

ISR_COM1_Pas.doc

```
1 Program ISR_COM1;
2 {+-----+
3   Höcht, 10.5.1996
4   Programmierung des Interrupt-Controllers
5   IContr_5;
6
7
8   ISR_COM1:   Adresse $2F0 bis $2FF
9
10  Mit diesem Programm wird der Ablauf eines Interrupts der seriellen Schnitt-
11  stelle COM1 (IRQ4) demonstriert, der durch die Modemleitungen DTS und DSR
12  ausgelöst wird.
13
14  Dazu muß eine Kurzschlußstecker in die serielle Schnittstelle COM1 gesteckt
15  werden mit folgenden Verbindungen:
16
17          25-poliger Stecker   9-poliger Stecker
18  TxD-RxD      2 - 3           3 - 2           (hier nicht verwendet)
19  RTS-CTS      4 - 5           7 - 8
20  DTR-DSR      20 - 6          4 - 6
21
22
23  Durch Betätigung der Tasten 1 und 2 wird RTS gesetzt bzw. zurückgesetzt,
24  durch Betätigung der Tasten 3 und 4 wird DTR gesetzt bzw. zurückgesetzt.
25
26  Über den Kurzschlußstecker werden die beiden Signale in die Eingänge
27  CTS und DSR zurückgeschickt an die eigne Schnittstelle und lösen dort
28  den Interrupt aus, falls sich der Status der betreffenden Leitung ändert.
29  +-----+
30  USES DOS, CRT;
31
32
33
34  Type Proc = PROCEDURE;
35  VAR   p_IRQ4_alt      : Pointer;
36        IRQ4_ISR_alt    : Procedure ABSOLUTE p_IRQ4_alt;    { alte COM1-ISR }
37
38        p_IRQ1_alt      : Pointer;
39        IRQ1_ISR_alt    : Procedure ABSOLUTE p_IRQ1_alt;    { alte Tastatur-ISR}
40
41
42        p_Alte_Exit_Prozedur : Pointer;
43
44
45
46  { Registerinhalte: }
47  Var IMR      : Byte;    { Interrupt Mask Register des Interruptcontrollers 8259A }
48      IMR_alt  : Byte;
49
50      IER      : Byte;    { Interrupt Enable Register der Seriellen Schnittstelle }
51      IER_alt  : Byte;
52      MCR      : Byte;    { Modem Control Register, insbesondere Zugriff auf OUT2,
53                          das aktiviert werden muß, damit die Interrupts an den
54                          Interrupt Controller weitergeschaltet werden }
55      MCR_alt  : Byte;
56      LCR      : Byte;    { Line Control Register }
57      LCR_Alt  : Byte;
58      MSR      : Byte;    { Modem Status Register }
59      LSR      : Byte;    { Line Status Register }
60      IIR      : Byte;    { Interrupt Identification Register }
61
62  CONST
63      { Port-Adressen:
64      Interrupt-Controller:}
```

ISR_COM1_Pas.doc

```
65     p_IMR    = $21; { Interrupt Mask Register 8259A }
66
67     { Serielle Schnittstelle COM1 mit IRQ4 }
68
69     p_RD_TD = $3F8; { Empfangs/Sendpuffer}
70     p_IER   = $3F9; { Interrupt Enable Register }
71     p_IIR   = $3FA; { Interrupt Identification Register }
72     p_LCR   = $3FB; { Line Control Register }
73     p_MCR   = $3FC; { Modem Control Register }
74     p_LSR   = $3FD; { Line Status Register }
75     p_MSR   = $3FE; { Modem Status Register }
76
77
78
79
80     VAR IVek : String;
81     bs,
82     bs1      : String;    { Bytestring zur Ausgabe auf dem Bildschirm }
83
84
85     Zei      : Char;
86
87
88     (*-----*)
89     Function Hexaus(Hex:Byte): String;
90     Var Zahl: Byte;
91         s      : String;
92     Procedure Halbbyteaus(Zahl:Byte);
93     Begin
94         If Zahl > 9 Then
95             Begin
96                 s := s + CHR(Zahl + 55);
97             End
98         Else
99             s := s + chr(Zahl + 48);
100     End;
101
102     Begin
103         s := '';
104         Zahl := Hex DIV 16;
105         Halbbyteaus(Zahl);
106         Hex := Hex MOD 16;
107         Halbbyteaus(Hex);
108         Hexaus := s;
109     End;
110
111
112
113     Function BitausWort(h : Word): String;
114     Var s      : String;
115         p,r    : Byte;
116         q      : Word;
117         max,i  : Byte;
118     Begin
119         Max := 16;
120         q := h;
121         p := 0;
122         if h < 256 then
123             Max := 8;
124         s := '';
125         For i := 1 to max do
126             Begin
127                 if (p mod 4 = 0) And (p > 0) then
128                     s := ' ' + s;
```

ISR_COM1_Pas.doc

```

129     inc(p);
130     r := q mod 2;
131     if r = 1 then
132         s := '1' + s
133     Else
134         s := '0' + s;
135     q := q div 2;
136     End;
137     BitausWort := s;
138 End;
139
140
141
142
143
144
145
146 Function Hol_Interrupt_Vektor(Nr:Byte):String;
147 Type Ptr_segof = record
148         offset,
149         segment      : Word;
150     end;
151
152 Var p      : Pointer;
153     s,o    : Word;
154     Hexzahl : String;
155 Begin
156     GetIntVec(Nr,p);
157     s := Ptr_segof(p).segment;
158     o := Ptr_segof(p).offset;
159
160     hexzahl := '';
161     hexzahl := hexaus(s div 256) + hexaus(s mod 256) + ':' +
162               hexaus(o div 256) + hexaus(o mod 256);
163
164     Hol_Interrupt_Vektor := Hexzahl;
165
166 End;
167
168 {|||||||      Hilfsprogramme zur Anzeige der Registerinhalte      |||||||}
169
170 Procedure Anzeige_Register(PNr:Word);
171 {+-----+
172 |  Höcht, 24.4.1996
173 |  Zeigt den Inhalt des Registers auf PortNr PNr auf dem Bildschirm an.
174 +-----+}
175 Var s, s1 : String[80];
176     r      : Byte;
177
178 Begin
179     s := Hexaus(PNr);
180     s := 'Portnummer: ' + s + 'H ';
181     s1 := ' ';
182     case PNr of
183         p_IMR:  s1 := ' Interrupt Mask Register ';
184
185         p_RD_TD: s1 := ' Sende/Empfangsregister ';
186         p_IER:  s1 := ' Interrupt Enable Register ';
187         p_IIR:  s1 := ' Interrupt Identification Register ';
188         p_LCR:  s1 := ' Line Control Register ';
189         p_MCR:  s1 := ' Modem Control Register ';
190         p_LSR:  s1 := ' Line Status Register ';
191         p_MSR:  s1 := ' Modem Status Register ';
192     End; { Case }

```

ISR COM1 Pas.doc

```

193   R := Port [PNr];
194   bs := Hexaus (R);
195   bs1 := BitAusWort (R);
196   s := s + s1 + ' ' + bs + 'H ' + bs1 + 'B';
197   Writeln(s);
198 End;
199
200 {||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
201
202
203 (*-----*)
204
205 {||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
206 {||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
207 {|||| Interrupt Service Routinen |||}
208 {||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
209 {||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
210
211
212 {||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
213 {|||| Keyboard Interrupt |||}
214 {||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
215
216 Procedure IRQ1_ISR_Neu;
217 {+-----+}
218 | Höcht, 10.6.1996
219 | Neue Interrupt-Service-Routine für den Tastaturinterrupt
220 |
221 | Mit den Zifferntasten 1 bis 4 werden die Steuerleitungen RTS, CTS von COM1,
222 | " " " 5 bis 8 " COM2,
223 | " " Tasten 9,0,ß,´ " COM3
224 | gesetzt und zurückgesetzt.
225 |
226 | Anschließend wird die ursprüngliche BIOS-Routine zur regulären Weiterbe-
227 | handlung des Tastendrucks aufgerufen.
228 |
229 | Dieser Tastaturinterrupt wird sowohl beim Drücken, als auch beim Loslassen
230 | der Taste ausgelöst. Beim Drücken wird das oberste Bit im Port $60 ge-
231 | löscht und in die verbleibenden 7 Bits die laufende Nummer der gedrückten
232 | Taste geschrieben. Beim Loslassen wird das oberste Bit wieder gesetzt,
233 | während die anderen 7 Bits erhalten bleiben.
234 | +-----+}
235
236 Interrupt;
237 Var Taste : Byte;
238 Regs : Registers;
239 Modcontrolreg : Byte;
240 Begin
241
242
243 Taste := Port [$60]; { Tastaturpuffer lesen }
244 if Taste < 127 then { also Interrupt beim DRÜCKEN, NICHT beim Loslassen!!!}
245 Begin
246 gotoxy(1,10); clreol; { Alte Anzeige bei Auslösen des Interrupts leeren}
247 Write('Diesmal KEIN COM1-Interrupt!');
248 gotoxy(1,22); clreol; { Alte Anzeige bei Auslösen des Interrupts leeren}
249 gotoxy(1,23); clreol; { Alte Anzeige bei Auslösen des Interrupts leeren}
250 sound(1000);
251 delay(500);
252 nosound;
253 End
254 Else
255 Begin { Zur Verdeutlichung, daß der Tastaturinterrupt zweimal aufgerufen
256 wird: Sowohl beim Drucken (tieferer Ton) als auch beim Loslassen

```

ISR_COM1_Pas.doc

```
257         (höherer Ton) }
258     sound(1500);
259     delay(200);
260     nosound;
261 End;
262
263 { COM1-Steuerleitungen:}
264 If Taste = 2 Then      { = Ziffer 1 }
265     Begin
266         Modcontrolreg := Port[$3fc];
267         Modcontrolreg := Modcontrolreg OR 2;      (*RTS = 1      *)
268         Port[$3fc]     := Modcontrolreg;          (* RTS := 1      *)
269     End;
270 If Taste = 3 Then      { = Ziffer 2 }
271     Begin
272         Modcontrolreg := Port[$3fc];
273         Modcontrolreg := Modcontrolreg AND $FD;    (* 1111 1101    *)
274         Port[$3fc]     := Modcontrolreg;          (* RTS := 0      *)
275     End;
276 If Taste = 4 Then      { = Ziffer 3 }
277     Begin
278         Modcontrolreg := Port[$3fc];
279         Modcontrolreg := Modcontrolreg OR 1;      (*DTR =          1*)
280         Port[$3fc]     := Modcontrolreg;          (* DTR :=          1*)
281     End;
282 If Taste = 5 Then      { = Ziffer 4 }
283     Begin
284         Modcontrolreg := Port[$3fc];
285         Modcontrolreg := Modcontrolreg AND $FE;    (* 1111 1110    *)
286         Port[$3fc]     := Modcontrolreg;          (* DTR:= 0      *)
287     End;
288
289 { COM2-Steuerleitungen:}
290 If Taste = 6 Then      { = Ziffer 5 }
291     Begin
292         Modcontrolreg := Port[$2fc];
293         Modcontrolreg := Modcontrolreg OR 2;      (*RTS = 1      *)
294         Port[$2fc]     := Modcontrolreg;          (* RTS := 1      *)
295     End;
296 If Taste = 7 Then      { = Ziffer 6 }
297     Begin
298         Modcontrolreg := Port[$2fc];
299         Modcontrolreg := Modcontrolreg AND $FD;    (* 1111 1101    *)
300         Port[$2fc]     := Modcontrolreg;          (* RTS := 0      *)
301     End;
302 If Taste = 8 Then      { = Ziffer 7 }
303     Begin
304         Modcontrolreg := Port[$2fc];
305         Modcontrolreg := Modcontrolreg OR 1;      (*DTR =          1*)
306         Port[$2fc]     := Modcontrolreg;          (* DTR :=          1*)
307     End;
308 If Taste = 9 Then      { = Ziffer 8 }
309     Begin
310         Modcontrolreg := Port[$2fc];
311         Modcontrolreg := Modcontrolreg AND $FE;    (* 1111 1110    *)
312         Port[$2fc]     := Modcontrolreg;          (* DTR:= 0      *)
313     End;
314
315     IRQ1_ISR_alt;      (* ursprüngliche Bios-Routine aufrufen*)
316 end;
317
318
319
320
```

ISR_COM1_Pas.doc

```
321 {|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
322 {||||          IRQ4 Interrupt von COM1          |||}
323 {|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
324
325 Procedure IRQ4_ISR_Neu;
326 {+-----+
327   Höchst, 10.6.1996
328   Neue Interrupt-Service-Routine für die Interruptanforderung (= Interrupt
329   Request) Nr. 4, also für COM1
330
331   Folgende Aktionen werden durchgeführt:
332   - In Zeile 10 des Bildschirms Meldung "Interrupt COM1 ausgelöst" mit Pieps
333   - In Zeile 22 der Inhalt des Interrupt Identification Register IIR
334   - anschließend wird das Modem-Status-Register MSR gelesen und ausgegeben
335   - In Zeile 23 dasselbe, um zu zeigen, daß ein Lesevorgang des Modem-Status-
336     registers das Bit Nr.0 des Interrupt-Identification-Registers IIR setzt
337     und damit die Interrupt-Leitung an den Interruptcontroller wieder auf
338     den inaktiven Status 1 (also negative Logik!) setzt.
339     Außerdem ist am Bit 0 des Modem Status-Registers zu sehen, daß durch das
340     Lesen dieses Registers Bit 0 wieder gelöscht wird, das die Änderung der
341     Leitung CTS registrierte und den Interrupt auslöste.
342
343   +-----+
344
345 Interrupt;
346 Var s      : String[80];
347 Begin
348   Gotoxy(1,10); ClrEol;
349   Write(' COM1 IRQ4 ausgelöst!');
350   Sound(500);
351   Delay(500);
352   Nosound;
353   Gotoxy(1,22);
354
355   IIR := Port[p_IIR];
356   s    := BitAusWort(IIR);
357   Write('Interrupt ID-Register: ' + s);
358
359
360   MSR := Port[p_MSR];
361   s    := BitAusWort(MSR);
362   Writeln('    Modem Statusregister: '+ s);
363   Gotoxy(40,10);
364   If MSR AND $02 = $02 then
365     Begin
366       Write('DSR (Pin 6) hat sich geändert auf ');
367       If (MSR AND $20) = $20 then
368         Write('1')
369       Else
370         Write('0');
371     End
372   Else
373     Begin
374       Write('CTS (Pin 5) hat sich geändert auf ');
375       If (MSR AND $10) = $10 then
376         Write('1')
377       Else
378         Write('0');
379     End;
380
381   Gotoxy(1,23);
382   IIR := Port[p_IIR];
383   s    := BitAusWort(IIR);
384   Write('Interrupt ID-Register: ' + s);
```

ISR_COM1_Pas.doc

```
385
386   MSR := Port[p_MSR];
387   s    := BitAusWort(MSR);
388   Writeln('    Modem Statusregister: ' + s);
389
390   {ASM
391   Pushf
392   End;
393   IRQ4_ISR_alt;}
394   port[$20] := $20;    { Interrupt Controller freigeben }
395 End;
396
397
398
399 Procedure Programm_Exit; far;
400 {+-----+
401 | Diese Prozedur wird beim Verlassen des Programms aufgerufen, also auch
402 | beim Verlassen durch einen Laufzeitfehler oder durch <Ctrl><C>.
403 | Dabei wird die alte Interrupt-Prozedur wiederhergestellt und der Inter-
404 | rupt-Controller wieder freigegeben, falls der Abbruch durch <Ctrl><C>
405 | erfolgte. In diesem Fall wird nämlich die alte Interrupt-Prozedur, die
406 | den Interrupt-Controller freigibt nicht mehr aufgerufen.
407 | +-----+}
408 Begin
409   SetIntVec($9,p_IRQ1_alt);    { Interruptvektor Nr. 9 = Zeiger auf alte
410                               | Serviceroutine des Tastaturinterrupts wieder
411                               | herstellen }
412   SetIntVec($C,p_IRQ4_alt);    { Interruptvektor Nr. 12 = Zeiger auf alte
413                               | Serviceroutine des COM1-Interrupts IRQ4
414                               | wieder herstellen }
415   ExitProc := p_Alte_Exit_Prozedur; { Vektor auf alte Standard-Exit-Prozedur
416                               | belegen }
417
418   Port[$20] := $20;            { Interrupt Controller sicherheitshalber
419                               | freigeben }
420
421   { Registerinhalte restaurieren:}
422   Port[p_IER] := IER_alt;    { Interrupt-Enable-Register von COM1}
423   Port[p_IMR] := IMR_alt;    { Interrupt-Mask-Register von COM1 }
424   Port[p_MCR] := MCR_alt;    { Modem-Control-Register von COM1 }
425   PORT[p_LCR] := LCR_alt;    { Line-Control-Register von COM1 }
426
427   if keypressed then readkey;
428 End;
429
430
431 {|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}
432
433
434
435 Begin
436   {Alte Registerinhalte retten für die Restaurierung nach Programmende:}
437   IMR_alt := Port[p_IMR];    { Interrupt-Mask-Register des
438                               | Interrupt-Controllers 8259A }
439   IER_alt := Port[p_IER];    { Interrupt-Enable-Register von COM1}
440   IMR_alt := Port[p_IMR];    { Interrupt-Mask-Register von COM1 }
441   MCR_alt := Port[p_MCR];    { Modem-Control-Register von COM1 }
442   LCR_alt := PORT[p_LCR];    { Line-Control-Register von COM1 }
443
444
445
446
447
448
```

ISR_COM1_Pas.doc

```
449
450 clrscr;
451 MSR := Port[p_MSR];
452 Writeln('COM1 auf Adressen 3F8 bis 3FF mit IRQ 4   Bit Nr. 4 für Modem-Status-Intr. ');
453 Anzeige_Register(p_IER);
454 IER := Port[p_IER];
455 IER := IER OR $08;
456 Port[p_IER] := IER;
457 Anzeige_Register(p_IER);
458
459
460 Writeln;
461 Writeln( 'Nun muß Bit 3 im Modem Control Register gesetzt werden (= OUT2), ' );
462 Writeln( 'damit die Interrupts weitergeleitet werden.' );
463 Anzeige_Register(p_MCR);
464 MCR := Port[p_MCR];
465 MCR := MCR OR $08; { Out2 aktiv schalten, damit Interrupts weitergeleitet
466                    werden an den Interrupt Controller }
467 Port[p_MCR] := MCR;
468 Anzeige_Register(p_MCR);
469 Writeln;
470
471
472 Writeln('Interrupt Controller (8259A): Maskenbit 4 löschen!');
473 Anzeige_Register(p_IMR);
474 IMR := Port[$21];
475 IMR := IMR AND $EF; { Maskenbit Nr 4 löschen: xxx0 xxxx }
476 Port[$21] := IMR;
477 Anzeige_Register(p_IMR);
478 Writeln;
479
480
481
482
483 {||||| Nun kommt das Verbiegen der Interruptvektoren von IRQ1 und IRQ4 ||||}
484
485
486 Writeln('IRQ4-Vektor an Adresse 4*(8+4)=48, Interrupt Nr 8+4 = CH');
487 IVek := Hol_Interrupt_Vektor(8+4); {IRQ4}
488 Writeln('Alter Interruptvektor IRQ4 im ROM-BIOS:      ', IVek);
489
490
491 GetIntVec($9, p_IRQ1_alt); { Zeiger auf alte Tastatur-Interruptprozedur retten }
492 GetIntVec($C, p_IRQ4_alt); { Zeiger auf alte COM1-Interruptprozedur retten }
493
494
495 p_Alte_Exit_Prozedur := ExitProc; { Exit-Handler installieren }
496 ExitProc := @Programm_Exit;
497
498 SetIntVec($9, @IRQ1_ISR_neu); { Zeiger auf neue ISR in die
499                               Speicherplätze 9*4 bis 9*4 + 3
500                               eintragen}
501
502 SetIntVec($C, @IRQ4_ISR_neu); { Zeiger auf neue ISR in die
503                               Speicherplätze 12*4 bis 12*4 + 3
504                               eintragen}
505
506
507
508 IVek := Hol_Interrupt_Vektor(8+4); {IRQ4}
509 Writeln('Neuer Interruptvektor IRQ4 eigene Routine:      ', IVek);
510
511
512 repeat until keypressed; { Warteschleife }
```


ISR_COM1_Pas.doc

```
513  clrscr;
514
515
516  {||||  Hilf für die Tastendrucke auf den Bildschirm          |||||}
517
518  ClrScr;
519  Writeln('Steuerung der V24-Steuerausgänge mit den Tasten 1 bis 4 bzw. 5 bis 8');
520  Textcolor(green);
521  Highvideo;
522  Writeln('**** Port 1/2 (COM1/COM2)          *****');
523  Normvideo;
524  textcolor(white);
525  Writeln('Taste 1/5 :   RTS = 1  Spannung +, LED grün');
526  Writeln('Taste 2/6 :   RTS = 0  Spannung -, LED rot ');
527  Writeln('Taste 3/7 :   DTR = 1  Spannung +, LED grün');
528  Writeln('Taste 4/8 :   DTR = 0  Spannung -, LED rot ');
529  Writeln;
530  Write('Programmende mit Taste p!');
531  gotoxy(1,20);
532  Write('Gedrückte Taste:   ');
533
534  Gotoxy(1,24);
535  Write('Tiefster TON: COM1-Interrupt');
536  Gotoxy(1,25);
537  Write('Mittlerer Ton: Tastatur-Intr. Drücken,  Hoher Ton: Loslassen');
538
539  Repeat
540
541      Zei := Readkey;
542      textcolor(white);
543      gotoxy(20,20);
544      Write(zei);
545      Gotoxy(1,12);
546      Anzeige_Register(p_IIR);
547      Anzeige_Register(p_MSR);
548      Anzeige_Register(p_LSR);
549      Anzeige_Register(p_MCR);
550      Anzeige_Register(p_LCR);
551
552
553
554  Until Zei = 'p';
555
556
557
558
559
560
561
562  End.
563
```