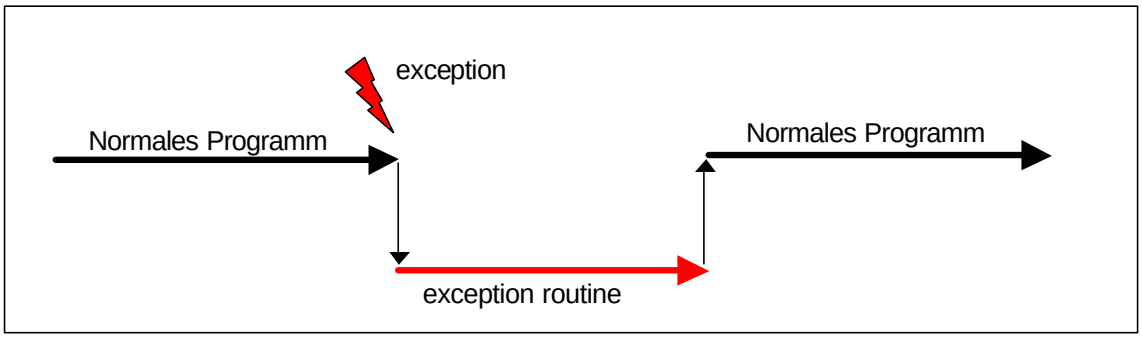
Exceptionhandling (Ausnahmebearbeitung)

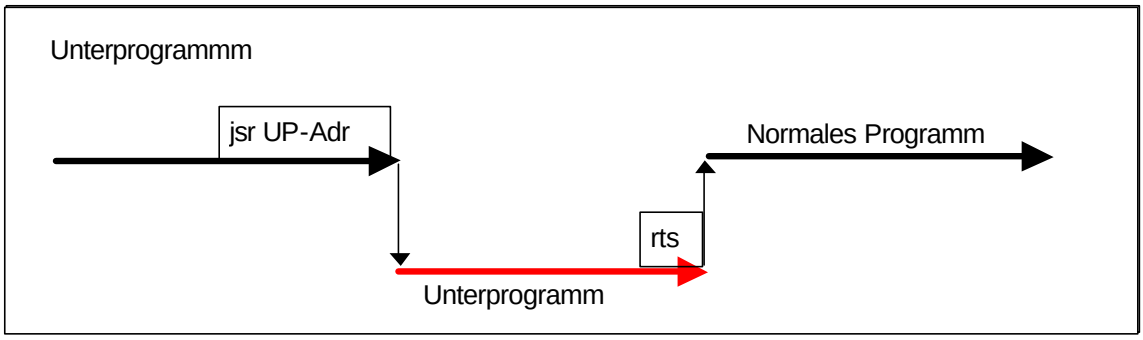
Exception: Exception ist eine besondere Situation, auf die der Prozessor mit einem speziellen Programm (Exceptionroutine) reagieren muss. Das gerade ablaufende „normale“ Programm wird dazu unterbrochen.

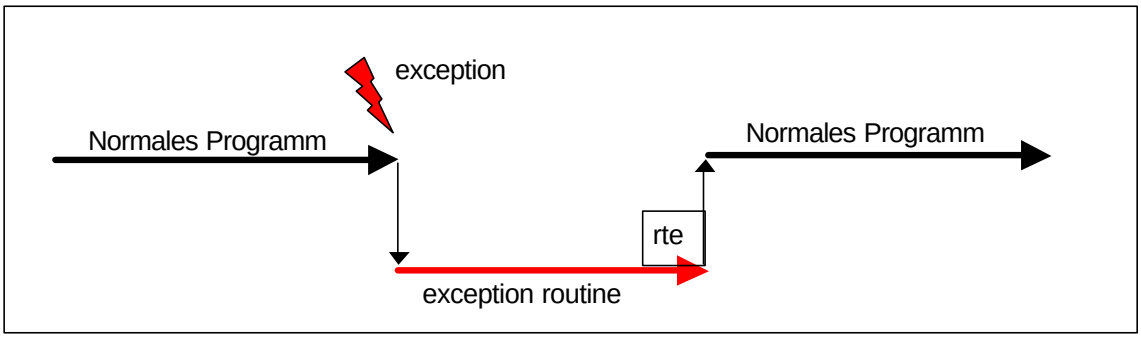


Nach Ablauf der Exceptionroutine wird das normale Programm an der Stelle, an der die Unterbrechung stattfand, wieder fortgesetzt. Es wird daher wie beim Unterprogrammaufruf die Rückkehradresse auf den Stack gerettet.

Unterschied zu Unterprogramm

Der Vorgang erscheint zunächst sehr ähnlich. Der Unterschied liegt jedoch beim Ermitteln der Startadresse.





Während beim Unterprogrammaufruf der Programmierer die Startadresse des Unterprogramms im Programmcode festlegt, muss bei einer Exception die Startadresse der Exceptionroutine von Prozessor ermittelt werden.
Außerdem wird zusätzlich zur Rückkehradresse noch das Prozessorstatuswort auf den Stack gerettet, da ja an jeder beliebigen Stelle des normalen Programms die Unterbrechung erfolgen kann und nach der Unterbrechung die Conditionflags mit ihren ursprünglichen Werten wieder zur Verfügung stehen müssen.

Exceptionursachen

Die möglichen Ursachen werden in interne und externe Ursachen eingeteilt. Interne Ursachen sind Ereignisse, die vom laufenden Programm verursacht werden, z.B. Fehlersituationen. Externe Ursachen werden immer durch Hardware verursacht.

Interne Ursachen

- Software-Interrupt

  Exception wird gezielt vom Programm ausgelöst, zum

  - Testen von Interruptroutinen
  - Aufruf von Betriebssystemfunktionen , die im Supervisormodus ablaufen

- Traps (Falle)

  Exception wird durch einen Fehler im laufenden Programm ausgelöst

  - Division durch Null
  - Ungültiger Befehlscode
  - Adressfehler (z.B. Wortzugriff auf ungerade Adresse)
  - Überschreitung vorgegebener Speichergrenzen (Chk-Befehl)
  - Privilegienverletzung
  - Einzelschrittbetrieb

Externe Ursachen

- Busfehlerleitung /BERR wurde aktiviert.
- Resetleitung /RESET wurde aktiviert.
- Eine Interruptleitung wurde aktiviert.

Interrupts unterscheiden sich von den übrigen Exceptions dadurch, dass sie nicht auf einer Fehlersituation beruhen. Sie sind gewollte Programmunterbrechungen, mit denen der Prozessor auf Anforderungen von Peripheriebausteinen reagiert.

Interrupts werden aufgeteilt in maskierbare und nicht maskierbare Interrupts.

Maskierbar:
Mit einem Flag (oder einer Maske) im Steuerregister des Prozessors legt der Programmierer fest, ob ein Interrupt zugelassen wird.

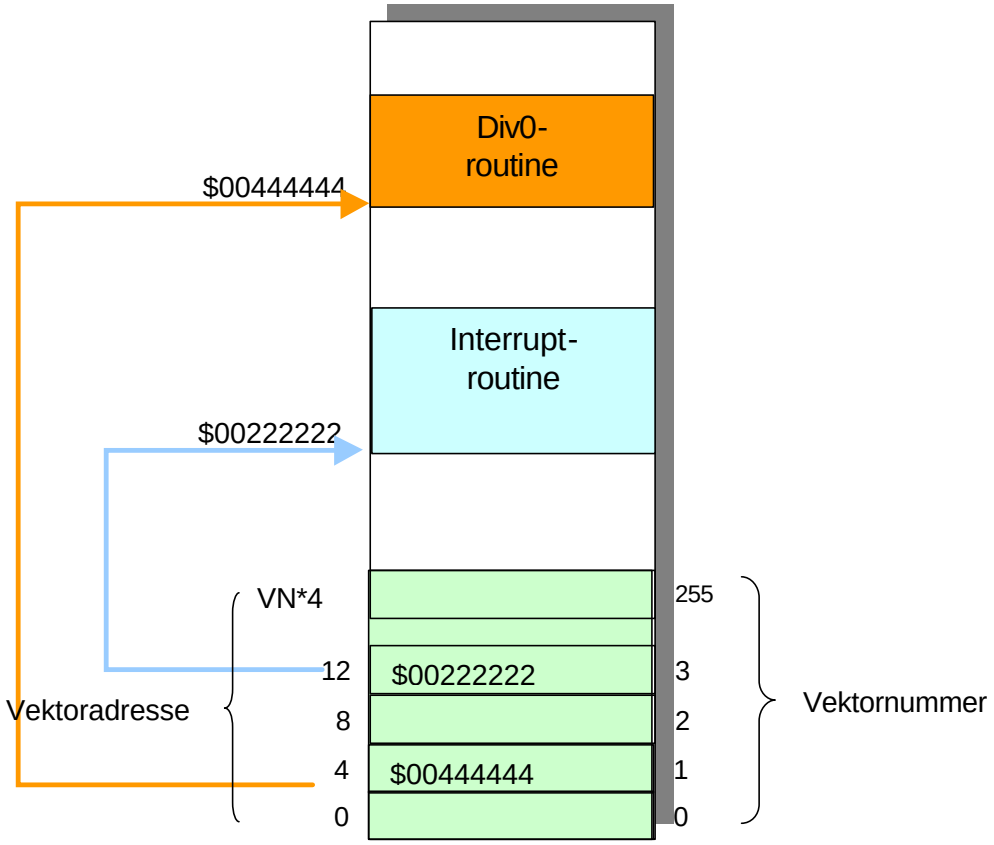
Nicht maskierbar :

Ein nicht maskierbarer Interrupt (NMI) kann nicht gesperrt werden

Ermitteln der Startadresse einer Exceptionroutine

Vektor = Zeiger auf eine Exceptionroutine = Startadresse
Vektortabelle = Tabelle, die die Startadressen aller Exceptionroutinen enthält. Diese Tabelle muss vom Programmierer angelegt werden. Sie liegt beim 68000 an der Speicheradresse 0.

Jede Exception bekommt eine Nummer zugeteilt. Diese Vektornummer gibt den Eintrag in der Vektortabelle an. Durch Auslesen der Vektortabelle an der durch die Vektornummer bezeichneten Position ermittelt der Prozessor die zur Exception gehörende Exceptionroutine.



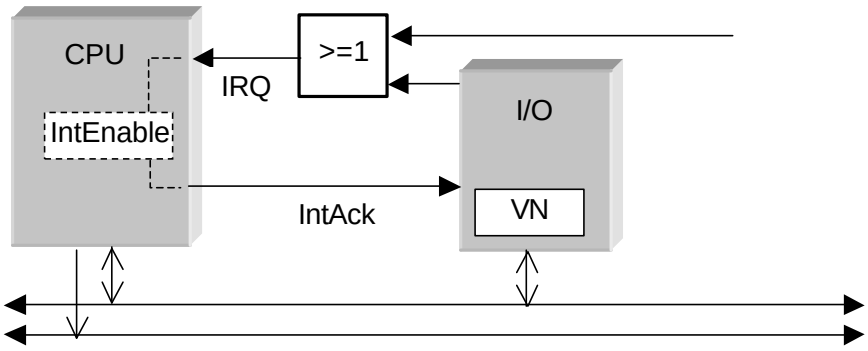
Vektoradresse = Vektornummer x 4 + Basisadresse

(Jeder Vektor belegt 4 Byte, die Basisadresse liegt beim 68000 fest auf 0)

Ablauf eines Interrupts

Prinzip:

Jede Interruptquelle meldet beim Auftreten eines Interrupts eine eindeutige Vektornummer über den Datenbus an die CPU. Die CPU ermittelt mit Hilfe der Vektortabelle die Startadresse der Interrupt-Service- Routine



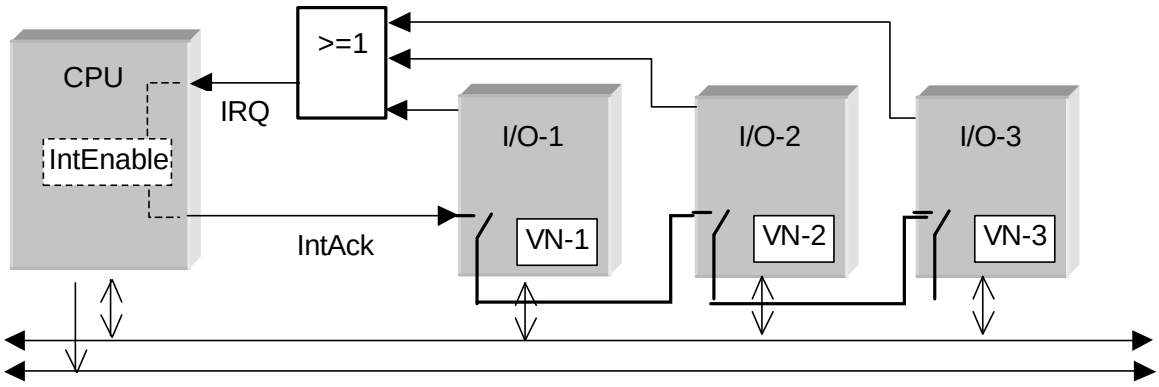
Zeitlicher Ablauf:

- I/O-Baustein setzt das Signal Interruptrequest (IRQ)
- Nach Ablauf des aktuellen Befehls startet der Interruptzyklus der CPU
- Test, ob der Interrupt zugelassen ist (IntEnable)
- Falls IntEnable gesetzt, rettet CPU Programmcounter und Status auf den Stack
- CPU setzt ein Interrupt Acknowledge Signal (IntAck)
- I/O-Baustein legt die Vektornummer auf den Datenbus
- CPU liest die Vektornummer und ermittelt die zugehörige Position in der Vektortabelle
- CPU liest aus der Vektortabelle die Startadresse der Interrupt-Routine aus
- CPU nimmt IntAck zurück
- I/O nimmt IRQ zurück
- CPU führt die Interrupt-Routine aus

Gleichzeitige Interrupts

Falls mehrere Interrupts gleichzeitig auftreten können (womit in der Regel immer gerechnet werden muss), muss eine Priorisierung der Interruptanforderungen erfolgen, da von der CPU immer nur ein Interrupt bearbeitet werden kann.

1. Möglichkeit : **Daisy Chain**

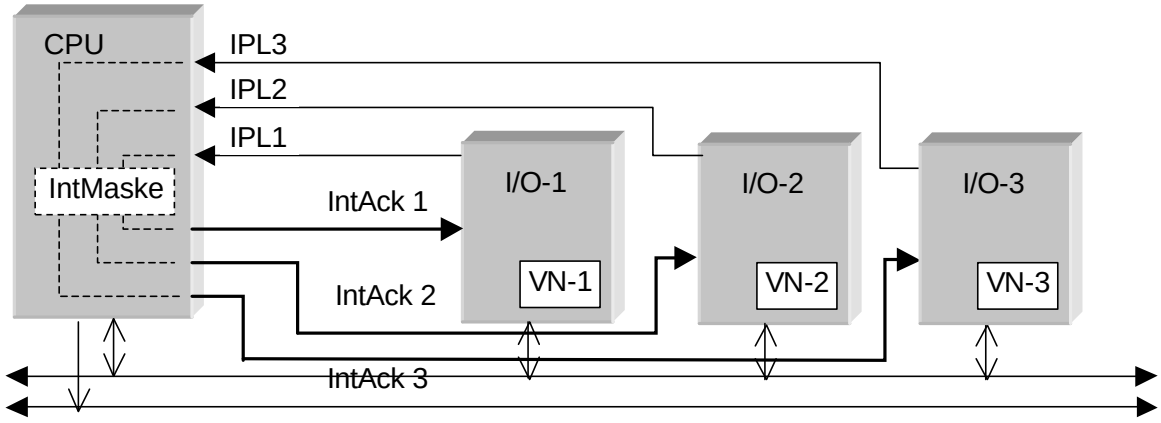


Das IntAck-Signal wird von einer Interruptquelle zur nächsten durchgeschleift. Wenn der entsprechende I/O-Baustein keinen Interruptrequest gesetzt hat, wird das IntAck-Signal zum nächsten Baustein weitergeleitet, bei gesetztem Interruptrequest werden die folgenden I/O-Bausteine abgehängt.

Die Priorisierung ergibt sich durch die Reihenfolge in der Kette, sie wird als **„geographische Priorität“** bezeichnet

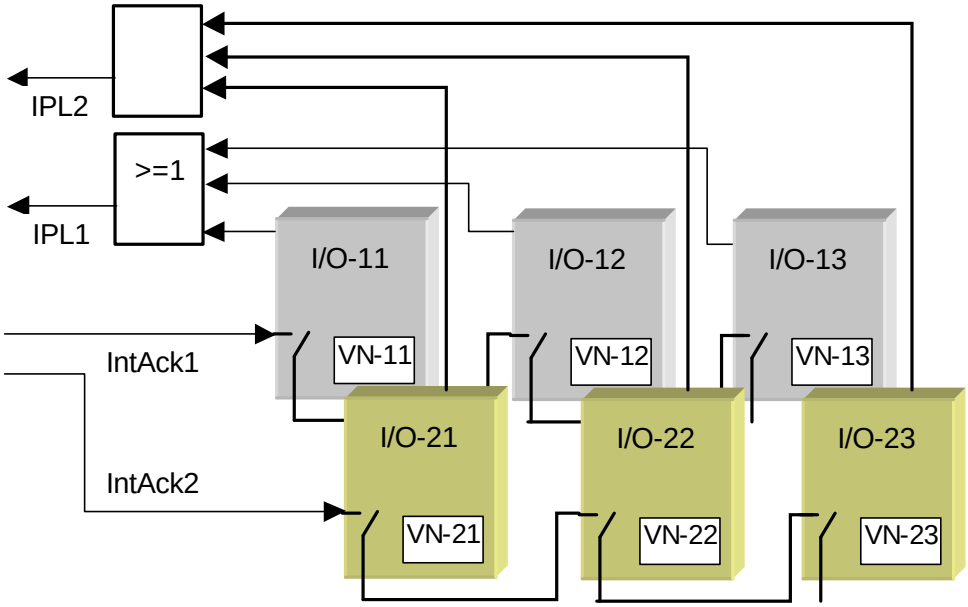
2. Möglichkeit Interruptlevel

Die CPU hat nun mehrere Interrupteingänge, die mit einer Priorität versehen sind. Dazu gehören ebenfalls die entsprechenden IntAck- Signale. Die CPU setzt bei gleichzeitigen Interrupts das IntAck Signal für den Interruptlevel mit der höheren Priorität. Aus dem InterruptEnable Bit wird nun eine Interruptmaske, da nun Interrupt ab einer bestimmten Prioritätsebene gesperrt werden können.



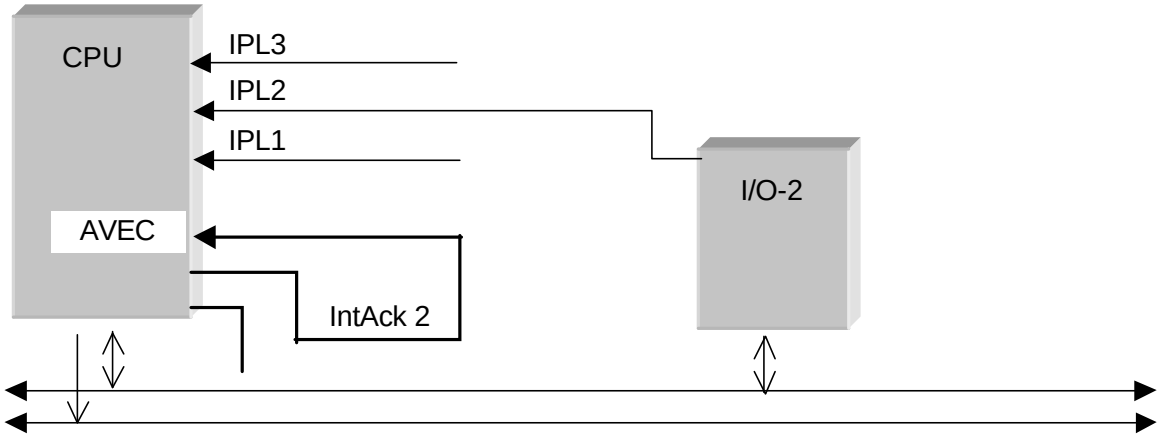
Kombination von Interruptlevel und Daisy-Chain

Die beiden Verfahren lassen sich kombinieren. Einer Gruppe von I/O-Bausteinen wird die selbe Priorität zugeordnet, ihre Interruptrequests werden auf einen Interruptlevel zusammengeführt. Die entsprechende IntAck Leitung wird dann entsprechend dem Daisy Chain Verfahren durchgeschleift.



Autovektor

Einfache Interruptquellen besitzen oft nicht die Funktion, einen Interruptvektor zu verwalten. Zur Unterstützung solcher Interruptquellen verfügen die meisten Mikroprozessoren über Autovektoren.
Ein Autovektor wird von der CPU automatisch erzeugt. Jeder Interruptpriorität ist ein Autovektor fest zugeordnet. Ein zusätzliches Eingangssignal der CPU dient zur Unterscheidung, ob ein externer Vektor existiert oder ob ein Autovektor verwendet wird.



Interruptschema des 68000

- 7 Prioritätsebenen
- 7 Autovektoren
- 192 externe Vektoren
- Interrupt-Eingangssignale /IPL2 – /IPL0
  mit diesen drei Leitungen wird eine Nummer zwischen 1 und 7 erzeugt, die der aktuellen Interruptpriorität entspricht

/IPL2	/IPL1	/IPL0	Maske	Prio
0	0	0		7 = NMI
0	0	1	110	6
0	1	0	101	5
0	1	1	100	4
1	0	0	011	3
1	0	1	010	2
1	1	0	001	1
1	1	1	000	Kein Int.

- „IntAck“ wird generiert mit 3 Functioncodeleitungen FC0 – FC2
  IntAck bedeutet FC0 = 1, FC1 = 1und FC2 = 1
- Die akzeptierte Interruptpriorität wird auf den Adressleitungen A1- A3 ausgegeben, alle übrigen Adressleitungen werden auf „1“ gesetzt.

  IntAck1 .. IntAck7 = FC0-FC2, A1-A3, /AS
- Interruptmaske wird automatisch auf die aktuelle Interruptpriorität gesetzt

Vektortabelle des MC68000

Vektor-nummer	Vektor-Offset	Ausnahme (Exception) -Vektor
$00	$000	Supervisor-Stackpointer bei RESET (SSP)
$01	$004	Supervisor Programmzähler bei RESET (PC)
$02	$008	Bus-Fehler
$03	$00C	Adreß-Fehler
$04	$010	Illegaler-Befehl
$05	$014	Ganzzahlige Division durch Null
$06	$018	CHK-Befehl
$07	$01C	TRAPV-Befehl
$08	$020	Privilegverletzung
$09	$024	Trace
$0A	$028	Nichtimplementierter OP-Code $Axxx
$0B	$02C	Nichtimplementierter OP-Code $Fxxx
$0C	$030	reserviert
$0D	$034	reserviert
$0E	$038	reserviert
$0F	$03C	Nicht initialisierter Interrupt
$10	$040	
bis	bis	reserviert
$17	$05C	
$18	$060	Falscher Interrupt
$19	$064	Autovektor (Ebene 1)
$1A	$068	Autovektor (Ebene 2)
$1B	$06C	Autovektor (Ebene 3)
$1C	$070	Autovektor (Ebene 4)
$1D	$074	Autovektor (Ebene 5)
$1E	$078	Autovektor (Ebene 6)
$1F	$07C	Autovektor (Ebene 7)
$20	$080	TRAP #0
bis	bis	bis
$2F	$0BC	TRAP #15
$30	$0C0	
bis	bis	reserviert
$3F	$0FC	
$40	$100	
...	...	
...	...	AnwenderVektoren
...	...	
$FF	$3FC	

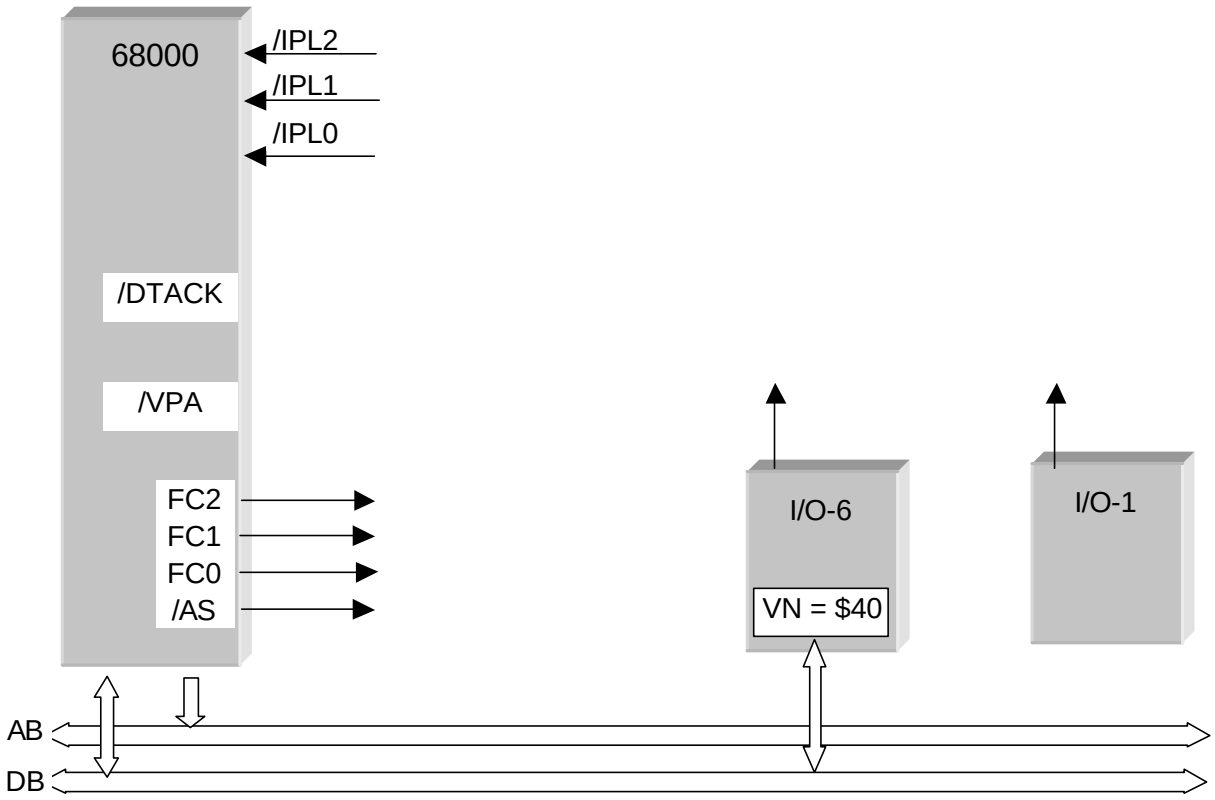
Vektortabelle des 68000

Der CPU-Hersteller hat viele mögliche Ausnahmen vorbereitet.
Jeder Ausnahme wurde eine Nummer zugeordnet (0..255). Diese Nummern verweisen auf Startadressen von Programmen (Ausnahmeprogramme), die die Reaktionen auf die Ausnahmen darstellen.

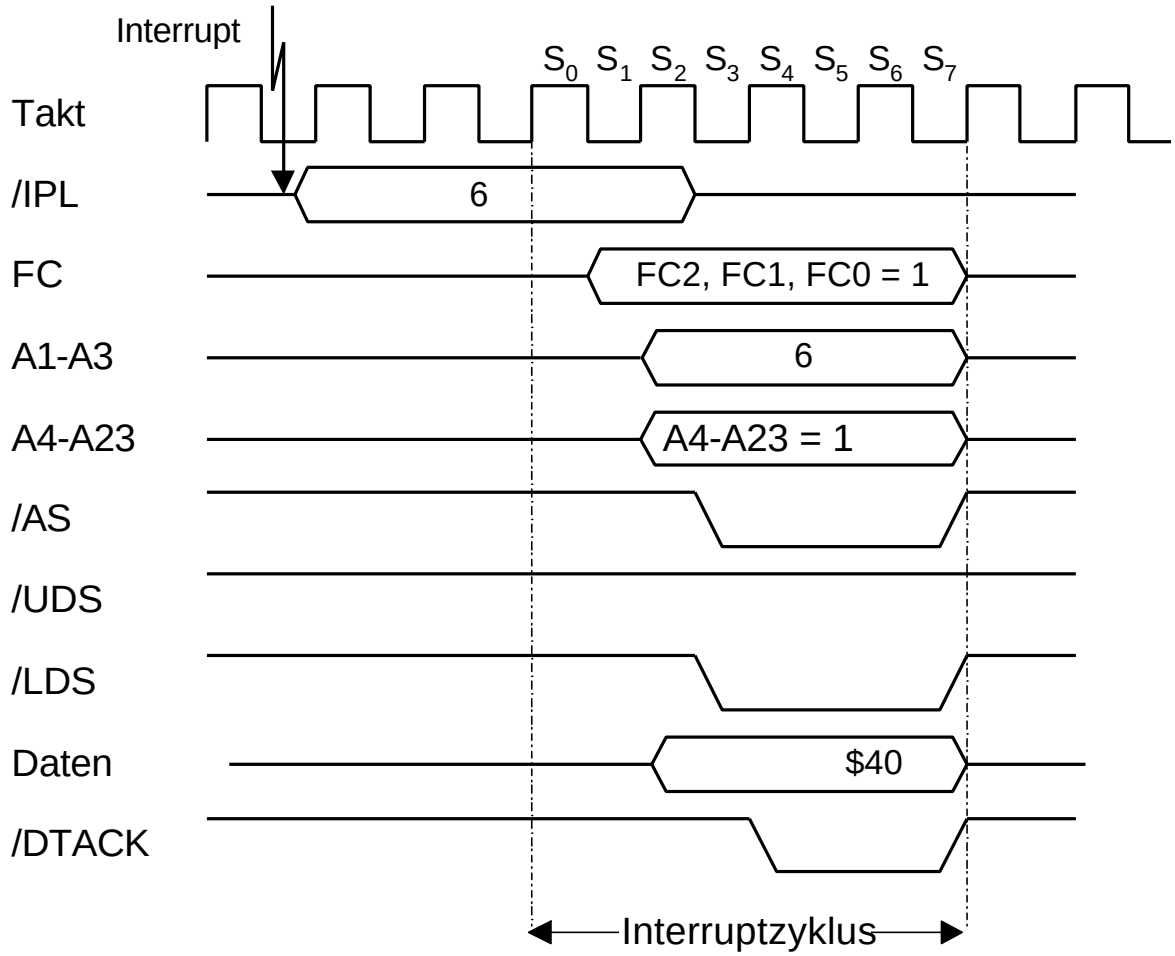
Vektor Nr.:

$00, $01	Nach dem Reset werden SSP und PC mit diesen Daten geladen.
$02	Die Busfehlerleitung wurde aktiviert
$03	Wortzugriff auf ungerade Adresse
$04	Nichtimplementierter Befehl.
$05	Division durch Null – Fehlermeldung vom Rechenwerk
$06	Befehl CHK Datenwort auf Grenzen prüfen
$07	Befehl TRAPV
$08	Privilegienverletzung – Befehle oder Zugriffe, die im Usermode nicht erlaubt sind
$09	Trace – Abhängig vom Tracebit führt CPU nach jedem Befehl eine Ausnahme aus
$0A	Befehlscode 1010 Emulator – Nachbilden der Befehle von Coprozessoren
$0B	Befehlscode 1111 Emulator – Nachbilden der Befehle einer FPU
$0C-$0E	reserviert
$0F	Nicht initialisierter Interrupt – im Vektorregister steht der Initialisierungswert
$10-$17	reserviert
$18	Falscher Interrupt – Zeitbedingungen für Interrupt wurden nicht eingehalten
$19-$1F	Autovektorinterrupt – werden von CPU in Abhängigkeit Interruptpriorität erzeugt
$20-$2F	TRAP-Befehle TRAP #1 - TRAP #15
$30-$3F	reserviert
$40-$FF	192 mögliche Anwender-Interruptvektoren

Blockschaltbild



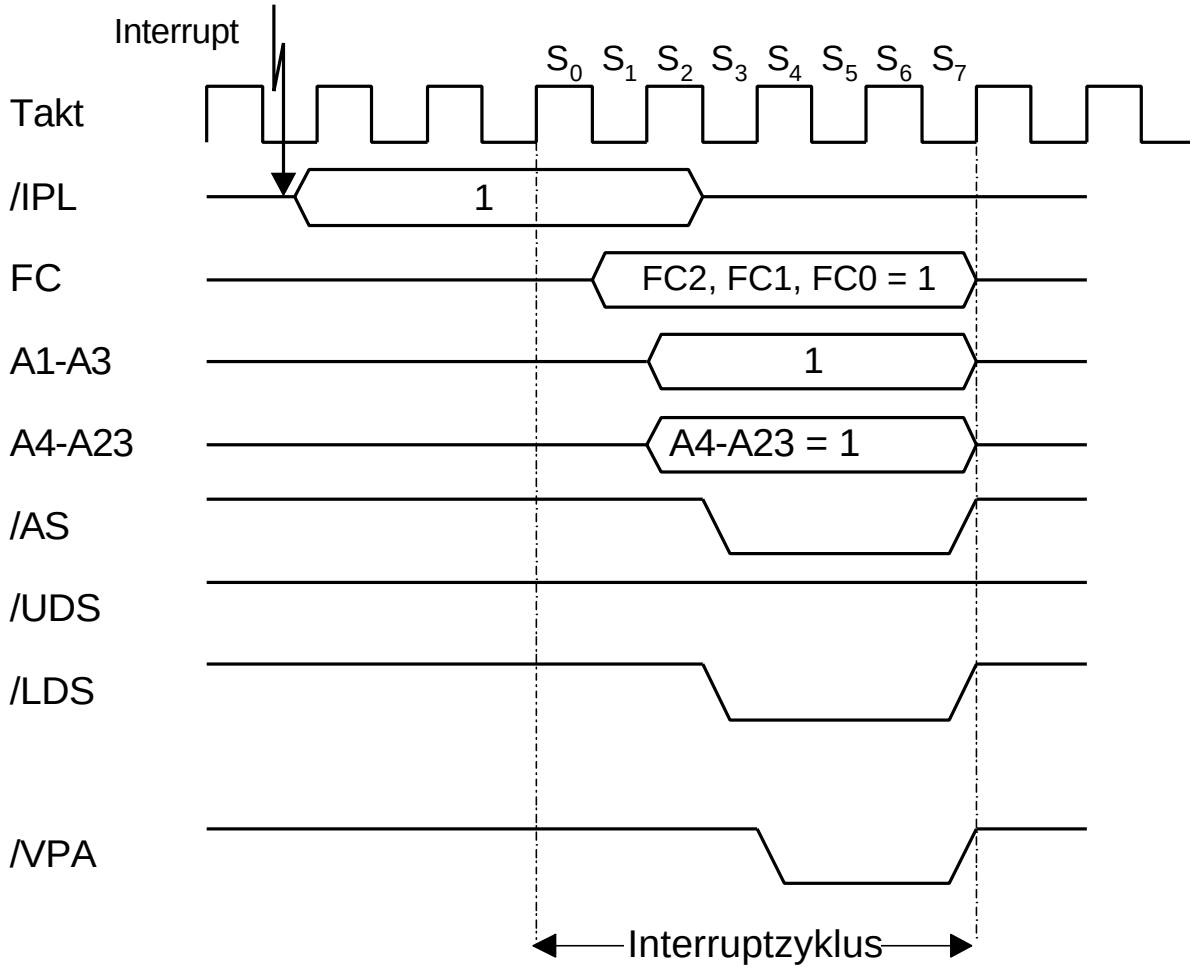
Zeitdiagramm für einen Interruptzyklus beim MC68000



Beispiel: Interruptlevel 6, Vektornummer $40

Der Buszyklus zum Einlesen der Vektornummer entspricht dem normalen Lesezyklus, allerdings wird statt einer üblichen Speicheradresse der vom Prozessor akzeptierte Interruptlevel auf den Adress-Leitungen A1-A3 angelegt, alle übrigen Adressleitungen haben den Wert „1“. Die Function-Code Leitungen sind alle auf „1“ gesetzt.

Zeitdiagramm für einen Interruptzyklus mit Autovektor



Der Baustein liefert keine Vektornummer. Statt dem Signal „/DTACK“ wird nun „/VPA“ aktiviert, worauf der Prozessor den Buszyklus beendet und den zum Interruptlevel gehörenden Autovektor verwendet.