traps häufig auch als Systemaufrufe (*Supervisor Call* – SVC) bezeichnet, da sie einen Übergang vom Benutzer- in den Betriebssystemmodus bewirken. Typische Ursachen für Traps sind:

- nach arithmetischen Operationen
  - Division durch 0 *(Division by Zero)*: Es wurde versucht, durch Null zu dividieren, oder der Quotient ‚paßt‘ nicht in das Ergebnisregister.
  - Überlauf *(Overflow)*: Das Ergebnis einer arithmetischen Operation verläßt den darstellbaren Zahlenbereich.
  - Bereichsüberschreitungen in Feldern *(Array Bound Check)*: Beim Zugriff auf ein Feldelement wurde ein Index berechnet, der die Feldgrenzen über- bzw. unterschreitet.

---

[5] Diese Nomenklatur findet man z.B. in der Intel-Literatur.
[6] Der Begriff „Fehler“ bezeichnet hier keinen Defekt, sondern er besitzt seine ursprüngliche Bedeutung: Es fehlt eine Seite im Speicher. Diese Fehler werden im Abschnitt 5.8 behandelt.

- ungültiger Befehlscode *(Invalid OpCode)*: Das Befehlsregister des Steuerwerks enthält eine Bitkombination, die keinem gültigen Befehl entspricht.
- Aufruf des Betriebssystems *(Supervisor Call Trap)*: Aus einem Benutzerprogramm wurde eine Anforderung an das im privilegierten Modus arbeitende Betriebssystem gestellt.
- illegaler Operationsaufruf *(Illegal Operation Trap, Privilege Violation)*: Aufruf eines privilegierten Befehls in einem Benutzerprogramm.
- unerlaubter Speicherzugriff *(Access Violation)*, z.B.:
  - Überschreiten von Speicherbereichsgrenzen,
  - Verletzung von Zugriffsrechten auf bestimmte Speicherbereiche[7],
  - Lesen eines Worts von einer ungeraden Speicheradresse (nur bei Prozessoren, die den Zugriff auf ein 16/32-bit-Wort nur an geraden Speicheradressen erlauben).
- Stack-Überlauf *(Stack Overflow)*: Der Wert des Stackregisters (s. Unterabschnitt 2.5.2) unterschreitet einen minimalen Wert; der Zugriff auf den Stack zerstört dadurch Daten im Speicher.
- Einzelschritt-Unterbrechung *(Single Step Interrupt, Trace)*: Wenn das *Trace Flag* im Steuerregister (s. Unterabschnitt 2.1.3) gesetzt ist, wird nach jeder Ausführung eines Befehls das Programm unterbrochen und eine vorgegebene Routine aufgerufen; das *Trace Flag* im Statusregister wird während der Abarbeitung dieser Routine zurückgesetzt und erst beim Verlassen der Routine durch den Befehl RTI wieder gesetzt. Die Fortsetzung des Programms durch ein spezielles Signal, z.B. einem Tastendruck, erlaubt die Ausführung des Programms in kontrollierbaren Einzelschritten, bei der in der Ausnahmeroutine die Registerinhalte des Prozessors dargestellt und u.U. modifiziert werden können.

2.2.4 Ermittlung der Startadresse einer Ausnahmeroutine

8-bit-Prozessoren bieten meist nur eine eingeschränkte Ausnahmebehandlung. Insbesondere unterstützen sie keine Traps. Die Anzahl der Software-Interrupts ist sehr begrenzt (z.B. auf drei). Aus diesem Grund kommen diese Prozessoren mit einer kleinen Tabelle im Speicher aus, welche die Startadressen der Ausnahmeroutinen enthält. Diese Startadressen werden üblicherweise als **Vektoren** *(Exception Vectors)* bezeichnet. Die Vektortabelle ist meist in einem Festwertspeicher des Mikrorechner-Systems unter den höchstwertigen Adressen abgelegt. Beim Auftreten einer bestimmten Ausnahmesituation lädt das Steuerwerk des Prozessors die Startadresse der zugehörenden Ausnahmeroutine aus der ihr zugeordneten Speicherzelle in den Programmzähler.

[7] Die virtuelle Speicherverwaltung kann einem Programm Speicherbereiche zuordnen, auf die es z.B. nur exklusiv, nur lesend, nur schreibend oder ‚ausführend' zugreifen darf, s. Kapitel 5.

Tabelle 2.2-1 zeigt als Beispiel die Vektortabelle des Motorola MC6809. Dieser Prozessor besitzt einen 8-bit-Datenbus und einen 16-bit-Adreßbus, so daß jeder Vektor byteweise in zwei aufeinanderfolgenden Speicherzellen abgelegt werden muß[8]. (Üblicherweise werden diese Bytes mit H-Byte und L-Byte bezeichnet; H=*high*, L=*low*.)

Tabelle 2.2-1: Die Vektortabelle des 6809 von Motorola

Speicheradressen		**Ausnahmesituation**	
H-Byte[9]	L-Byte		
$FFFE	$FFFF	RESET	Rücksetzen
$FFFC	$FFFD	NMI	nicht maskierbarer Interrupt
$FFFA	$FFFB	SWI1	Software-Interrupt 1
$FFF8	$FFF9	IRQ	maskierbarer Interrupt
$FFF6	$FFF7	FIRQ	schneller maskierbarer Interrupt
$FFF4	$FFF5	SWI2	Software-Interrupt 2
$FFF2	$FFF3	SWI3	Software-Interrupt 3
$FFF0	$FFF1		(reserviert)

Wir müssen hier nur noch den ‚schnellen' maskierbaren Interrupt erklären. Der FIRQ *(Fast Interrupt Request)* rettet lediglich den Programmzähler und das Statusregister in den Stack. Nur darin unterscheidet er sich vom IRQ, der sämtliche Registerinhalte abspeichert. Dementsprechend geschieht das „Umschalten" auf die FIRQ-Ausnahmeroutine erheblich schneller als auf die IRQ-Routine

2.2.5 Die Behandlung mehrerer Interruptquellen

In einem komplexen System gibt es in der Regel viele Komponenten, die eine Unterbrechungsanforderung an den Prozessor stellen können. Da die Anzahl der Anschlußstifte des Gehäuses möglichst klein gehalten werden muß, kann der Prozessor nicht jeder Quelle einen eigenen Interrupteingang zur Verfügung stellen. Daher sind – im Minimalfall – alle Komponenten, die einen Interrupt auslösen können, über eine gemeinsame Leitung entweder am IRQ- oder am NMI-Eingang des Prozessors angeschlossen. Daraus ergibt sich für den Prozessor das Problem, beim Auftreten eines Interrupts festzustellen, welche Quelle die Unterbrechung angefordert hat.

[8] Die Motorola-Prozessoren benutzen das sog. Big-Endian-Format, bei dem die höherwertigen Adreßteile unter niederwertigen Speicheradressen abgelegt werden.

[9] Durch das vorgestellte ‚$'-Zeichen werden im gesamten Buch hexadezimale Zahlen gekennzeichnet. Üblich sind auch das Präfix ‚0x...' oder das Postfix ‚...h'.

2.2.5.1 Das Polling-Verfahren

Die einfachste Methode, die bei den 8-bit-Prozessoren bevorzugt angewendet wird, bezeichnet man mit *Polling*[10]. Bei diesem Verfahren fragt der Prozessor zu Beginn der Interruptroutine nach einer fest vorgegebenen Reihenfolge alle Komponenten ab, die einen Interrupt auslösen können. Diese Abfrage besteht im wesentlichen darin, das Statusregister der Komponente zu lesen und darin ein bestimmtes Bit zu überprüfen. Dieses Bit wird *Interrupt Flag* (IF) genannt. Es wird von der Komponente immer dann gesetzt, wenn sie ihren Interruptausgang aktiviert hat[11]. Sobald der Prozessor eine Komponente mit gesetztem Interrupt Flag findet, bricht er die Abfrageroutine ab und ruft eine spezifische, der Interruptquelle zugeordnete Ausnahmeroutine auf. Nach Ausführung der Anforderung wird die Interruptroutine (über den Befehl RTI) wieder verlassen.

Eine gewisse Zeit nach der Abarbeitung eines Interrupts werden in der Regel erneut Interruptanforderungen ausgelöst. Andererseits können aber auch schon während der Abarbeitung des letzten Interrupts weitere Anforderungen aufgetreten sein. Es gibt zwei gebräuchliche Alternativen, wie der Prozessor mit diesen Interruptanforderungen verfahren kann. Beide Varianten sind im Bild 2.2-1 skizziert.

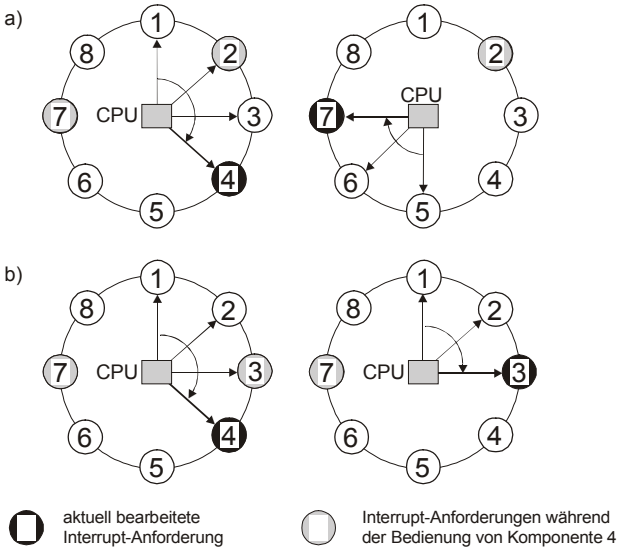

Bild 2.2-1: Zwei verschiedene Varianten des Polling-Verfahrens,
a) faire Zuteilung, b) Zuteilung mit festen Prioritäten

- Bei der ersten Variante wird nach dem erneuten Eintritt in die Interruptroutine die zyklische Abfrage mit derjenigen Komponente fortgesetzt, die in der vorgegebenen

[10] Deutsche Begriffe wie „Abfrageverfahren“ oder „Wählverfahren“ haben sich nicht durchgesetzt.
[11] Im Kapitel II.3 wird genauer auf den Aufbau und die Funktion dieser Komponenten eingegangen.

Reihenfolge der zuletzt bedienten Komponente folgt. Diese Variante bietet allen Komponenten die gleiche Chance, bedient zu werden. Sie wird deshalb als „faire" Prozessorzuteilung bezeichnet. Ihr Einsatz empfiehlt sich dann und nur dann, wenn alle Komponenten die gleiche Wichtigkeit besitzen.

- Bei der zweiten Variante beginnt die Abfrage jeweils wieder mit der in der gegebenen Reihung eindeutig festgelegten ersten Komponente. Bei dieser Variante ist eine Komponente um so bevorzugter, je weiter sie in der vorgegebenen Reihenfolge „vorne steht". Auf diese Weise werden die Komponenten mit bestimmten Prioritäten *(Priority Level)* versehen. Komponenten mit hoher Priorität werden außerdem schneller bedient, da die Abfragezeit bis zu ihrer Identifizierung kürzer ist. Diese Variante wird häufiger als die erste eingesetzt, da die verschiedenen Komponenten eines Systems in der Regel unterschiedlich wichtige Aufgaben für das System ausführen.

2.2.5.2 Die Ausnahmevektor-Tabelle

Der größte Nachteil des Polling-Verfahrens ist die Tatsache, daß durch die zyklische Abfrage aller möglichen Komponenten u.U. unvertretbar viel Zeit gebraucht wird, bis die Interruptquelle identifiziert ist. Dieser Zeitverlust ist bei den modernen 16/32-bit-Prozessoren nicht mehr zu vertreten. Bei ihnen hat man deshalb das folgende Verfahren entwickelt, um die Quelle eines Interrupts möglichst schnell ermitteln zu können[12].

Die Startadressen aller Ausnahmeroutinen werden in einer Tabelle zusammengefaßt, die wir verkürzend mit (Ausnahme-)**Vektortabelle** bezeichnen wollen. Im Englischen sind dafür die Begriffe *Exception Vector Table* und (vereinfachend) *Interrupt Vector Table* üblich. Typisch sind Tabellen mit bis zu 256 Einträgen. Dem entsprechend sind die Vektornummern 8 bit lang. Die Tabelle 2.2-2 stellt ein Beispiel solch einer Vektortabelle dar. Die letzte Spalte der Tabelle zeigt die relative Adresse jedes Eintrags, bezogen auf den Wert des Basisregisters, wenn ein Eintrag (beispielsweise) 4 byte lang ist. Sie berechnet sich aus dem Produkt von Index und Eintragslänge. Im betrachteten Fall umfaßt die gesamte Vektortabelle 1024 byte.

Die Vektortabelle ist häufig in einer bestimmten, meistens der untersten Speicherseite abgelegt, d.h. unter den niederstwertigen Adressen. Bei vielen Prozessoren kann sie aber auch an beliebiger Stelle im Speicher untergebracht werden. Diese Prozessoren besitzen dazu ein spezielles Basisadreß-Register (*Vector Base Register*, s. Unterabschnitt 2.5.1), das die Basisadresse der Tabelle enthält. Jeder Ausnahmeursache wird als Vektornummer der Index zugeordnet, unter dem die Startadresse ihrer Behandlungsroutine in der Tabelle gespeichert ist. Diese Startadresse wird auch Vektor oder Zeiger *(Pointer)* genannt. Durch Einschreiben eines neuen Werts in das Basisregister, kann jederzeit (durch das Betriebssystem) auf eine andere Vektortabelle umge-

[12] Zur Vereinfachung der Darstellung gehen wir dabei davon aus, daß der Prozessor im sog. 'realen' Betriebsmodus arbeitet, d.h. hier, die Unterbrechungsroutinen werden durch absolute physikalische Adressen angesprochen. Wie ihre Startadressen gewonnen werden, wenn der Prozessor im 'virtuellen' Betriebsmodus arbeitet, also unter Anwendung einer virtuellen Speicherverwaltung, werden wir erst im Abschnitt 5.8 beschreiben.

schaltet und dadurch jeder Ausnahmeursache *(Exception)* eine neue Behandlungsroutine zugewiesen werden.

Tabelle 2.2-2: Beispiel einer Vektortabelle

Index	**Ausnahmesituation**		rel. Adr.
0	*Divide by 0*	Division durch 0	$000
1	*Overflow*	Zahlenbereichsüberschreitung	$004
2	*Array Bound Check*	Indexbereichsüberschreitung	$008
3	*Invalid Opcode*	illegaler Befehlscode	$00C
4	*SVC (Supervisor Call)*	Betriebssystemaufruf	$010
5	*Privilege Violation*	unerlaubter Aufruf privilegierten Operation	$014
6	*Trace*	Einzelschritt-Modus	$018
7	....		$01C
	....	(weitere Traps)	
i	....		(i*4)
i+1	RESET	Rücksetzen	
i+2	BERR	Busfehler	
i+3	NMI	nicht maskierbarer Interrupt	
-	....		
k	.....	(weitere Ausnahmesituationen)	(k*4)
k+1	*Error*	Coprozessor-Fehler	
-	....	(weitere Coprozessor-Meldungen,	
l		z.T. über spezielle Statusleitungen)	(l*4)
l+1	*Page Fault*	Seitenfehler (s. Unterabschnitt 5.4.4)	
-	....	(weitere Fehler der	
m	....	Speicherverwaltung)	(m*4)
m+1	....	(reserviert für zukünftige Erweiterungen	
-	....	und für Testzwecke des Herstellers)	
n	....		(n*4)
n+1	*User Vectors*	(durch Systementwickler frei	
-	....	zuzuordnende, maskierbare Interrupts)	
255	....		$3FC

Für die Ausnahmesituationen, welche die Überwachung des Systems steuern, wie Traps, RESET und NMI, werden meist die ersten Tabellenplätze reserviert. Es folgen einige Gruppen von Ausnahmesituationen, die durch spezielle Komponenten verursacht werden, wie Coprozessoren und Speicherverwaltungsbausteine. Für jede der eben erwähnten Unterbrechungen werden die Tabellenplätze – genauer ihre relative Lage in der Tabelle – vom Steuerwerk fest vorgegeben und die Vektoren prozessorintern erzeugt.

Die maskierbaren Interrupts belegen meist die letzten Tabellenplätze. Der Systementwickler kann im Prinzip jeder Interruptquelle einen dieser Tabellenplätze frei zuordnen. Nur wenn auf dem Mikrorechner ein universelles Betriebssystem laufen soll, ist er aus Kompatibilitätsgründen gezwungen, sich an die vom Betriebssystem vorgegebene Zuteilung der Tabellenplätze zu halten.

Bei Verwendung der virtuellen Speicherverwaltung (s. Kapitel 5) können die Einträge der Tabellen neben den Startadressen weitere Informationen, wie z.B. Zugriffsrechte auf die Ausnahmeroutinen, enthalten. Dies ist u.a. bei den Intel-Prozessoren der Fall, bei denen die Einträge 8 byte lang sind. Bei DSPs und Mikrocontrollern werden z.T. in der Vektortabelle nicht die Startadressen, sondern die ersten Befehle der Ausnahmeroutinen selbst eingetragen. Je nach Prozessortyp können dann kurze Ausnahmeroutinen, die ganz in einen Tabelleneintrag passen, sehr schnell abgearbeitet werden *(Fast Interrupts)*, da sie ohne Retten des Prozessorzustands auf dem Stack ausgeführt werden. Längere Ausnahmeroutinen (an beliebiger Stelle im Speicher) müssen durch einen Unterprogrammsprung im Tabelleneintrag angesprochen werden. Bei anderen Prozessoren wird der Prozessorzustand in jedem Fall gerettet, und längere Ausnahmeroutinen werden durch einen Sprungbefehl im Tabelleneintrag ausgeführt.

Immer wenn ein (maskierbarer) Interrupt auftritt, prüft das Steuerwerk des Prozessors zunächst, ob das *Interrupt Enable Bit* (IE) in seinem Steuerregister gesetzt ist (s. Unterabschnitt 2.2.1). Nur wenn das der Fall ist, führt es einen speziellen Buszyklus zur Identifizierung der Interruptquelle durch *(Interrupt Acknowledge Cycle)*. Ein Beispiel dafür ist im Bild 2.2-2 als Zeitdiagramm dargestellt.

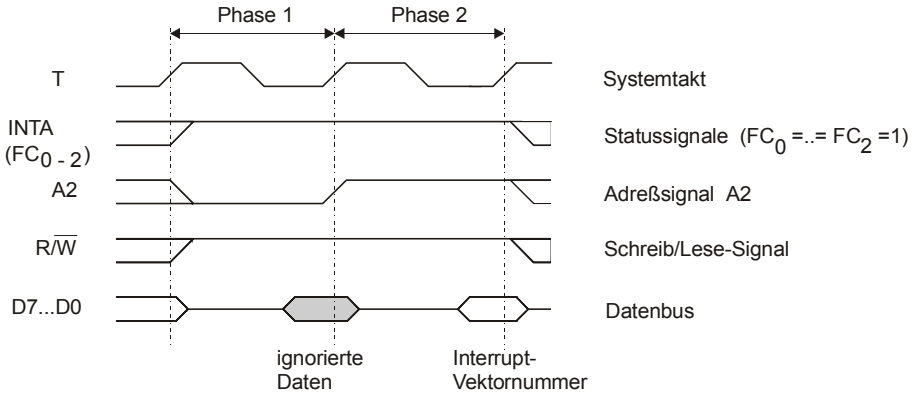

Bild 2.2-2: Zyklus zur Identifikation der Interruptquelle

In Phase 1 des Zyklus teilt der Prozessor über seine Statussignale zunächst allen Komponenten mit, daß er eine Interruptanforderung bearbeiten will. Diese Information, z.B. $FC_2 =...= FC_0 =1$, wird von allen Komponenten decodiert. Bei anderen Prozessortypen erfolgt diese Mitteilung über ein spezielles Ausgangssignal INTA *(Interrupt Acknowledge)*. Das Ausgangssignal des Decoders bzw. INTA aktiviert in der Komponente, welche die Interruptanforderung gestellt hat, ein Register, in dem die zugeord-

nete Interrupt-Vektornummer (IVN) gespeichert ist. (Anstelle des Registers kann auch ein Schaltnetz vorhanden sein, in dem der Vektor fest verdrahtet ist.)

In der Phase 2 wird nun der Registerinhalt auf den unteren Teil des Datenbusses gegeben und vom Prozessor eingelesen. Damit ist der Prozessor in der Lage, die Quelle eindeutig zu identifizieren, die Startadresse der entsprechenden Ausnahmeroutine aus der Vektortabelle zu ermitteln und diese Routine auszuführen.

Zur Unterscheidung der beiden Phasen benutzt der Prozessor in unserem Beispiel das Adreßsignal A2. Gibt die ausgewählte Komponente am Ende der Phase 1, die ja prozessorseitig einen Lesezyklus darstellt, Daten auf den Systembus, so werden diese vom Prozessor ignoriert.

Wie die Ermittlung der Startadresse vor sich geht, ist im Bild 2.2-3 skizziert. Darin sieht man, daß sich die Startadresse aus der Addition des Inhalts des Basisadreß-Registers und der relativen Adresse des indizierten Eintrags der Vektortabelle ergibt. Wie oben beschrieben, berechnet sich die relative Adresse durch die Skalierung der Vektornummer IVN, wobei der konstante Skalierungsfaktor durch die Anzahl der Bytes pro Tabelleneintrag gegeben ist.

Wie bereits gesagt, hängen in einem komplexen Mikrorechner-System viele mögliche Interruptquellen an einem gemeinsamen Eingang (IRQ, INT). In diesem Fall kann durch einen Interrupt-Controller bei gleichzeitigen Unterbrechungsanforderungen festgestellt werden, welche Quelle zunächst bedient werden soll. Diese Quelle oder – häufiger – der Interrupt-Controller selbst liefert dann die IVN an den Prozessor. (Auf Interrupt-Controller gehen wir im Abschnitt II.3.2 ausführlich ein.)

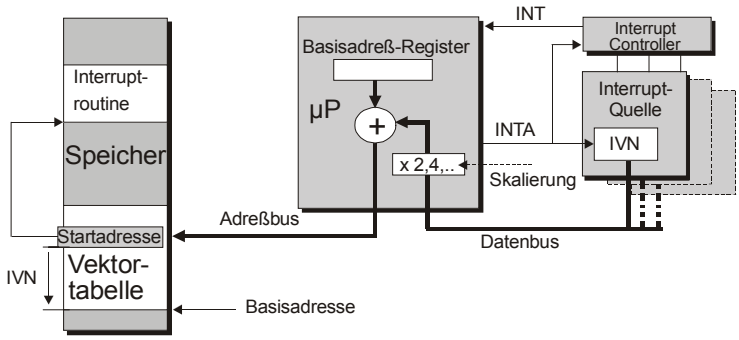

Bild 2.2-3: Die Berechnung der Startadresse einer Interruptroutine

Software-Interrupts

Durch den oben beschriebenen Software-Interrupt INT n wird das Steuerwerk veranlaßt, den Eintrag mit dem Index n (meist $0 \leq n \leq 255$) aus der Vektortabelle in den Programmzähler zu laden und danach die entsprechende Ausnahmeroutine auszuführen. Mit diesem Befehl ist der Zugriff auf alle Unterbrechungsroutinen möglich.

2.2.6 Prioritäten bei mehrfachen Unterbrechungen

Wie oben erwähnt, werden Unterbrechungsanforderungen vom Steuerwerk des Prozessors i.d.R. erst nach der Ausführung der aktuellen Operation erfüllt. Natürlich kann es passieren, daß während einer Befehlsausführung mehrere Anforderungen gleichzeitig oder kurz hintereinander auftreten. Am Ende der Befehlsausführung stellt das Steuerwerk des Prozessors dann das simultane Vorliegen dieser Anforderungen fest. In diesem Fall muß der Prozessor die Ausnahmesituationen in einer gewissen Reihenfolge behandeln. Diese Reihenfolge ist bei den Mikroprozessoren in der Regel fest vorgegeben. Durch sie werden den verschiedenen Anforderungen bestimmte **Prioritäten** zugeordnet. Der oben beschriebene Einsatz des nicht maskierbaren Interrupts (NMI) zur Erkennung von Ausfällen der Versorgungsspannung verlangt beispielsweise, daß der NMI stets vor den maskierbaren Interrupts (IRQs) ausgeführt wird.

Zu beachten ist jedoch, daß durch diese Prioritäten nur die Reihenfolge vorgegeben ist, in der die Behandlung der Interrupts begonnen wird. Werden in einer Ausnahmeroutine nicht alle anderen Unterbrechungen gesperrt, so kann es passieren, daß noch vor dem ersten Befehl der Routine eine gleichzeitig vorliegende Ausnahmesituation mit niedrigerer Priorität behandelt wird. Dies ist z.B. bei den Traps der Fall, die in der Regel die Interrupteingänge nicht sperren. Vor der Ausführung der Trap-Routine kann daher beispielsweise noch ein maskierbarer Interrupt (IRQ) ausgeführt werden.

Die Vergabe der Prioritäten ist bei allen Mikroprozessor-Herstellern unterschiedlich. Gemeinsam ist allen nur, daß das Rücksetzen (RESET) und die Traps i.d.R. hohe Prioritäten zugewiesen bekommen. Tabelle 2.2-3 zeigt beispielhaft die Prioritäten bei den Prozessoren der Firma Intel.

Tabelle 2.2-3: Prioritäten bei der Ausnahmebehandlung der Intel-Prozessoren

Priorität	Ausnahmesituation	
0	RESET	Rücksetzen, Initialisieren des Systems
1	TRAPS INT	Ausnahmesituation bei der Befehlsausführung als Spezialfall: Software-Interrupt
2	TRACE	Einzelschrittausführung
3	NMI	nicht maskierbarer Interrupt
4	...	Coprozessor-Fehler
5	IRQ	maskierbarer Interrupt

Durch die beschriebene Vergabe unterschiedlicher Prioritäten ist noch nicht das oben angesprochene Problem gelöst, wie der Prozessor „gleichzeitig“ auftretende maskierbare Interrupts (IRQs) behandeln soll. Eine Möglichkeit, das Polling, wurde bereits beschrieben. Wie gesagt, ist dieses Verfahren jedoch sehr zeitaufwendig. Im Abschnitt

II.3.2 wird ein Spezialbaustein beschrieben, der die Aufgabe hat, die gleichzeitig vorliegenden IRQs in eine bestimmte Reihenfolge zu bringen. Dieser *Interrupt Controller* löst diese Aufgabe hardwaremäßig und damit sehr viel schneller.

2.2.7 Exkurs: Interruptkontrolle der Prozessoren MC680x0

Anhand der Motorola-Prozessoren 680x0 (x=0,..,4) soll hier schon kurz auf eine Realisierungsmöglichkeit eines einfachen Interrupt-Controllers eingegangen werden. Dieser ist bereits auf dem Chip der Prozessoren integriert.

Die Prozessoren besitzen drei Interruptanschlüsse IPL2#,IPL1#,IPL0# (*Interrupt Priority Level*, vgl. Bild 2.2-4). Alle interruptfähigen Komponenten sind mit diesen Anschlüssen verbunden. Wird von keiner Komponente ein Interrupt gewünscht, so liegen alle drei Eingänge auf H-Potential, sind also logisch ‚1'. Jede Komponente, die einen Interrupt auslösen will, legt eine bestimmte, ihr fest zugeordnete Bitkombination ($\neq$ 111) auf die Anschlußleitungen IPL2#,IPL1#,IPL0#.

Den Anschlüssen sind im Status-/Steuerregister SR des Prozessors die drei Bits I2, I1,I0 als **Interruptmaske** *(Interrupt Priority Mask)* zugeordnet. Diese drei Bits, aufgefaßt als Dualzahlen, definieren 2^3=8 Prioritätenebenen. Von diesen wird die Ebene 0 nicht benutzt, da sie den Fall repräsentiert, daß kein Interrupt vorliegt, also dem Eingangszustand IPL2#=IPL1#=IPL0#=1 entspricht.

Jede Interruptanforderung wird durch ihre Eingangsbelegung IPL2#,IPL1#,IPL0# in eine Prioritätenklasse eingeteilt. Sie wird bestimmt durch die als Dualzahl aufgefaßte invertierte Information (IPL2,IPL1,IPL0) der Interrupteingänge. Diese Bitkombination bestimmt die Priorität der Komponente nur gegenüber denjenigen Komponenten, die eine andere Bitkombination anlegen[13].

Einer Interruptanforderung wird höchstens dann stattgegeben, wenn sie einer Klasse angehört, deren Nummer größer ist als die durch die Steuerbits I2,I1,I0 vorgegebene Ebenennummer. Eine Ausnahme bildet die Klasse 7 [(IPL2,IPL1,IPL0) = 111], die für den nicht maskierbaren Interrupt (NMI) benutzt wird. In diesem Fall werden die Steuerbits I2,I1,I0 nicht ausgewertet und die Interruptroutine unbedingt ausgeführt. Die Klassen 1–6 umfassen die maskierbaren Interrupts. Durch diese Vorgabe bekommt die Klasse 7 die höchste Priorität, die Klasse 1 die niedrigste.

Beispiele

- Liegt an den Eingängen die Information IPL2# = 0, IPL1# = 1, IPL0# = 0, so gehört die Anforderung zur Klasse (IPL2,IPL1,IPL0) = $101_2 = 5_{10}$.
- Sind die Interruptbits des Steuerregisters durch I2,I1,I0 auf die 3. Ebene gesetzt, so werden nur Anforderungen der Klassen 4–7 berücksichtigt, denen die dualen Eingangsinformationen (IPL2#,IPL1#,IPL0#) = 011, 010, 001, 000 zugeordnet sind.

[13] Am Ende der Fallstudie wird gezeigt, wie Prioritäten zwischen denjenigen Komponenten verteilt werden können, die dieselbe Bitkombination auf die Eingänge IPL2# – IPL0# geben.

Tabelle 2.2-4 zeigt in aufsteigender Reihenfolge die Prioritäten, den entsprechenden Zustand der Steuerbits I2,I1,I0 sowie die möglichen Kombinationen der Interruptsignale IPL2#,IPL1#,IPL0# für alle zugelassenen Interruptklassen.

Tabelle 2.2-4: Interruptprioritäten bei den Motorola-Prozessoren MC680x0

Priorität Ebene	**Statusbits I2,I1,I0**	**zugelassene Int.-Klassen**	**Eingangskombinationen** IPL2#,IPL1#,IPL0#
0	0 0 0	1 – 7	000 – 110
1	0 0 1	2 – 7	000 – 101
2	0 1 0	3 – 7	000 – 100
3	0 1 1	4 – 7	000 – 011
4	1 0 0	5 – 7	000 – 010
5	1 0 1	6 – 7	000 – 001
6	1 1 0	7 (nur NMI)	000
7	1 1 1	keine	---

Die von der Komponente ausgegebene Bitkombination muß stets solange an den Anschlüssen IPL2# – IPL0# anliegen, bis der Prozessor über seine Statusleitungen FC_2 – FC_0 (durch die Bitkombination ‚111‘) anzeigt, daß er der Interruptanforderung stattgegeben hat (vgl. Unterabschnitt 2.2.1). Gleichzeitig gibt er über die Adreßleitungen A3, A2, A1 die Nummer der akzeptierten Interruptklasse aus. Anhand dieser Nummer kann jede Komponente feststellen, ob der Prozessor ihre Interruptanforderung oder aber eine eventuell gleichzeitig vorliegende höher privilegierte Anforderung bearbeitet.

Zu Beginn der Interruptbearbeitung werden die Steuerbits I2,I1,I0 im Steuerregister automatisch auf die Prioritätsebene des aktuellen Interrupts gesetzt. Dadurch sind während der Ausführung der Interruptroutine nur noch Interrupts mit höherer Priorität zugelassen. Wie bereits gesagt, hat der NMI die höchste Priorität 7 und kann daher alle (maskierbaren) Interrupts der Klassen 1–6 unterbrechen. Wird aktuell gerade eine NMI-Routine abgearbeitet, d.h. $I2,I1,I0 = 111$, so sind ohne weiteres keine anderen Unterbrechungen zugelassen. Jedoch kann in diesem Fall, wie auch bei der Abarbeitung eines maskierbaren Interrupts, der Anwender durch bestimmte (privilegierte) Befehle das Steuerregister gezielt mit einer speziellen Interruptmaske I2,I1,I0 laden und dadurch das Steuerwerk auch Interruptanforderungen niedrigerer Priorität akzeptieren lassen. Jede Interruptroutine wird mit dem Befehl RTE *(Return from Exception)* beendet. Dabei wird wieder die vor ihrer Ausführung gültige Interruptmaske ins Statusregister geladen.

Das Daisy-Chain-Verfahren

Werden mehr als 7 Interruptquellen in einem System eingesetzt, müssen diese auf die Klassen 1–7 aufgeteilt werden. Innerhalb der einzelnen Klassen muß nun wieder nach

einem geeigneten Verfahren eine Auswahl stattfinden, z.B. softwaremäßig durch das oben beschriebene Polling-Verfahren.

Ein häufig benutztes Hardwareverfahren ist das *Daisy Chaining*[14]. Es soll hier für den MC680x0 skizziert werden.

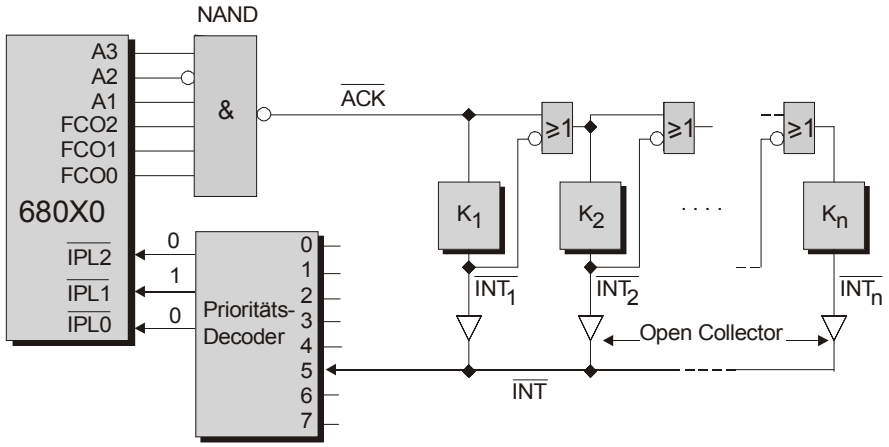

Bild 2.2-4: ***Daisy*** *Chaining* beim Motorola 680x0

Alle Komponenten K_i der gleichen Prioritätsklasse stellen ihre Interruptanforderung INT_i durch ein L-Signal auf der gemeinsamen Leitung INT. An dieser sind sie z.B. über *Open-Collector*-Gatter angeschlossen. Das INT-Signal wird auf einen der acht Eingänge eines Prioritätendecoders gegeben. Die Nummer dieses Decodereingangs legt die Klassennummer (hier z.B. 5) fest. Der Decoder erzeugt entsprechend dieser Nummer die Belegung der Prozessoreingänge IPL2# – IPL0#, im Beispiel also (IPL2#,IPL1#,IPL0#)=010. Liegen an mehreren Eingängen des Decoders gleichzeitig Anforderungssignale, muß er nach fest vorgegebenen Prioritäten genau einen dieser Eingänge auswählen. Diese Prioritätenreihenfolge kann zwar, sie muß aber nicht mit der auf den Prioritätenklassen des Prozessors vorgegebenen übereinstimmen. Insgesamt erhält man nun eine dreistufige Auswahl unter den Interruptquellen:

- Der Prozessor regelt durch seine Klassenvergabe lediglich, ob eine weitere, zwischenzeitlich auftretende Anforderung eine so hohe Priorität hat, daß sie eine laufende Interruptroutine unterbrechen darf.
- Der Prioritätendecoder hingegen entscheidet durch Vergabe einer eindeutigen Priorität, welcher von mehreren gleichzeitig vorliegenden Anforderungen (aus verschiedenen Prioritätenklassen) zuerst stattgegeben wird.
- Durch das *Daisy Chaining* wird nun seinerseits entschieden, welche von mehreren Interruptquellen aus einer bestimmten, festen Prioritätenklasse zuerst bedient wird.

[14] *Daisy Chain*: Kette aus Gänseblümchen

Der Prozessor zeigt die Annahme der Interruptanforderung über seine Statussignale FC_2 – FC_0, genauer durch FC_2 =...= FC_0 = 1. Über die Adreßleitungen A3, A2, A1 wird gleichzeitig die Prioritätsebene der Unterbrechung ausgegeben. (Im Beispiel: A3,A2,A1 = 5). Aus diesen Signalen erzeugt die NAND-Schaltung das Quittungssignal. Dieses wird nun – „verkettet" mit den Interrupt-Anforderungssignalen INT_i – sukzessiv zu den Komponenten K_i geführt. Genauer betrachtet, leitet die Komponente K_i das Signal über das ODER-Gatter nur dann an K_{i+1} weiter, wenn sie selbst keinen Interrupt angefordert hat. Auf diese Weise hat (innerhalb der festen Prozessor-Prioritätsklasse) jede Komponente eine um so höhere Priorität, je „näher" sie dem Prozessor ist.

2.3 Adreßwerk

Das Adreßwerk (*Address Unit* – AU, *Address Generation Logic*) hat die Aufgabe, nach den Vorgaben des Steuerwerks aus den Inhalten bestimmter Speicherzellen und Register[1] die Adresse eines gewünschten Befehls oder Operanden zu bilden. Es ist sicher die interne Komponente der Mikroprozessoren, die in den letzten Jahren den stärksten Entwicklungsschüben unterworfen war. Bei den älteren 8-bit-Prozessoren wurde diese Adreßberechnung zum großen Teil noch von der ALU des Operationswerks vorgenommen, so daß bei ihnen ein spezielles Adreßwerk entfiel. Jedoch besaß bereits Mitte der 70er Jahre der µP 2650 von Valvo einen speziellen Adreßaddierer *(Offset Adder)*. Seitdem sind viele Funktionen zu den Aufgaben des Adreßwerks hinzugekommen. Dadurch wird es möglich, die Adreßberechnung parallel zu den Aktivitäten des Operationswerks durchzuführen und somit die Verarbeitungsgeschwindigkeit des Prozessors wesentlich zu erhöhen[2]. Moderne Prozessoren, die (intern) eine Harvard-Architektur aufweisen, besitzen zwei Adreßwerke für die gleichzeitige Berechnung einer Befehls- und einer Operandenadresse. DSPs verfügen sogar über drei Adreßwerke, da sie in jedem Taktzyklus neben einem Befehl auch noch zwei Operanden selektieren müssen (vgl. Abschnitt 6.2).

Zu den Aufgaben des Adreßwerks bei Hochleistungsprozessoren gehört insbesondere die Verwaltung eines virtuellen Speichers. Vereinfachend gesagt, geht es dabei um die Umwandlung von logischen Programmadressen in physikalische Speicheradressen. Bei den älteren Prozessortypen wurde diese Aufgabe von speziellen Bausteinen (*Memory Management Unit* – MMU) vorgenommen. Die neueren 16/32-bit-Prozessoren besitzen jedoch eine (oder mehrere) auf dem Chip integrierte MMU. Um dieses Kapitel inhaltlich nicht zu überfrachten, werden wir auf die MMUs erst im Kapitel 5 ausführlich eingehen. Hier wollen wir nur erwähnen, daß zur virtuellen Speicherverwaltung das Adreßwerk auf eine große Anzahl von Registern und Tabellen

[1] Adreßregister, s. Abschnitt 2.5.1
[2] Wie bereits erwähnt, spricht man hier von *Pipeline*-Verarbeitung. Ein anderes Beispiel für diese Form der Parallelverarbeitung hatten Sie schon bei der Vordecodierung der Befehle im Steuerwerk des Intel 80486 kennengelernt, s. Abschnitt 2.1.

zugreifen muß. Am Ende dieses Abschnitts werden wir eine vereinfachte Darstellung bringen.

Bild 2.3-1 zeigt ein sehr einfaches Beispiel für ein Adreßwerk, zusammen mit den angeschlossenen Komponenten des µPs. In ihm sind, wie angekündigt, alle Komponenten zur Speicherverwaltung nur durch ihre Register und Tabellen repräsentiert.

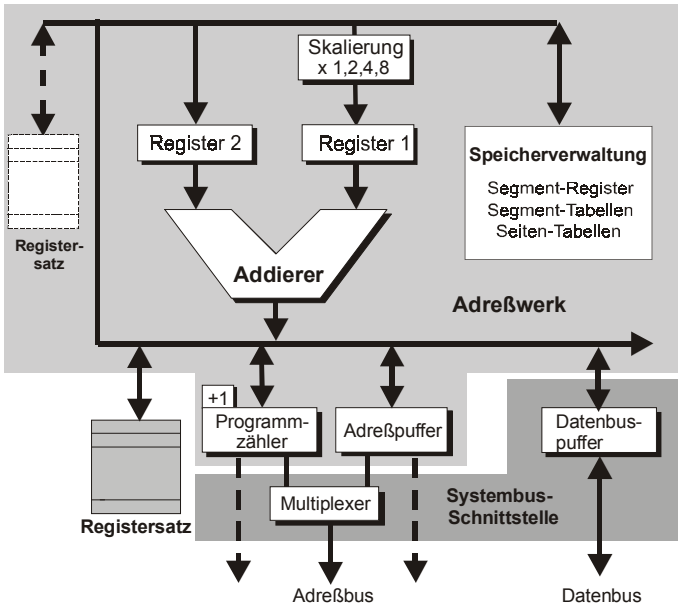

Bild 2.3-1: Aufbau eines einfachen Adreßwerks

Das Adreßwerk besteht im wesentlichen aus einem Adreßaddierer. Als „Akkumulatoren“, also als kombinierte Ein-/Ausgaberegister, dienen diesem Addierer wahlweise der Programmzähler oder der Adreßpuffer, je nachdem, ob die Adresse des nächsten Programmbefehls oder eines Operanden berechnet werden muß.

Wie bereits gesagt, enthält der Programmzähler *(Program Counter)*, der auch Befehlszähler genannt wird (*Instruction Pointer* – IP), stets die Adresse der Speicherzelle, in welcher der nächste auszuführende Befehl (oder Befehlsteil) liegt. Beim sequentiellen, nicht durch Sprünge oder Verzweigungen unterbrochenen Programmablauf wird der Programmzähler mit jedem Befehl (bzw. Befehlsteil) nur um 1 erhöht. (Dies ist im Bild 2.3-1 durch das Symbol ‚+1‘ angedeutet.) Zur Realisierung von Sprüngen und Verzweigungen wird der Programmzähler mit der neuen Programmadresse geladen. Wird durch eine falsch berechnete Sprungadresse nicht zu einem neuen Befehl, sondern statt dessen z.B. in einen Datenbereich des Speichers ver-

zweigt, so führt das in der Regel zu unvorhersehbaren Auswirkungen[3]. Die für die Ausführung eines Maschinenbefehls benötigten Operanden werden durch den Adreßpuffer selektiert. Der Adreßpuffer wird vom Adreßwerk mit der Adresse des Operanden geladen, die nach bestimmten fest vorgegebenen Verfahren berechnet wird. Diese als Adressierungsarten bezeichneten Verfahren werden Sie im Abschnitt 3.3 kennenlernen.

Das Ergebnis der Adreßberechnung wird über die Systembusschnittstelle an die Peripheriekomponenten des Prozessors ausgegeben. Bei Prozessoren mit von-Neumann-Architektur schaltet ein Multiplexer in der Schnittstelle entweder den Programmzähler (zum Zugriff auf den nächsten Befehl) oder den Adreßpuffer (zum Zugriff auf den nächsten Operanden) nach außen. Bei Prozessoren mit Harvard-Architektur werden beide Registerwerte über getrennte Adreßbusse nach außen gegeben (gestrichelte Pfeile im Bild).

Die Werte der Akkumulator-Register werden dem Addierer zugeführt und dort mit weiteren Eingabewerten zur Adreßberechnung verknüpft. Diese Eingabewerte können einerseits aus dem allgemeinen Registersatz entnommen werden[4]. Die Adreßwerke von DSPs zur Berechnung von Operandenadressen *(Data Address Generator/Unit)* verfügen meist über separate Registersätze mit jeweils bis zu 24 Adreßregistern. Andererseits können die Eingabewerte aber auch aus dem Datenbuspuffer in der Systembusschnittstelle des Prozessors kommen. In diesem Fall handelt es sich um eine im Befehl angegebene absolute Adresse oder eine Adreßdistanz zu einer Basisadresse im Registersatz.

Die mit **Skalierung** bezeichnete Komponente existiert hauptsächlich bei 32-bit-Prozessoren. Sie erlaubt es, einen Operanden wahlweise mit den Werten 1, 2, 4 oder 8 zu multiplizieren. Da es sich bei den Eingabewerten zur Adreßberechnung häufig um Adressen oder Adreßdistanzen *(Offsets)* von Speicherzellen handelt, die z.B. in einem Register stehen, kann man sukzessiv auf Datenwörter der Länge 1, 2, 4 oder 8 byte zugreifen, indem man den Registerwert vor der Skalierung jeweils um 1 erhöht bzw. erniedrigt. Dies wird bei den im Abschnitt 3.3 dargestellten Adressierungsarten ausgenutzt[5].

Bei Prozessoren ohne virtuelle Speicherverwaltung bildet das Ergebnis der bisher beschriebenen Adreßberechnung die physikalische Speicheradresse des Befehls oder Operanden. Bei **virtueller Speicherverwaltung** liegt hingegen zunächst eine logische (virtuelle) Adresse vor, die durch mehrfachen Zugriff auf Register und Tabellen der Speicherverwaltung erst in eine physikalische Adresse umgewandelt und danach auf den Adreßbus gegeben wird.

Wie bereits angekündigt, soll anhand von Bild 2.3-2 die Adreßumwandlung der virtuellen Speicherverwaltung nur kurz und vereinfacht dargestellt werden.

---

[3] Das Steuerwerk kann von sich aus nicht feststellen, ob das eingelesene Speicherwort einen Maschinenbefehl oder einen Operanden darstellt. In diesem Fall muß der µP wieder in einen definierten Anfangszustand zurückgesetzt werden.

[4] wozu die Index- und Basisregister herangezogen werden, s. Abschnitt 2.5

[5] Im Unterabschnitt 2.4.2 stellen wir mit dem *Barrel Shifter* eine Schaltung vor, welche die Skalierung in einem Taktzyklus vornehmen kann.

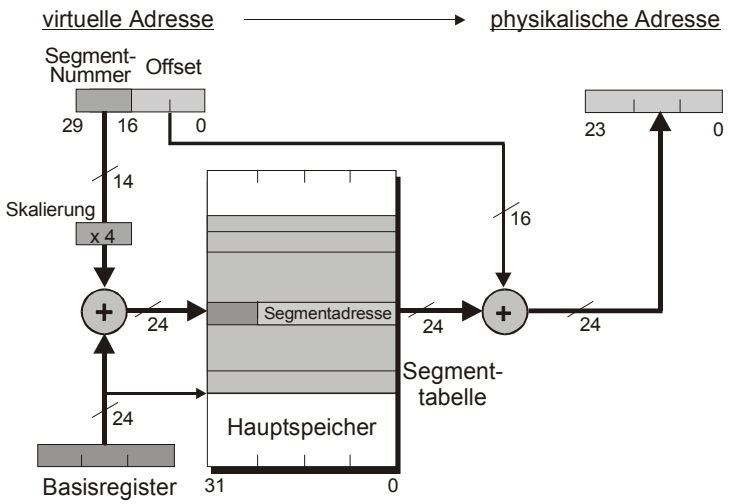

Bild 2.3-2: Adreßumwandlung bei virtueller Speicherverwaltung

Unser hypothetischer Prozessor[6] besitze einen 24-bit-Adreßbus – und damit einen 16 Mbyte umfassenden physikalischen Adreßraum. Dieser wird durch die Speicherverwaltung in Segmente mit maximal 64 kbyte unterteilt. Die im Programm angegebenen virtuellen (logischen) Adressen sind jedoch jeweils 30 bit lang, so daß der virtuelle Adreßraum 1 Gbyte groß ist. Die virtuellen Adressen bestehen aus einer 14 bit langen Segmentnummer und einem 16-bit-Offset, der die Lage eines Datums in dem adressierten Segment angibt.

Das Betriebssystem legt für jedes neue Programm eine eigene Tabelle im Hauptspeicher an, in der die Startadressen aller Segmente[7], die zu diesem Programm gehören, eingetragen werden. Die Anfangsadresse dieser Tabelle wird in ein Basisregister eingetragen.

Bei der Adreßumsetzung wird die 14-bit-Segmentnummer in der virtuellen Adresse durch die Speicherverwaltungseinheit (MMU) mit der Anzahl der Bytes der Einträge (hier 4) skaliert und dann zu dem Inhalt des Basisregisters addiert. Das Ergebnis zeigt auf einen Eintrag in der Segmenttabelle, aus dem die Anfangsadresse des angesprochenen Segments entnommen wird. Die Addition des Offsets in der virtuellen Adresse zu dieser Anfangsadresse ergibt schließlich die gesuchte physikalische Speicheradresse.

Im Kapitel 5 werden wir zeigen, daß reale Prozessoren eine komplexere Form der Speicherverwaltung durchführen, bei der sukzessive auf mehrere Tabellen zugegriffen werden muß.

---

[6] angelehnt an den Intel 80286
[7] Maximal können mit der 14-bit-Segmentnummer 16.384 Segmente verwaltet werden.