TFH-Berlin

Mikrocomputer-Technik

MCT 49

Teil 6: Interrupt

Studiengang Technische Informatik (BA)
Prof. Dr.-Ing. Alfred Rożek

Inhalt

TFH-Berlin

- Anwendung
- Grundlagen der Interruptverarbeitung
  - Polling
  - Interrupts
  - Einteilung der Interrupts
- Interruptverarbeitung in 8086-Systemen
  - Ablauf der Interruptverarbeitung
  - Interrupt-Vektortabelle
  - Programmierung von Interrupt-Service-Routinen (ISR)
- Programmierbarer Interrupt-Controller 8259 (PIC)
  - Integration in ein Mikrocomputersystem
    - » Systemeinbindung
    - » Kaskadierung
    - » IRQ-Belegung im PC
  - Interne Struktur des Bausteins 8259
- Programmierung des PIC 8259
  - Initialisierungskommandos
  - Operationssteuerung
- Interrupt Verarbeitung beim ADNP

Anwendung

TFH-Berlin

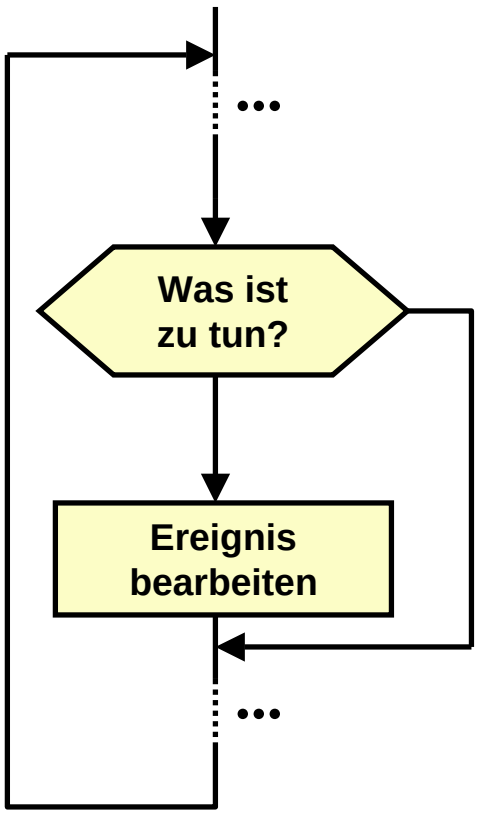
- Automatisierungsaufgaben
  - Regelkreise z.B. alle 2 ms oder 4 ms abarbeiten
  - Zeitscheibensysteme
- Erzeugung von zeitpräzisen Signalen (Timer)
- Treiben von Schnittstellen, externen Geräten
- Watch Dog Timer (NMI)
- Taskwechsel in Betriebssystemen
- Verwaltung von Hintergrundprogrammen
- Interaktive Oberflächen (Maus, Tastatur, ...)

Grundlagen der Interruptverarbeitung

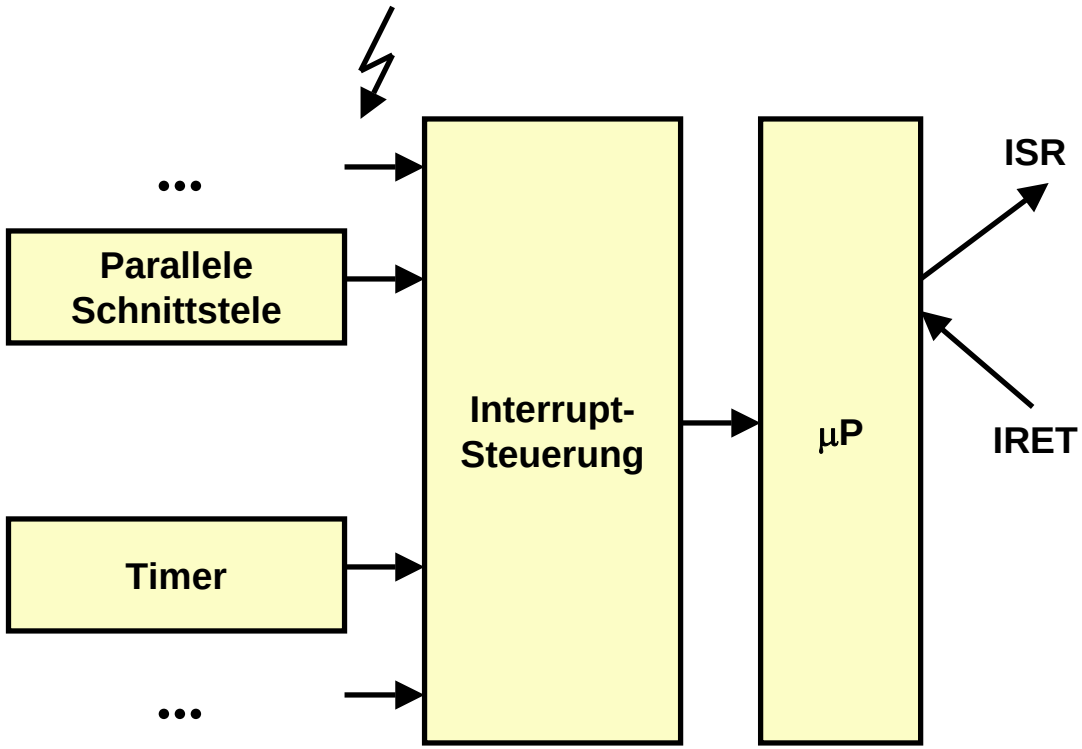
TFH-Berlin

Zwei Prinzipien zur Verarbeitung von Ereignissen:

Polling (Abfragemodus)



Interrupt (Unterbrechungsmodus)



Einteilung der Interrupts$_1$

TFH-Berlin

Ein Interruptaufruf unterbricht die gerade ablaufende Befehlsfolge in der Weise, daß nach Beendigung des laufenden Befehls nicht der nächste Befehl des Hauptprogramms ausgeführt, sondern eine ISR (Interrupt-Service-Routine) aufgerufen wird.
Nach Beendigung der ISR wird an der ursprünglichen Position im Programm fortgefahren.

Nicht „Intel-kompatible“ Prozessoren anderer Hersteller, beispielsweise Motorola, weisen im Prinzip die gleiche folgende Interrupt-Einteilung auf. Es werden aber andere Bezeichnungen benutzt, beispielsweise TRAP statt INT.

Einteilung der Interrupts$_2$

TFH-Berlin

Die Interrupts des 8086 können eingeteilt werden in:

- **SW-Interrupts (Interrupt-Befehle, auch interner Interrupt)**

  Werden vom Programmierer ausgelöst (ähnlich einem Unterprogrammaufruf).

  – INT 21H: DOS-Funktionsaufruf oder
  – INT 10H: BIOS-Funktionsaufruf
  – INT 3H: Sonderstellung, da keine INT-Nr. folgt

  SW-Interrupte werden synchron zur Programmausführung ausgelöst

- **Exceptions (Quelle: Prozessor selbst)**

  Werden in Ausnahmesituationen durch den Prozessor selbst ausgelöst.

  – z.B. Division durch Null: Auslösung von Interrupt Typ 0 oder
  – ungültiger OP-Code (0D6H oder 0F1H); kein Co-Prozessor vorhanden

  Auswirkungen wie bei SW-Interrupts

Einteilung der Interrupts$_3$

TFH-Berlin

- **HW-Interrupts (Quelle: Peripherie)**

  Externe asynchrone Interrupts. Diese Interrupte werden durch externe Bausteine beziehungsweise Ereignisse ausgelöst.

  Die CPU hat zwei Eingänge für asynchrone Interrupts:

  1. **NMI (Non Maskable Interrupt)**
     dient zur Mitteilung von „Katastrophenmeldungen“, z.B.
     - » bei einem Paritätsfehler im Speicher oder
     - » bei einer fehlerhaften Busarbitrierung oder
     - » bei Netzausfall

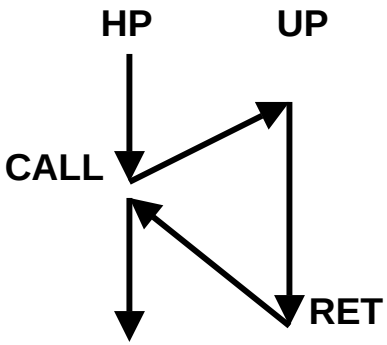
     Ein NMI hat eine höhere Priorität als die maskierbaren Interrupts

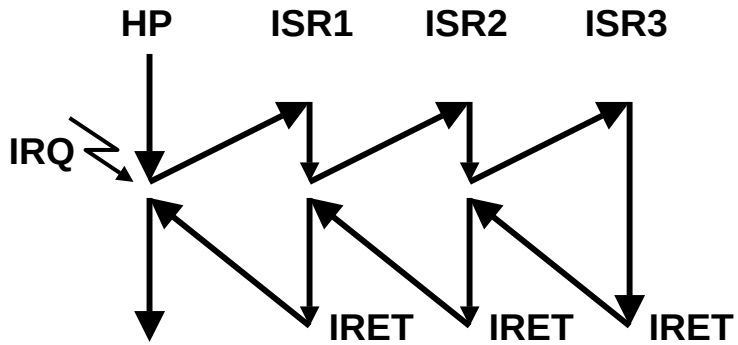
  2. **IRQ oder auch INTR (Interrupt-Request)**

     Vektorisierter maskierbarer Interrupt, gedacht für die Interruptanforderungen des normalen Betriebs
     - » Maskierbar durch das Interrupt-Flag (Assemblerbefehle CLI/STI)
     - » Die Behandlung dieses Interrupts wird durch einen gesonderten Controllerbaustein unterstützt, beispielsweise den Programmable-Interrupt-Controller (PIC 8259A)

UP contra ISR

TFH-Berlin





HP:	Hauptprogramm
UP:	Unterprogramm
CALL:	Unterprogrammaufruf
RET	Return from Subroutine

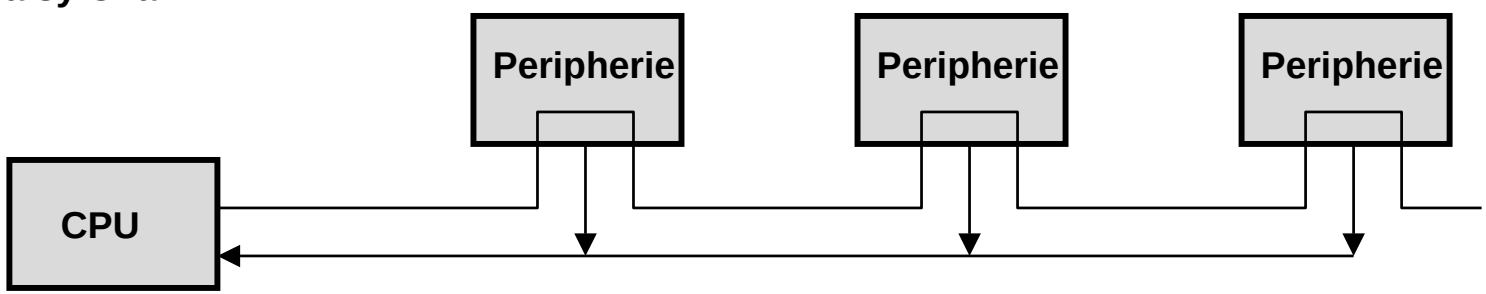
ISR:	Interrupt-Service-Routine
IRQ:	Interrupt Request
IRET	Return from Interrupt

INTERRUPT-Verfahren

TFH-Berlin

Nested Interrupt

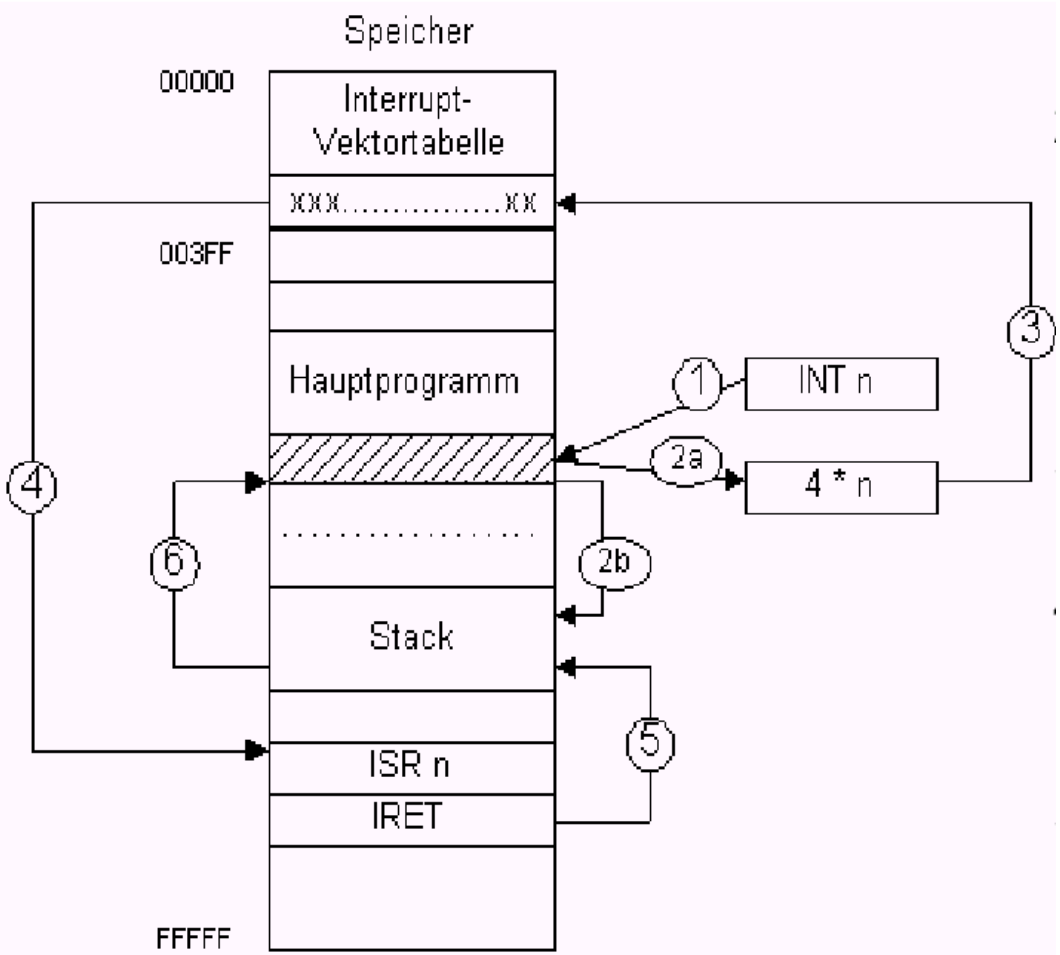
Daisy Chain



Abhängigkeit von Sammelleitung (geographische Lage)

Interrupt-Verarbeitung in 8086-Systemen

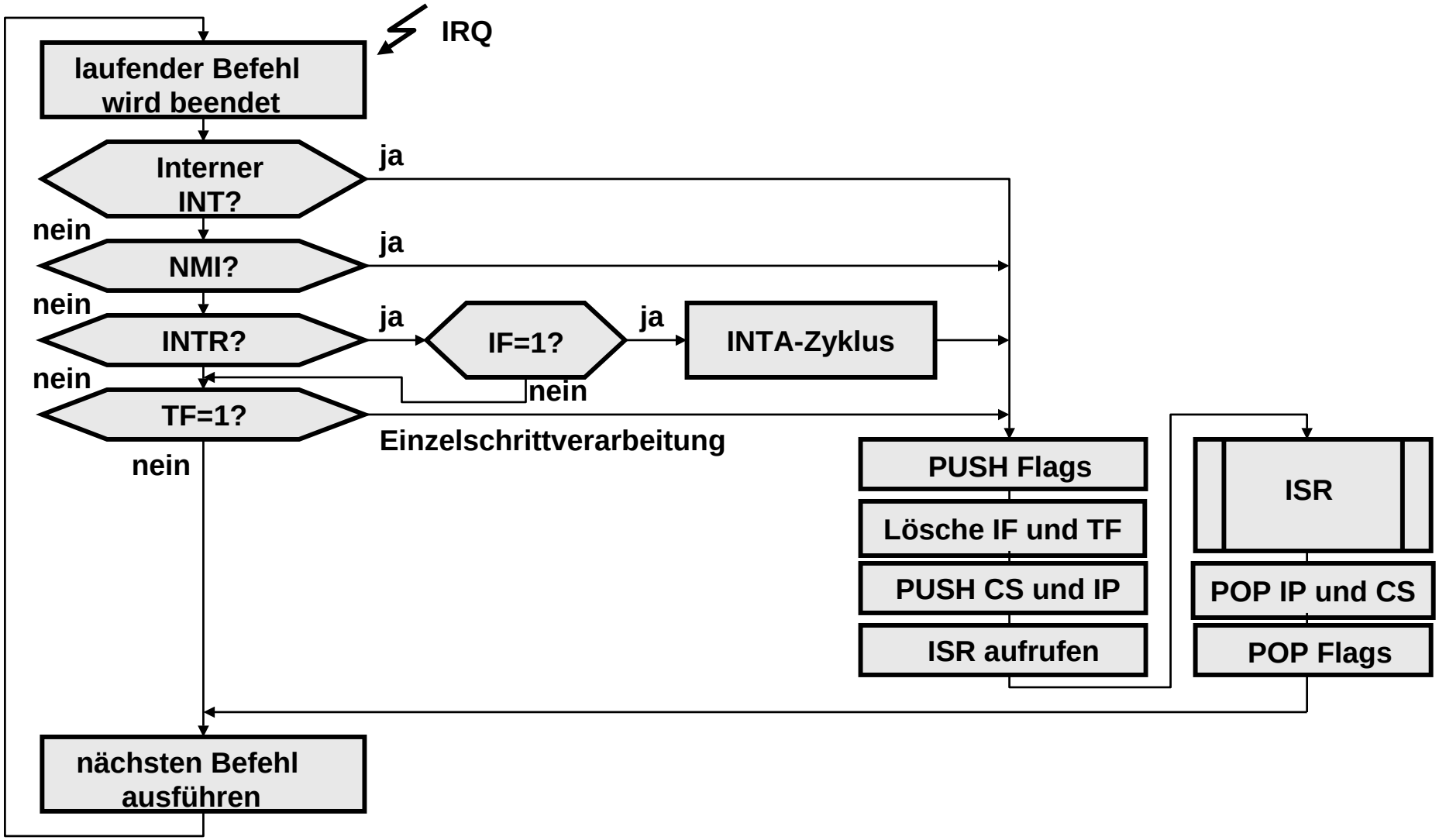
TFH-Berlin



1. HW- oder SW-Interrupt innerhalb eines Programmes ist aufgetreten, der gerade in Abarbeitung befindliche Befehl wird zu Ende ausgeführt
2. Interrupt-Nummer wird mit vier multipliziert (2a), Adresse des nächsten Programmbefehls des Hauptprogramms (Rücksprungadresse) und die Flags werden auf den Stack gesichert (2b)
3. Zugriff auf die Interrupt-Vektortabelle mit der in Schritt 2a berechneten Adresse
4. Interrupt-Vektortabelle enthält für alle Interrupts die zugehörigen Einsprungadressen (für Programm n: ISR n)
5. Am Ende des Interrupt-Programmes befindet sich ein IRET-Befehl (return from interrupt), IRET liest die Flags und die Rückkehradresse vom Stack
6. der Prozessor kann die Abarbeitung des Programms mit dem auf den Interruptbefehl folgenden Befehl fortsetzen

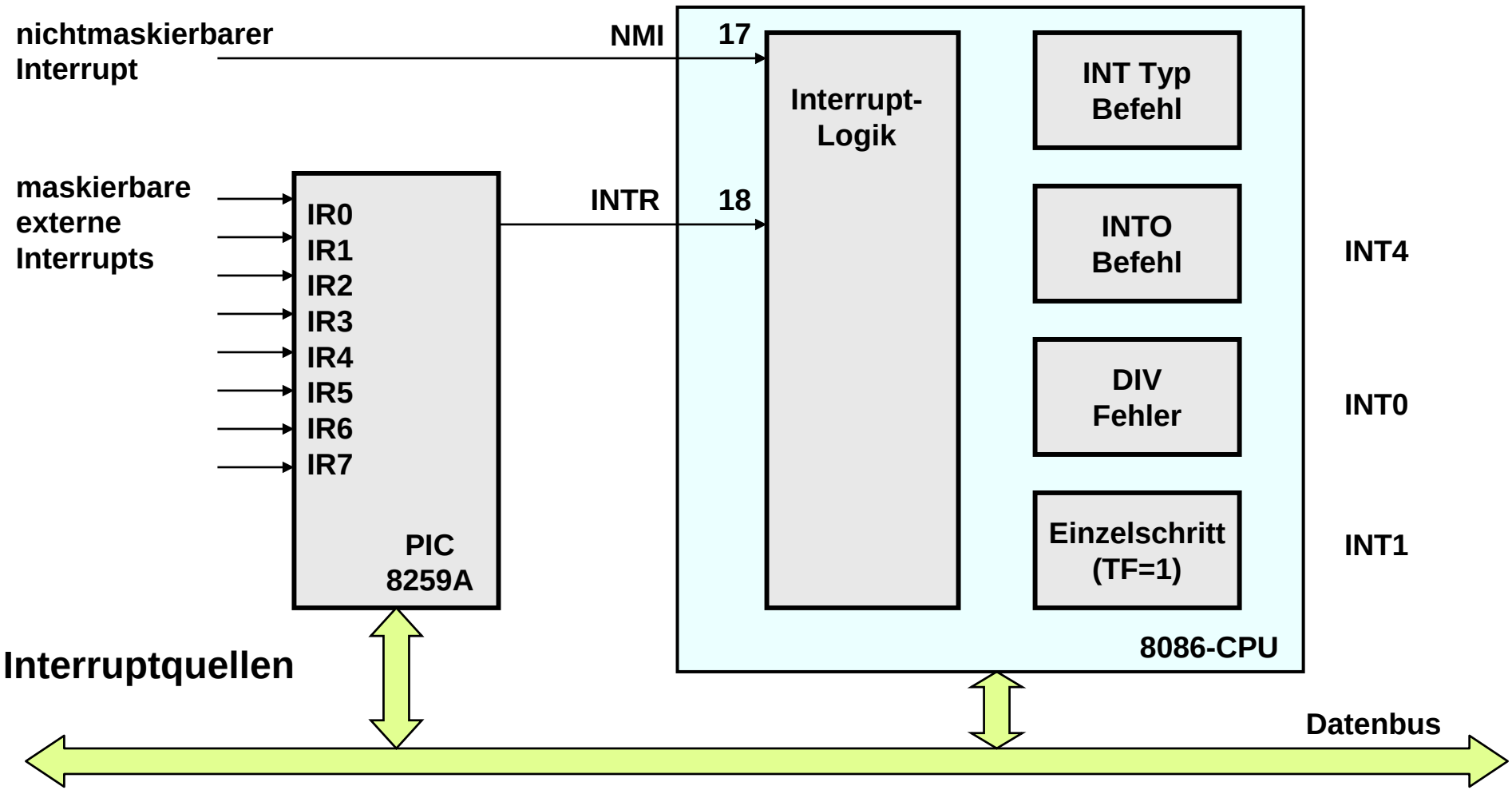
Interne Interruptverarbeitung (CPU: 8086)

TFH-Berlin



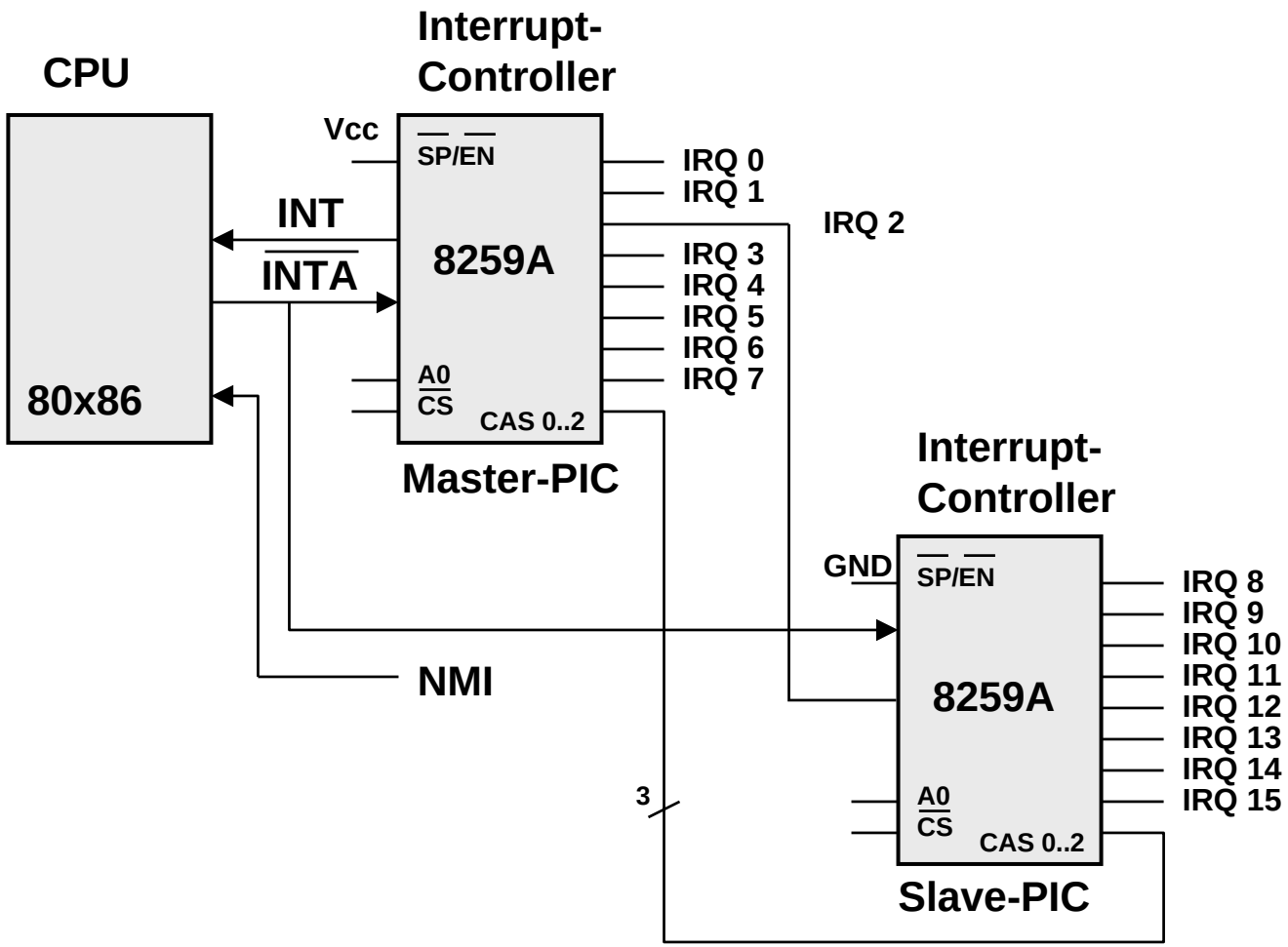
Interruptverarbeitung (CPU: 8086)

TFH-Berlin



Kaskadierung von Interrupt-Controllern

TFH-Berlin



Typische Interruptverteilung
in einem PC-AT (Advanced Technology)

TFH-Berlin

Kanal	Vektor	Anwendungen (bezogen auf AT-Technik)
NMI	02H	Nicht maskierbarer Interrupt (Parität, Fehler auf Erweiterungskarten, Speicherauffrischung)
IRQ 0	08H	System-Zeitgeber (Kanal 0 des Timers 8253/54)
IRQ 1	09H	Tastatur
IRQ 2	0AH	Kaskadierung mit Interrupt 9 (vom Slave PIC)
IRQ 3	0BH	COM Port 2
IRQ 4	0CH	COM Port 1
IRQ 5	0DH	Paralleler Drucker LPT 2 (auch: Standard-Belegung durch Soundkarte (SB-Pro kompatibel))
IRQ 6	0EH	Diskettenlaufwerks-Controller
IRQ 7	0FH	Paralleler Drucker LPT 1
IRQ 8	70H	Echtzeituhr
IRQ 9	71H	Redirection zu IRQ 2 (ein IRQ 9 führt zu einem Aufruf des Interrupt-Handlers von IRQ 2)
IRQ 10	72H	reserviert bzw. frei (auch: PCI-Graphickarte)
IRQ 11	73H	reserviert bzw. frei (auch: PCI-Graphickarte)
IRQ 12	74H	reserviert bzw. frei (auch: PS/2 Maus)
IRQ 13	75H	Mathematischer Coprozessor (z.B.: 80x87)
IRQ 14	76H	Erste Enhanced IDE-Schnittstelle (gewöhnlich Festplatte)
IRQ 15	77H	Zweite Enhanced IDE-Schnittstelle (gewöhnlich CD-ROM)

Interruptvektortabelle eines DOS-Systems

TFH-Berlin

Nummer		Adresse	Segment:Offset	Anwendung
0	00H	0000H	407BH:00CEH	Prozessor: Division durch Null
1	01H	0004H	1352H:145CH	Prozessor: Einzelschrittsteuerung (Trace)
2	02H	0008H	0DDFH:0016H	Nicht maskierbarer Interrupt: Systemfehler
3	03H	000CH	1352H:1465H	Haltepunkt durch Befehlscode CCH (Breakp.)
4	04H	0010H	0070H:075CH	Befehl INTO (Interrupt bei Overflow)
5	05H	0014H	F000H:FF54H	BIOS: Hardcopy durch Druck-Taste
6	06H	0018H	F000H:F856H	Prozessor: (80286: unbekannter Code)
7	07H	001CH	F000H:F856H	Prozessor: (ab 80286: Speicherschutz)
..	...	...	...	...
16	10H	0040H	1352H:1711H	BIOS: Aufruf der Bildschirmfunktionen
..	...	...	...	...
22	16H	0058H	F000H:E82EH	BIOS: Tastatur- und Druckerfunktionen
..	...	...	...	...
28	1CH	0070H	F000H:FF53H	BIOS: Timer-Interrupt (18.2 mal pro Sek.)
..	...	...	...	...
33	21H	0084H	1352H:171FH	DOS: Aufruf der Betriebssystemfunktionen

Typische Interruptverteilung in einem PC

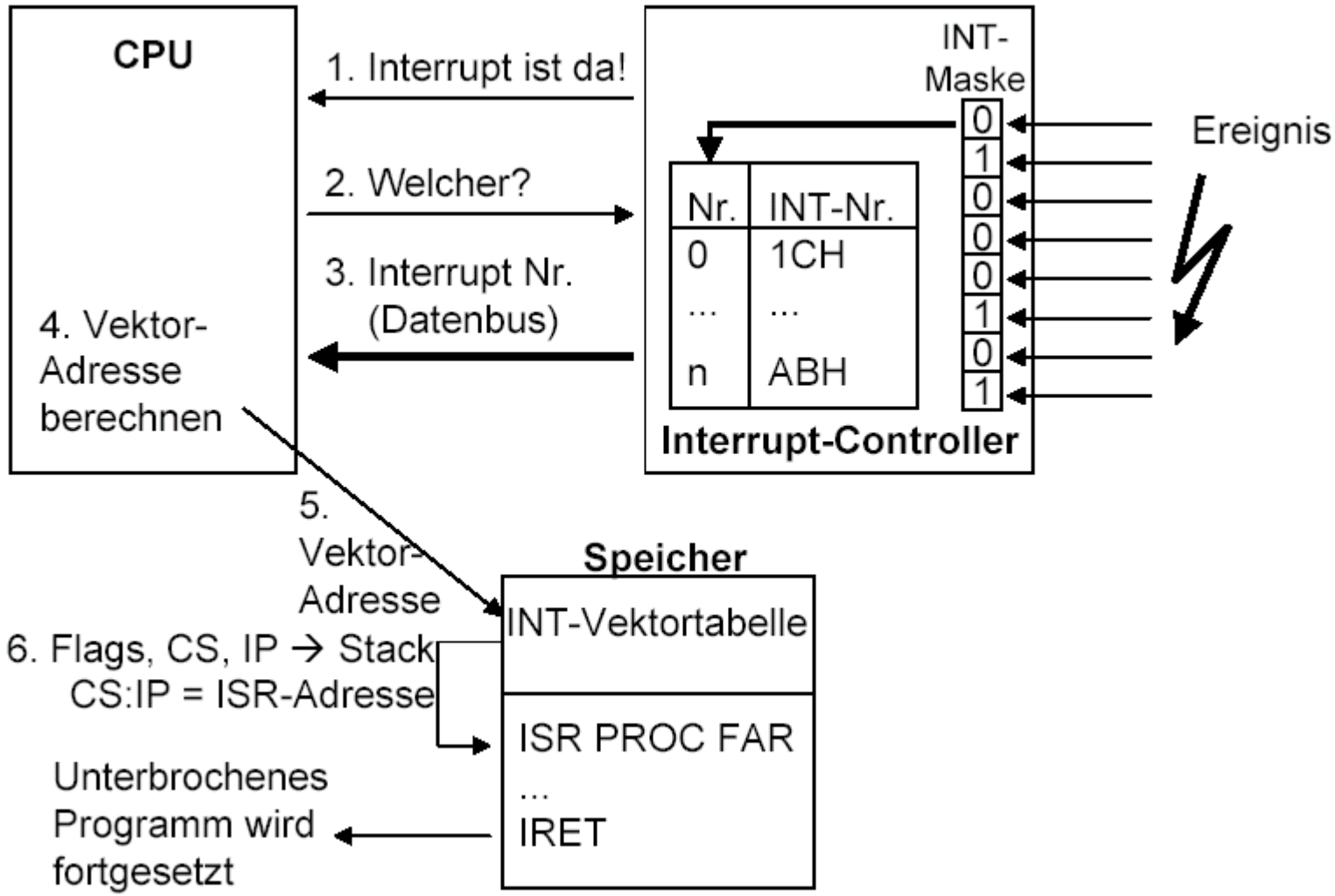
TFH-Berlin

IRQ 0	Systemzeitgeber	OK
IRQ 1	Standard (101/102 Tasten) oder Microsoft Natural Keyboard	OK
IRQ 2	Programmierbarer Interrupt-Controller	OK
IRQ 3	COM-Anschluss (COM2)	OK
IRQ 4	COM-Anschluss (COM1)	OK
IRQ 5	ACPI IRQ-Holder für PCI-Steuerung	OK
IRQ 5	Creative AudioPCI (ES1371,ES1373) (WDM)	OK
IRQ 5	AVM ISDN-Controller FRITZ!Card PCI	OK
IRQ 6	Standard-Diskettenlaufwerk-Controller	OK
IRQ 7	Druckeranschluss (LPT1)	OK
IRQ 8	CMOS-/Echtzeitsystemuhr	OK
IRQ 9	SCI IRQ belegt von ACPI-Bus	OK
IRQ 10	ACPI IRQ-Holder für PCI-Steuerung	OK
IRQ 10	VIA Tech 3038 PCI-zu-USB Universeller Host Controller	OK
IRQ 10	VIA Tech 3038 PCI-zu-USB Universeller Host Controller	OK
IRQ 10	National Semiconductor Corp. DP83815 10/100 MacPhyter3v PCI Adapter	OK
IRQ 11	ACPI IRQ-Holder für PCI-Steuerung	OK
IRQ 11	NVIDIA GeForce2 MX	OK
IRQ 13	Numerischer Coprozessor	OK
IRQ 14	VIA Bus Master PCI IDE Controller	OK
IRQ 14	Erster IDE Controller (Dual FIFO)	OK
IRQ 15	VIA Bus Master PCI IDE Controller	OK
IRQ 15	Zweiter IDE Controller (Dual FIFO)	OK

Systemübersicht | Hardwareressourcen | IRQs

Schematischer Ablauf eines externen Interrupts

TFH-Berlin



Programmierung von Interrupt Service Routinen

Verwendung von DOS-Funktionen für den Eintrag der ISR in die Interrupt-Vektor-Tabelle:

```
#include <dos.h>
// Deklaration der globalen Variablen
//-------------------------------------------------------------
void interrupt ( *AlterIntVektor) (void);// Merker alter Vektor
//=============================================================
//      Interrupt-Service-Routine
//=============================================================
void interrupt IntService(void) {
       {...............};    // Das Interrupt-Programm
}
//=============================================================
// main: Vordergrundprogramm
//=============================================================
void main() {
    AlterIntVektor = getvect(0x1C); // Alten Vektor sichern
    setvect(0x1C,IntService); // Eigene ISR eintragen
       {...............}
    setvect(0x1C,AlterIntVektor);// Alten Interrupt-Vektor restaur.
}
```

ISR in C programmieren - ohne Betriebssystemaufrufe

TFH-Berlin

```
// Deklaration der globalen Variablen und der Konstanten
//-------------------------------------------------------------
unsigned long int far AlterIntVektor; //Speichert alten INT-Vektor
//Die Adresse des Interrupts 1CH ist 70H:
long int far *Vektoreintrag = (long int far *) 0x70; //Zeiger V-Tab.
//=============================================================
//      Interrupt-Service-Routine
//=============================================================
void interrupt IntService(void) {
       {.............};     // Das Interrupt-Service-Programm
}
//=============================================================
//      main: Vordergrundprogramm
//=============================================================
void main() {
   AlterIntVektor = *Vektoreintrag;   // Alten Vektor sichern
   *Vektoreintrag = (long int far) &IntService; //Eigene ISR eintr.
   {.....................}
   *Vektoreintrag = AlterIntVektor;   // Alten Inter-Vektor restaur.
}
```

Beispiel: Interrupt-Service-Routine in ASM

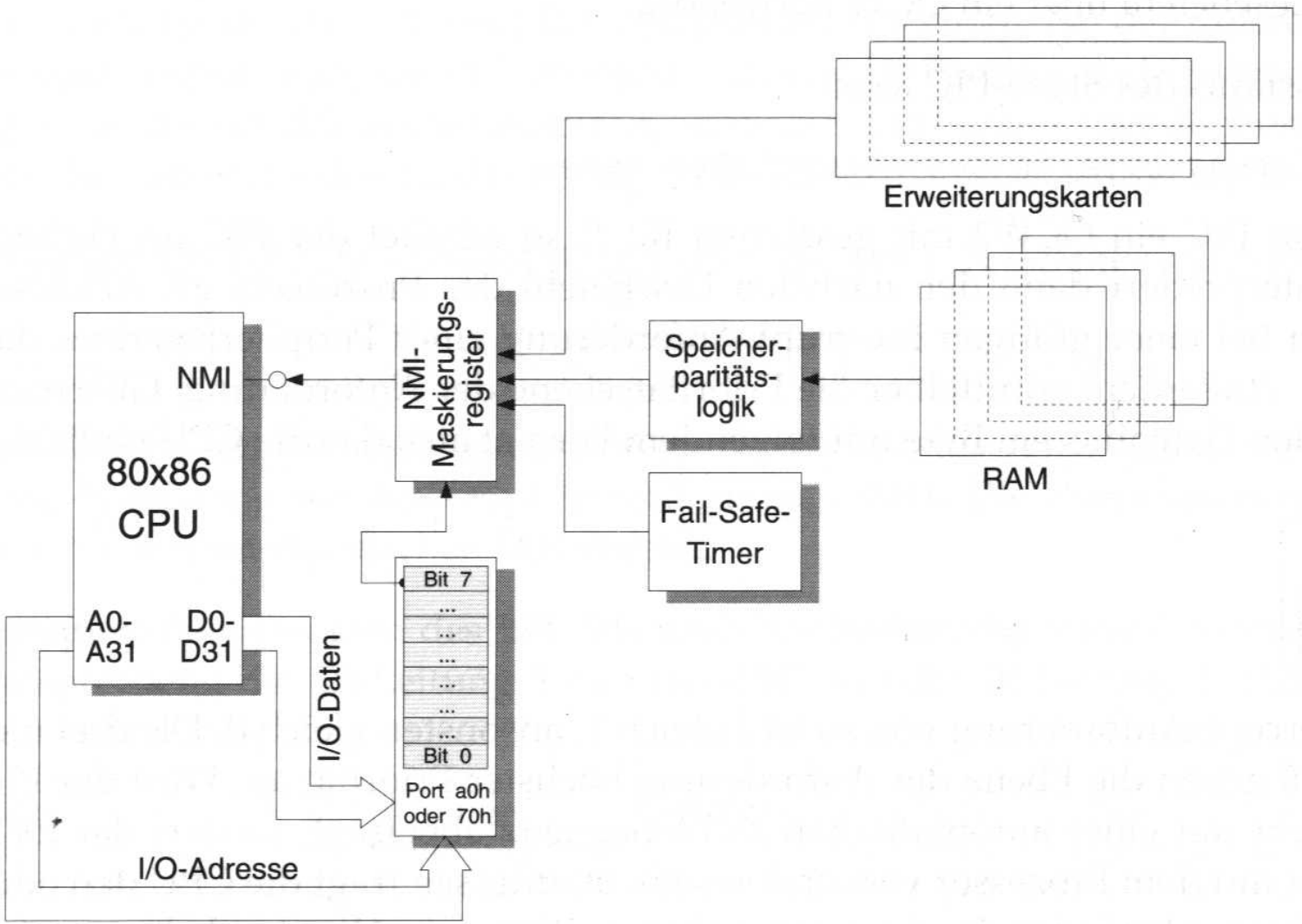
TFH-Berlin

```
    .CODE
    MOV  AX,@data          ;Beginn des Hauptprogramms
    MOV  DS,AX
; Interrupt-Vektor in Tabelle eintragen
    MOV  AX,0              ;Int.-Vektor-Tabelle beginnt mit Adr. 0
    MOV  ES,AX             ;ES zeigt auf Beginn Vektor-Tabelle
    MOV  AX,@CODE          ;Beginn des Code-Segments der ISR
    MOV  BX,OFFSET ISR     ;IP der Interrupt-Service-Routine
    CLI                    ;Interrupts sperren
    MOV  ES:1CH*4+2,AX     ;Vektor in Interrupt-Vektor-Tabelle
    MOV  ES:1CH*4,BX       ;    eintragen (z.B. 1C = INT-Nummer)
    STI                    ;Interrupts freigeben
<<<< das eigentliche Hauptprogramm >>>>

; Interrupt-Service-Routine
;==========================
ISR PROC FAR
    PUSH AX                ;Alle in ISR benutzten Register sichern
    PUSH DS
    MOV  AX,@DATA          ;Aufsetzen, da DOS beispielsweise bei der
    MOV  DS,AX             ; Zeichenausgabe DS veraendert
<<<< die eigentliche Interrupt-Service-Routine >>>>
    POP DS                 ;Register restaurieren
    POP AX
    IRET                   ;Return from Interrupt
ISR ENDP
```

NMI und NMI-Maskierungsregister

TFH-Berlin



C-Programmbeispiel zum Maskieren und Aktivieren des NMI im PC-AT

TFH-Berlin

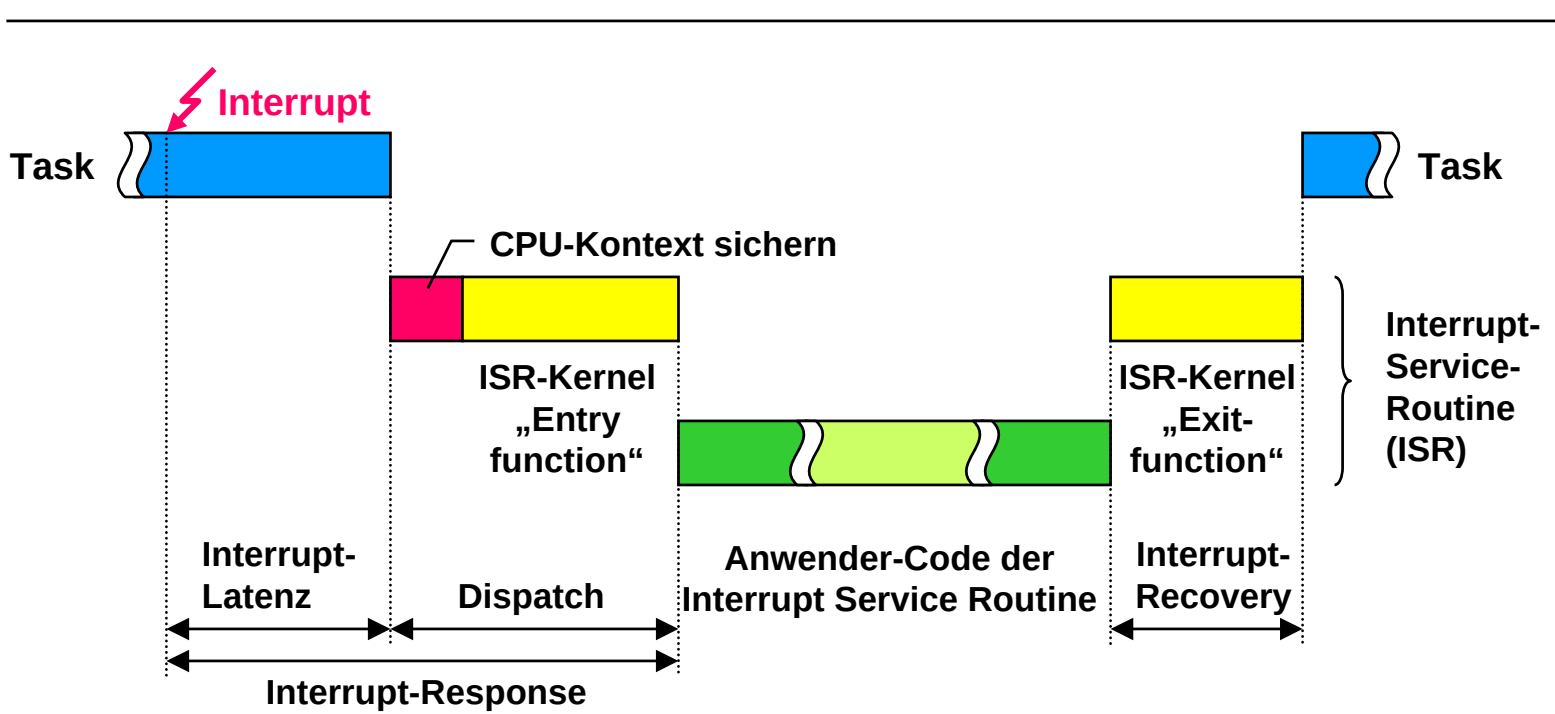
```
/* nmi.c */
main(argc, argv)
int argc;
char *argv[];

{ int i;

  if (strcmpi(argv[1], „mask“) != 0) {                    /* prüfen, ob maskieren */
    if (strcmpi(argv[1], „unmask“) != 0) {                /* prüfen, ob aktivieren */
      printf(„\n\nUngültiges Argument %s“, argv[1]);      /* kein gültiges Argument */
      exit(1);                                            /* Exit mit ERRORLEVEL 1 */
    }
    else {                                    /* NMI aktivieren */
      i = inp(0x70);                          /* Byte des Port 70h lesen */
      i = i & 0x7f;                           /* Bit 7 löschen */
      outp(0x70, i);                          /* Byte in Port 70h zurückschreiben */
      printf(„\n\nNMI aktiviert !!\n\n“);      /* Nachricht ausgeben */
    }
  }
  else {                                      /* NMI maskieren */
    i = inp(0x70);                            /* Byte des Port 70h lesen */
    i = i | 0x80;                             /* Bit 7 setzen */
    outp(0x70, i) ;                           /* Byte in Port 70h zurückschreiben */
    printf(„\n\nNMI maskiert !!\n\n“);         /* Nachricht ausgeben */
  }
  exit(0);                                    /* Exit mit ERRORLEVEL 1 */
}
```

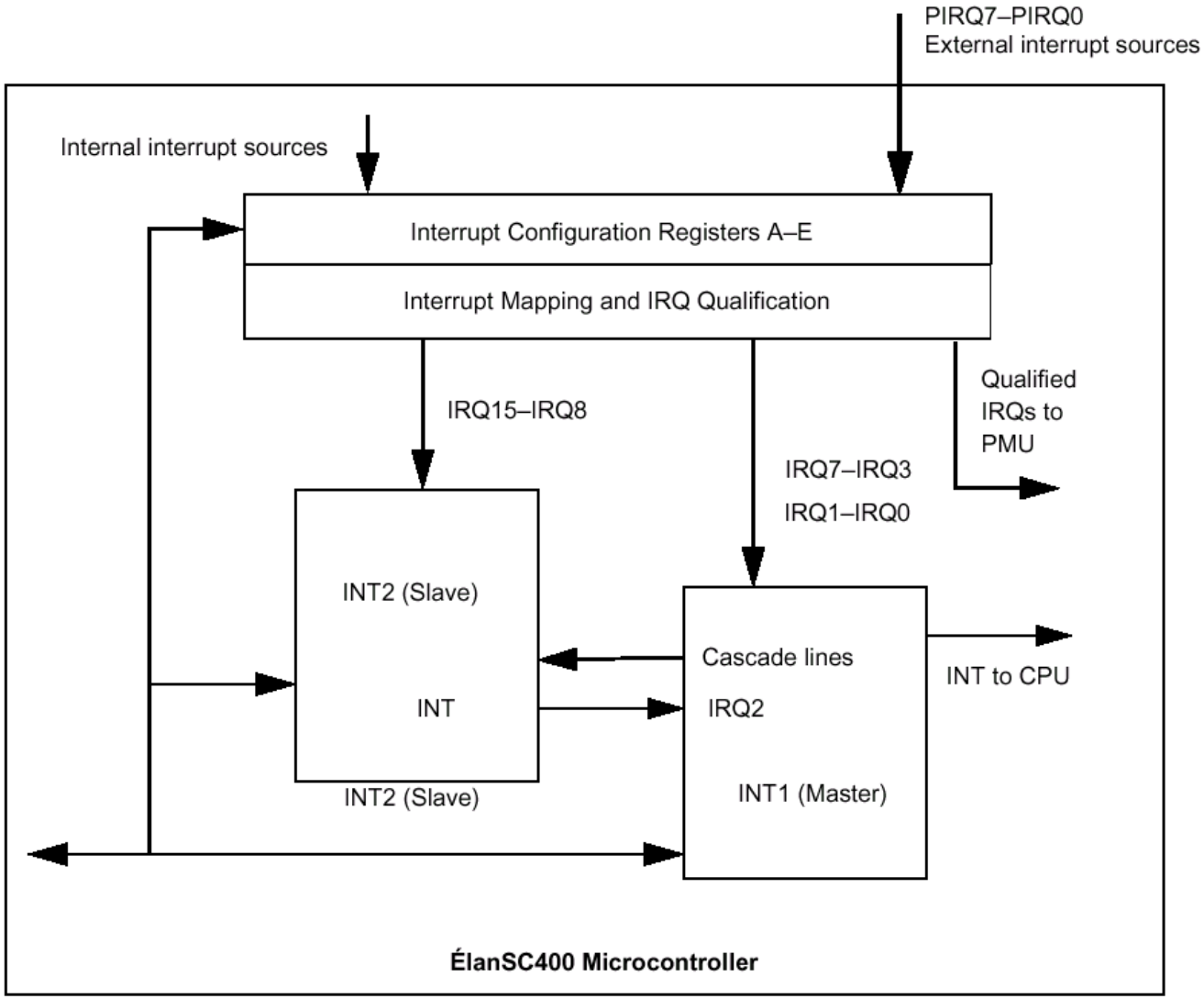
Interrupts brauchen Zeit

TFH-Berlin



Systemverhalten und -zeiten bei einem Kontextwechsel

Programmable Interrupt Controller (SC410)



Quelle: CDROM-SSV; SC400UM.PDF, S. 11.3

IRQ-Mapping (SC410)

TFH-Berlin

Microcontroller Resource	IRQ Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Programmable Interval Timer	X															
Matrix Keyboard Controller (Keyboard Output Buffer Full)		X														
XT Keyboard Controller (Shift Buffer Full)		X														
UART (8-Pin Serial and Infrared Ports)				X	X											
Parallel Port						X		X								
Real-Time Clock									X							
Graphics Controller (Cursor Control Address Register Accesses)										X						
Matrix Keyboard Controller (Mouse Output Buffer Full)													X			
PC Card (Sockets A and B):[1] —I/O and Memory mode ($\overline{\text{IREQx}}$ pin) —Memory-only mode (status change/RI)				X	X	X		X		X	X	X	X		X	X
PIRQ0 Programmable IRQ I/O Pin		X		X	X	X	X	X	X	X	X	X	X	X	X	X
PIRQ1 Programmable IRQ I/O Pin		X		X	X	X	X	X	X	X	X	X	X	X	X	X
PIRQ2 Programmable IRQ I/O Pin		X		X	X	X	X	X	X	X	X	X	X	X	X	X
PIRQ3 Programmable IRQ I/O Pin		X		X	X	X	X	X	X	X	X	X	X	X	X	X
PIRQ4 Programmable IRQ I/O Pin		X		X	X	X	X	X	X	X	X	X	X	X	X	X
PIRQ5 Programmable IRQ I/O Pin		X		X	X	X	X	X	X	X	X	X	X	X	X	X
PIRQ6 Programmable IRQ I/O Pin		X		X	X	X	X	X	X	X	X	X	X	X	X	X
PIRQ7 Programmable IRQ I/O Pin		X		X	X	X	X	X	X	X	X	X	X	X	X	X

Quelle: CDROM-SSV; SC400UM.PDF, S. 11.4

Interrupt-Vektoren (SC410)

TFH-Berlin

IRQ	Vector Value	
IRQ0	8h	**ADNP**
IRQ1	9h	
IRQ2	Not available (used for cascading)	
IRQ3	Bh	**INT1**
IRQ4	Ch	
IRQ5	Dh	
IRQ6	Eh	**INT4**
IRQ7	Fh	**INT5**
IRQ8	70h	
IRQ9	71h	
IRQ10	72h	**INT3**
IRQ11	73h	
IRQ12	74h	**INT2**
IRQ13	75h	
IRQ14	76h	
IRQ15	77h	

Quelle: CDROM-SSV; SC400UM.PDF, S. 11.5

Interruptprogrammierung

TFH-Berlin

* PC/AT-Architekturen besitzen zwei Interrupt Controller (PIC1, PIC2). Über diese Controller können externe Bausteine Interrupts erzeugen.

Int.	Funktion	PIC
IRQ0	PIT	PIC1
IRQ1	Unbenutzt	PIC1
IRQ2	Verbindung zum PIC2	PIC1
IRQ3	Externer Interrupteingang INT1 (Default)	PIC1
IRQ4	COM1	PIC1
IRQ5	LAN	PIC1
IRQ6	Externer Interrupteingang INT4 (Default)	PIC1
IRQ7	Externer Interrupteingang INT5 (Default)	PIC1
IRQ8	RTC	PIC2
IRQ9	Unbenutzt	PIC2
IRQ10	Externer Interrupteingang INT3 (Default)	PIC2
IRQ11	Unbenutzt	PIC2
IRQ12	Externer Interrupteingang INT2 (Default)	PIC2
IRQ13	Unbenutzt	PIC2
IRQ14	Unbenutzt	PIC2
IRQ15	Unbenutzt	PIC2

Interruptprogrammierung

TFH-Berlin

* Der AMD SC410 besitzt eine recht aufwendige interne Schaltung, um eine möglichst flexible Interrupt-Zuordnung zu ermöglichen.

DNP-Int.	SC410-Int.	SC410-Register
INT1	PIRQ3	Interrupt Configuration Register B
INT2	PIRQ4	Interrupt Configuration Register C
INT3	PIRQ5	Interrupt Configuration Register C
INT4	PIRQ6	Interrupt Configuration Register D
INT5	PIRQ7	Interrupt Configuration Register D

* Für den DIL/NetPC gilt: Es gibt fünf externe Interrupteingänge (INT1 bis INT5).

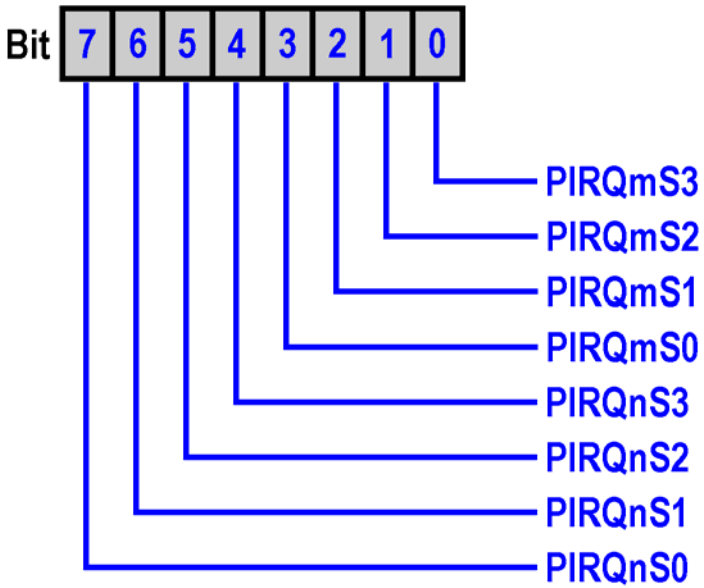
* Diese fünf Eingänge sind SC410-intern den fünf „PIRQs“ (Programmable Interrupt Request) PIRQ3 bis PIRQ7 zugeordnet.

* Jeder PIRQ kann über ein „Interrupt Configuration Register“ mit einem freien IRQ verknüpft werden.

Interruptprogrammierung

TFH-Berlin

* Beispiel für ein „**Interrupt Configuration Register**“




```
// DIL/NetPC-Interrupt-Zuodnung: Verbinde den Interrupt-
// Eingang INT1 (SC410-PIRQ3) mit IRQ14.

windex (0xd5, (rindex (0xd5)& 0x0f) | 0xe0);
```

Interruptprozeduren

TFH-Berlin

```
;Initialisierung von ISRs (Interrupt-Service-Routinen)

INTVEKTOR   SEGMENT     AT 0        ;Initialisieren des Vektors
                        ORG 2*4     ;für Typ2 (NMI) mit der Adresse
            DD          I_NMI       ;der NMI-Prozedur
                        ORG 33*4    ;Initialisieren des Vektors für Typ33
            DD          IT33        ;mit der Adresse der ISR IT33
INT_VEKTOR              ENDS

STACK                   SEGMENT     STACK
                        ...
STACK                   ENDS
DATA                    SEGMENT
                        ...
DATA                    ENDS
CODE                    SEGMENT
                        ...         ;Hauptprogramm
CODE                    ENDS

INT PROC    SEGMENT
            I_NMI       PROC FAR
                        ...         ;Im Allg. kein Rücksprung aus einer NMI-Prozedur
            I_NMI       ENDP

            IT33        PROC FAR
                        PUSH AX
                        PUSH BX
                        ...         ;Eigentliche ISR
                        POP BX
                        POP AX
                        STI
                        IRET
            IT33        ENDP
INT PROC    ENDS
```

Assemblerbeispiel

Interrupt Mapping beim DIL/NetPC

TFH-Berlin

IRQ	IRQ-line	Usage
IRQ 0	NA	PIT (onboard)
IRQ 1	none	free
IRQ 2	NA	NA
IRQ 3	INT1	can be remapped by user
IRQ 4	NA	COM1 (onboard)
IRQ 5	NA	LAN (onboard)
IRQ 6	INT4	can be remapped by user
IRQ 7	INT5	can be remapped by user
IRQ 8	NA	RTC (internal)
IRQ 9	none	free
IRQ 10	INT3	can be remapped by user
IRQ 11	none	free
IRQ 12	INT2	can be remapped by user
IRQ 13	none	free
IRQ 14	none	free
IRQ 15	none	free

NA = not available

ADNP Pin	ADNP Line	Elan PIRQ	Index register
86	7	1	0xD4 (higher nibble)
85	6	0	0xD4 (lower nibble)
44	1	3	0xD5 (higher nibble)
43	2	4	0xD6 (lower nibble)
42	3	5	0xD6 (higher nibble)
41	4	6	0xD7 (lower nibble)
40	5	7	0xD7 (higher nibble)

Der Interruptcontroller des SC410 löst einen Interrupt aus, wenn an einem der möglichen PIRQs eine steigende Signalflanke erkannt wird.
(siehe Kapitel 12 des unten genannten Dokuments)

Soll beispielsweise über die INT1-Leitung anstelle von IRQ3 (default) ein IRQ14 ausgelöst werden, dann erreicht man das Mapping durch folgende Codezeile:

```
windex(0xD5,(rindex(0xD5)&0x0F) | 0xE0);      // maps PIRQ3 (INT1) to IRQ14
```

Quelle: SSV-CD-ROM V1.14: ADNP-HW-DOC.PDF

DIL/NetPC DNP/1486-3V I/O Address Mapping$_1$

TFH-Berlin

I/O Addr. Range	*DNP/1486-3V Usage*	*PC/AT Standard*
000h - 00Fh	8237 DMA Controller #1	8237 DMA Controller #1
020h - 021h	8259 Master Interrupt Controller	8259 Master Interrupt Controller
022h - 023h	SC410 CSCIR, CSCDR (Index and Data)	---
040h - 043h	8253 Programmable Timer	8253 Programmable Timer
060h - 06Fh	Port 61h	8042 Keyboard Controller
070h - 07Fh	RTC, NMI Mask Register	RTC, NMI Mask Register
080h - 09Fh	DMA Page Registers	DMA Page Registers
0A0h - 0B1h	8259 Slave Interrupt Controller	8259 Slave Interrupt Controller
0C0h - 0DFh	8237 DMA Controller #2	8237 DMA Controller #2
0F0h - 0F1h	SC410	Math Coprocessor
0F8h - 0FFh	SC410	Math Coprocessor
170h - 177h	Unused	Hard Disk Controller #2
1F0h - 1F8h	Unused	Hard Disk Controller #1
200h - 207h	Unused	Game Port
238h - 23Bh	Unused	Bus Mouse
23Ch - 23Fh	Unused	Alt. Bus Mouse
240h - 277h	Unused	Unused
278h - 27Fh	Unused	Parallel Printer
280h - 2AFh	Unused	Unused
2B0h - 2BFh	Unused	EGA
2C0h - 2CFh	Unused	EGA

DIL/NetPC DNP/1486-3V I/O Address Mapping$_2$

TFH-Berlin

I/O Addr. Range	*DNP/1486-3V Usage*	*PC/AT Standard*
2D0h - 2DFh	Unused	EGA
2E0h - 2E7h	Unused	GPIB
2E8h - 2EFh	Unused	Serial Port
2F8h - 2FFh	Unused	Serial Port
300h - 30Fh	On-board LAN Controller	Prototype Card
310h - 31Fh	On-board LAN Controller	Prototype Card
320h - 32Fh	Unused	Hard Disk Controller XT
330h - 33Fh	Unused	Unused
340h - 36Fh	Unused	Unused
370h - 377h	Unused	Floppy Disk Controller #2
378h - 37Fh	Unused	Parallel Printer
380h - 38Fh	Unused	SDLC Adapter
390h - 39Fh	Unused	Unused
3A0h - 3AFh	Unused	SDLC Adapter
3B0h - 3BBh	Unused	MDA Adapter
3BCh - 3BFh	Unused	Parallel Printer
3C0h - 3CFh	Unused	VGA/EGA Adapter
3D0h - 3DFh	Unused	CGA Adapter
3E8h - 3EFh	Unused	Serial Port
3F0h - 3F7h	Unused	Floppy Controller #1
3F8h - 3FFh	Serial Port COM1 and IrDA	Serial Port