```
1   $NOMOD51
2   #include <Reg517a.inc>
3
4       ; -- [Definieren der Konstanten] -- ;
5       Nmax EQU 60
6       Px EQU 20
7
8       ; -- Definition der beiden Eckpunkte A und B im Format VVVV VV.NN | NNNN NNNN + i * VVVV VV.NN |
    NNNN NNNN -- ;
9       ; A = -2,25 - 1,5i
10      A_RE_H EQU 111101$11b
11      A_RE_L EQU 00000000b
12      A_IM_H EQU 111110$10b
13      A_IM_L EQU 00000000b
14
15      ; B = 0,75 + 1,5i
16      B_RE_H EQU 000000$11b
17      B_RE_L EQU 00000000b
18      B_IM_H EQU 000001$10b
19      B_IM_L EQU 00000000b
20
21      ; -- [Definieren von genutzten Speicheradressen] -- ;
22      ; Aufteilung in High- (H) und Low-Byte (L)
23      ; -- Speicherstellen für Addition von komplexen Zahlen A + B im Format VVVVVV.NN | NNNNNNNN + i
    * VVVVVV.NN | NNNNNNNN -- ;
24      ; Aufteilung in Real- und Imaginaerteil
25
26      ; A
27      ADD_A_RE_H EQU 020h
28      ADD_A_RE_L EQU 021h
29
30      ADD_A_IM_H EQU 022h
31      ADD_A_IM_L EQU 023h
32
33      ; B
34      ADD_B_RE_H EQU 024h
35      ADD_B_RE_L EQU 025h
36
37      ADD_B_IM_H EQU 026h
38      ADD_B_IM_L EQU 027h
39
40      ; -- Speicherstellen fuer die Addition zweier 16Bit Festkommazahlen A und B -- ;
41      ADD_A_H EQU 028h
42      ADD_A_L EQU 029h
43
44      ADD_B_H EQU 02Ah
45      ADD_B_L EQU 02Bh
46
47      ; -- Speicherstellen fuer die Multiplikation von zwei Festkommazahlen A und B -- ;
48      MUL_A_H EQU 02Ch
49      MUL_A_L EQU 02Dh
50
51      MUL_B_H EQU 02Eh
52      MUL_B_L EQU 02Fh
53
54      ; -- Speicherstellen fuer die die Quadrierung einer komplexen Zahl im Format (a + b*i) -- ;
55      QUAD_A_H EQU 030h
56      QUAD_A_L EQU 031h
57
58      QUAD_B_H EQU 032h
59      QUAD_B_L EQU 033h
60
61      ; -- Speicherstellen fuer Division A/B von  zwei 16Bit Zahlen im obigen Format -- ;
62      DIV_A_H EQU 034h
63      DIV_A_L EQU 035h
64
65      DIV_B_H EQU 036h
66      DIV_B_L EQU 037h
67
68      ; -- Komplementbildung -- ;
69      comp_H EQU 038h
70      comp_L EQU 039h
71
72      ; -- Abstand zwischen den Punkten -- ;
73      dist_adr_H EQU 03Ah
74      dist_adr_L EQU 03Bh
75
```

```
76        ; -- Schleifenzaehler -- ;
77        loop_outer EQU 03Ch
78        loop_inner EQU 03Dh
79
80        ; -- Temporärer Punkt c -- ;
81        ; Aufteilung in Real- und Imaginaerteil
82        C_RE_H EQU 03Eh
83        C_RE_L EQU 03Fh
84
85        C_IM_H EQU 040h
86        C_IM_L EQU 041h
87
88        ; -- Mandelbrot-Folge -- ;
89        ; Aufteilung in Real- und Imaginaerteil
90        Z_RE_H EQU 042h
91        Z_RE_L EQU 043h
92
93        Z_IM_H EQU 044h
94        Z_IM_L EQU 045h
95
96        ; ------------------------------------------------- ;
97
98
99
100       ; -- [Abstand von A und B auf der reellen Achse ausrechnen] -- ;
101       ; Der Abstand ist gegeben durch (-A + B)/Px
102       ; Der Abstand auf der imaginaeren Achse ist gleichzusetzen
103
104       ; -- Komplement von A -- ;
105       MOV comp_H, #A_RE_H
106       MOV comp_L, #A_RE_L
107
108       LCALL comp
109
110       ; -- Berechnung von -A + B -- ;
111       ; Schreiben der Speicherstellen fuer Addition
112       ; Imaginaerteil ist 0
113       MOV ADD_A_H, comp_H
114       MOV ADD_A_L, comp_L
115
116       MOV ADD_B_H, #B_RE_H
117       MOV ADD_B_L, #B_RE_L
118
119       LCALL add16
120
121       ; -- Ergebnis von (-A + B) durch Px teilen -- ;
122       ; Ergebnis ist in den ersten vier Byte (urspruenglich A)
123
124       ; Schreiben der Speicherstellen fuer Division
125       MOV DIV_A_H, ADD_A_H
126       MOV DIV_A_L, ADD_A_L
127
128       MOV DIV_B_H, #0d
129       MOV DIV_B_L, #Px
130
131       LCALL div16
132
133       ; -- Schreiben des Ergebnisses in dist_adr -- ;
134       MOV dist_adr_H, DIV_A_H
135       MOV dist_adr_L, DIV_A_L
136
137       ; ------------------------------------------------- ;
138
139
140
141       ; -- [Hauptschleife] -- ;
142   main:
143
144       ; -- Aeusserer Schleifencounter initialisieren --> Anzahl der Reihen -- ;
145       ; - Bilden des Komplements von A auf der imaginaeren Achse
146       MOV comp_H, #A_IM_H
147       MOV comp_L, #A_IM_L
148
149       LCALL comp
150
151       ; - Berechnen des Abstands (-A + B) auf der imaginaeren Achse;
152       ; Schreiben der Speicherstellen fuer Addition
```

```
153         MOV ADD_A_H, comp_H
154         MOV ADD_A_L, comp_L
155
156         MOV ADD_B_H, #B_IM_H
157         MOV ADD_B_L, #B_IM_L
158
159         LCALL add16
160
161         ; - Ergebnis der Addition durch den Abstand auf der reellen Achse teilen
162         ; Ergebnis ist in den ersten vier Byte (urspruenglich A)
163
164         ; Schreiben der Speicherstellen fuer Divison
165         MOV DIV_A_H, ADD_A_H
166         MOV DIV_A_L, ADD_A_L
167
168         MOV DIV_B_H, dist_adr_H
169         MOV DIV_B_L, dist_adr_L
170
171         LCALL div16
172
173         ; Anzahl der Punkte ist als Dezimalzahl in dem Low-Byte des Ergebnis der Division
174         MOV loop_outer, DIV_A_L
175
176         ; -- Anfangspunkt fuer C -- ;
177         ; Realteil aus A, Imaginaerteil von B
178         MOV C_RE_H, #A_RE_H
179         MOV C_RE_L, #A_RE_L
180         MOV C_IM_H, #B_IM_H
181         MOV C_IM_L, #B_IM_L
182
183         ; -- Berechnung und Ausgabe des Apfelmaennchens -- ;
184         outer_loop:
185             ; Counter der inneren Schleife zuruecksetzen
186             MOV loop_inner, #Px
187
188             ; Realteil von C zuruecksetzen
189             ; -> Links am Rand anfangen und nach rechts vorarbeiten
190             MOV C_RE_H, #A_RE_H
191             MOV C_RE_L, #A_RE_L
192
193             inner_loop:
194                 ; Zuruecksetzen des Iterationscounters
195                 MOV R7, #0d
196
197                 ; Zuruecksetzen von Z
198                 MOV Z_RE_H, #0d
199                 MOV Z_RE_L, #0d
200                 MOV Z_IM_H, #0d
201                 MOV Z_IM_L, #0d
202
203                 ; Mandelbrotberrechnung ausfuehren
204                 LCALL mandelbrot
205
206                 ; Farbwert berechnen und ausgeben
207                 LCALL calc_ascii
208
209                 ; C + Abstand auf reeller Achse
210                 MOV ADD_A_H, C_RE_H
211                 MOV ADD_A_L, C_RE_L
212                 MOV ADD_B_H, dist_adr_H
213                 MOV ADD_B_L, dist_adr_L
214
215                 LCALL add16
216
217                 MOV C_RE_H, ADD_A_H
218                 MOV C_RE_L, ADD_A_L
219
220                 ; loop_inner verringern und zurueckspringen, falls nicht 0
221                 DJNZ loop_inner, inner_loop
222
223             ; C - Abstand auf reeller Achse, aber in imaginaerer Richtung
224             ; -> Eine "Reihe" nach unten
225             MOV ADD_A_H, C_IM_H
226             MOV ADD_A_L, C_IM_L
227
228             ; Komplement der Distanz, da Subtraktion
229             MOV comp_H, dist_adr_H
```

```
230             MOV comp_L, dist_adr_L
231
232             LCALL comp
233
234             MOV ADD_B_H, comp_H
235             MOV ADD_B_L, comp_L
236
237             ; Berechnung von C - Abstand
238             LCALL add16
239
240             ; Imaginaerteil des neuen Punktes speichern
241             MOV C_IM_H, ADD_A_H
242             MOV C_IM_L, ADD_A_L
243
244             ; Zeilenumbruch in UART ausgeben
245             MOV R7, #10d
246             LCALL write_ascii
247
248             ; loop_outer verringern und zurueckspringen, falls nicht 0
249             DJNZ loop_outer, outer_loop
250
251         ; Programm beenden
252         LJMP finish
253
254         ; ------------------------------------------------- ;
255
256
257
258         ; -- [Berechnung einer Mandelbrot-Iteration -- ;
259     mandelbrot:
260         ; Zuruecksetzen des Watchdog-Timers, da sonst das Programm nach zu vielen Iterationen
        neugestartet wird
261         ORL 0A8h, #0100$0000
262         ORL 0B8h, #0100$0000
263
264         ; Iterationszaehler erhoehen
265         INC R7
266
267         ; Berechnung von zn^2
268         MOV QUAD_A_H, Z_RE_H
269         MOV QUAD_A_L, Z_RE_L
270
271         MOV QUAD_B_H, Z_IM_H
272         MOV QUAD_B_L, Z_IM_L
273
274         LCALL quad
275
276         ; Berechnung von zn^2 + c
277         ; Ergebnis der Quadrierung direkt fuer die Addition weiterverwenden
278         MOV ADD_A_RE_H, QUAD_A_H
279         MOV ADD_A_RE_L, QUAD_A_L
280         MOV ADD_A_IM_H, QUAD_B_H
281         MOV ADD_A_IM_L, QUAD_B_L
282
283         MOV ADD_B_RE_H, C_RE_H
284         MOV ADD_B_RE_L, C_RE_L
285         MOV ADD_B_IM_H, C_IM_H
286         MOV ADD_B_IM_L, C_IM_L
287
288         LCALL addImAB
289
290         ; Ergebnis in Z speichern
291         MOV Z_RE_H, ADD_A_RE_H
292         MOV Z_RE_L, ADD_A_RE_L
293
294         MOV Z_IM_H, ADD_A_IM_H
295         MOV Z_IM_L, ADD_A_IM_L
296
297         ; Schauen, ob zn^2 > 4, also a^2 + b^2 > 4
298         LCALL checkIfSmaller
299
300         ; Schauen, ob Ergebnis der Berechnung von checkIfSmaller negatives Vorzeichen hat
301         ; -> zn^2 < 4
302         MOV A, ADD_A_H
303         ANL A, #1000$0000b
304         JNZ check_over ; Springe, falls negatives Vorzeichen
305
```

```
306       ; Ab hier gilt zn^2 >= 4
307       ; Schauen, ob Ergebnis der Rechnung genau 0 -> zn^2 == 4
308       MOV A, ADD_A_H
309       ORL A, ADD_A_L
310       JNZ mandelbrot_finished ; Springe, falls Ergebnis nicht 0 (also > 0 -> zn^2 > 4)
311
312       check_over:
313
314       ; Wiederholen, falls maximale Anzahl an Iterationen noch nicht erreicht
315       CJNE R7, #Nmax, mandelbrot
316
317       mandelbrot_finished:
318
319       ; Zurueck
320       RET
321
322       ; Teilprogramm, welches die Abbruchbedingung der Mandelbrotmenge ueberprueft
323       checkIfSmaller:
324           ; Quadrieren (Multiplizieren mit sich selbst) des Realteils (a) von Z
325           MOV MUL_A_H, Z_RE_H
326           MOV MUL_A_L, Z_RE_L
327           MOV MUL_B_H, Z_RE_H
328           MOV MUL_B_L, Z_RE_L
329
330           LCALL mul16
331
332           ; Speichern fuer die folgende Addition
333           MOV ADD_A_H, MUL_A_H
334           MOV ADD_A_L, MUL_A_L
335
336           ; Quadrieren des Imaginaerteils (b) von Z
337           MOV MUL_A_H, Z_IM_H
338           MOV MUL_A_L, Z_IM_L
339           MOV MUL_B_H, Z_IM_H
340           MOV MUL_B_L, Z_IM_L
341
342           LCALL mul16
343
344           ; Speichern fuer Addition
345           MOV ADD_B_H, MUL_A_H
346           MOV ADD_B_L, MUL_A_L
347
348           ; Berechnung von a^2 + b^2
349           LCALL add16
350
351           ; Speichern der Zahl -4 in der Darstellung VVVVVV.NNNNNNNNN
352           MOV ADD_B_H, #111100$00b
353           MOV ADD_B_L, #00000000b
354
355           ; Subtraktion mit 4 von dem Ergebnis der Addition a^2 + b^2
356           LCALL add16
357
358           ; Zurueck
359           RET
360
361       ; ------------------------------------------------ ;
362
363
364
365       ; -- Division -- ;
366       ; Das Ergebnis findet sich in den Speicherzellen der Zahl A
367   div16:
368       ; Algorithmus:
369       ; - 3 Teile:
370       ;   - 1. Teil: Divisor left-shift bis erste 1 ganz links angekommen
371       ;   - 2. Teil: Divisor right-shift und von Dividend abziehen, sofern moeglich
372       ;   - 3. Teil: Ergebnis abspeichern
373       ;   - Wiederholen, bis Divisor wieder gleich wie bei Anfang
374
375       ; Divisor
376       MOV R3, DIV_B_H
377       MOV R2, DIV_B_L
378
379       ; Dividend
380       MOV R1, DIV_A_H
381       MOV R0, DIV_A_L
382
```

```
383         ; B als Counter
384         MOV B, #0d
385         div1:
386             ; B erhoehen
387             INC B
388 
389             ; Low-Byte von Divisor in A und left-shift
390             MOV A, R2
391             RLC A
392             MOV R2, A
393 
394             ; High-Byte von Divisor in A und left-shift
395             MOV A, R3
396             RLC A
397             MOV R3, A
398 
399             ; Wiederholen, bis Carry-Flag von High-Byte gesetzt
400             ; -> Ganz ans Ende "geschoben"
401             JNC div1
402 
403         ; Divisor wieder nach und nach right-shift
404         div2:
405             ; Zuerst mit High-Byte von Divisor
406             MOV A, R3
407             RRC A
408             MOV R3, A
409 
410             ; Dann Low-Byte von Divisor
411             MOV A, R2
412             RRC A
413             MOV R2, A
414 
415             CLR C
416 
417             ; Erstelle Sicherheitskopie von Dividend, falls Subtraktion fehlschlaegt
418             ; Nutze Speicheradressen der Register R7 und R6, um einfacher zu kopieren
419             MOV 07h, R1
420             MOV 06h, R0
421 
422             ; Low-Byte von Dividend in A
423             MOV A,R0
424 
425             ; Subtrahiere Low-Byte von Divisor
426             SUBB A, R2
427             MOV R0, A
428 
429             ; Wiederholen fuer High-Byte
430             MOV A, R1
431             SUBB A, R3
432             MOV R1, A
433 
434             ; Testen, ob Subtraktion erfolgreich
435             ; Ergebnis ist 1, wenn Carry nicht gesetzt
436             JNC div3
437 
438             ; Sonst Sicherheitskopie wiederherstellen
439             ; Nutze Speicheradressen, um einfacher zu kopieren
440             MOV R1, 07h
441             MOV R0, 06h
442 
443         div3:
444             ; Invertiere Carry (Ergebnis 1, falls Carry nicht gesetzt)
445             CPL C
446 
447             ; Low-Byte Ergebnis in R4
448             MOV A, R4
449 
450             ; Shift mit Carry um Ergebnis direkt in R4 zu speichern
451             RLC A
452             MOV R4, A
453 
454             ; Wiederholen mit High-Byte
455             MOV A, R5
456             RLC A
457             MOV R5, A
458 
459             ; Wiederholen, bis counter 0
```

```
460             DJNZ B, div2
461
462         ; Ergebnis zurückschreiben
463         MOV DIV_A_H, R5
464         MOV DIV_A_L, R4
465
466         ; Zuruecksetzen der Register
467         MOV R0, #0d
468         MOV R1, #0d
469         MOV R2, #0d
470         MOV R3, #0d
471         MOV R4, #0d
472         MOV R5, #0d
473         MOV R6, #0d
474         MOV R7, #0d
475
476         ; Zurueck
477         RET
478
479         ; ------------------------------------------------- ;
480
481
482
483         ; -- [Addieren von zwei Komplexen Zahlen A und B] -- ;
484         ; Das Ergebnis findet sich in den Speicherzellen der Zahl A
485     addImAB:
486         ; Rechnung: (        Re(A)       + i *        Im(A)       ) + (        Re(B)       + i *
    Im(B)       )
487         ; Format:   (VVVVVV.NN |NNNNNNNN + i * VVVVVV.NN |NNNNNNNN) + (VVVVVV.NN |NNNNNNNN + i *
    VVVVVV.NN |NNNNNNNN)
488
489         ; Eingabe:  |   020h   |  021h    |   |  022h    |  023h    | |  024h    |  025h    |   |
    026h    |  027h    |
490         ; Konstante:|ADD_A_RE_H|ADD_A_RE_L|   |ADD_A_IM_H|ADD_A_IM_L| |ADD_B_RE_H|ADD_B_RE_L|
    |ADD_B_IM_H|ADD_B_IM_L|
491
492         ; Ausgabe im gleichen Format in urspruengliche Speicherstellen von A
493
494         ; Zuerst Realteil der beiden Zahlen addieren
495         MOV ADD_A_H, ADD_A_RE_H
496         MOV ADD_A_L, ADD_A_RE_L
497
498         MOV ADD_B_H, ADD_B_RE_H
499         MOV ADD_B_L, ADD_B_RE_L
500
501         LCALL add16
502
503         ; Ergebnis speichern
504         MOV ADD_A_RE_H, ADD_A_H
505         MOV ADD_A_RE_L, ADD_A_L
506
507
508         ; Dann Imaginaerteil
509         MOV ADD_A_H, ADD_A_IM_H
510         MOV ADD_A_L, ADD_A_IM_L
511
512         MOV ADD_B_H, ADD_B_IM_H
513         MOV ADD_B_L, ADD_B_IM_L
514
515         LCALL add16
516
517         ; Ergebnis speichern
518         MOV ADD_A_IM_H, ADD_A_H
519         MOV ADD_A_IM_L, ADD_A_L
520
521         ; Zurueck
522         RET
523
524         ; ------------------------------------------------- ;
525
526
527         ; -- [Addition von zwei 16 Bit Zahlen A und B] -- ;
528         ; Das Ergebnis findet sich in den Speicherzellen der Zahl A
529     add16:
530         ; -- Schauen, ob Zahlen negativ sind -- ;
531         ; Abtrennung des MSB von dem Highbyte von A
532         MOV A, ADD_A_H
```

```
533         RL A                ; MSB wird LSB
534         ANL A, #00000001b   ; Alle anderen Bits ignorieren
535         MOV R4, A           ; Zwischenspeichern
536
537         ; Gleiches fuer B wiederholen
538         MOV A, ADD_B_H
539         RL A
540         ANL A, #00000001b
541         MOV R5, A
542
543         ; R4 enthaelt ob A neg
544         ; R5 enthaelt ob B neg
545         MOV A, R4
546         ANL A, R5
547
548         ; Springen, falls nur eins negativ -> Addition kann nach normalen Regeln geschehen
549         JZ add_calc
550
551         ; Sonst: Beide komplementieren und das Ergebnis der Rechnung ebenfalls flippen
552
553         ; Zweierkomplement von A
554         MOV comp_H, ADD_A_H
555         MOV comp_L, ADD_A_L
556
557         LCALL comp
558
559         MOV ADD_A_H, comp_H
560         MOV ADD_A_L, comp_L
561
562         ; Zweierkomplement von B
563         MOV comp_H, ADD_B_H
564         MOV comp_L, ADD_B_L
565
566         LCALL comp
567
568         MOV ADD_B_H, comp_H
569         MOV ADD_B_L, comp_L
570
571         ; -- Berechnen von A + B -- ;
572         add_calc:
573             ; Clear carry
574             CLR C
575
576             ; Addieren der Low-Bytes von A und B
577             MOV R6, ADD_A_L
578             MOV A, R6
579             ADD A, ADD_B_L
580             MOV ADD_A_L, A
581
582             ; Addieren der High-Bytes inklusive des Carry-Bit der vorherigen Addition
583             MOV R6, ADD_A_H
584             MOV A, R6
585             ADDC A, ADD_B_H
586             MOV ADD_A_H, A
587
588             ; Schauen, ob beide negativ waren
589             MOV A, R4
590             ANL A, R5
591             JNZ add_flip
592
593             ; Zurueck
594             RET
595
596             ; Flippe Ergebnis, falls beide negativ waren
597             add_flip:
598                 MOV comp_H, ADD_A_H
599                 MOV comp_L, ADD_A_L
600
601                 LCALL comp
602
603                 MOV ADD_A_H, comp_H
604                 MOV ADD_A_L, comp_L
605
606                 ; Zurueck
607                 RET
608
609         ; ------------------------------------------------ ;
```

```
610
611
612
613
614         ; -- [Quadrierung einer imaginaeren Zahl] -- ;
615         ; Formel: (a + bi)^2 = a^2 - b^2 + 2abi
616         ; Das Ergebnis findet sich in den Speicherzellen der Zahl A
617     quad:
618         ; -- Berechnung von a^2 -- ;
619         MOV MUL_A_H, QUAD_A_H
620         MOV MUL_B_H, QUAD_A_H
621
622         MOV MUL_A_L, QUAD_A_L
623         MOV MUL_B_L, QUAD_A_L
624
625         LCALL mul16
626
627         ; Ergebnis der Rechnung fuer spaetere Addition zwischenspeichern
628         MOV ADD_A_RE_H, MUL_A_H
629         MOV ADD_A_RE_L, MUL_A_L
630
631         MOV ADD_A_IM_H, #0d
632         MOV ADD_A_IM_L, #0d
633
634         ; -- Berechnung von b^2 -- ;
635         MOV MUL_A_H, QUAD_B_H
636         MOV MUL_B_H, QUAD_B_H
637
638         MOV MUL_A_L, QUAD_B_L
639         MOV MUL_B_L, QUAD_B_L
640
641         LCALL mul16
642
643         ; Das Ergebnis von b^2 komplementieren
644         MOV comp_H, MUL_A_H
645         MOV comp_L, MUL_A_L
646
647         LCALL comp
648
649         ; -- Berechnung von a^2 - b^2 -- ;
650         MOV ADD_B_RE_H, comp_H
651         MOV ADD_B_RE_L, comp_L
652
653         MOV ADD_B_IM_H, #0d
654         MOV ADD_B_IM_L, #0d
655
656
657         LCALL addImAB
658
659         ; -- Berechnung von 2*a*b fuer den Imaginaerteil -- ;
660         ; Berechung von 2*a
661         MOV MUL_A_H, QUAD_A_H
662         MOV MUL_A_L, QUAD_A_L
663
664         ; Konvertiere 2 in die Darstellung VVVVVV.NNNNNNNNNN
665         MOV MUL_B_H, #000010$00b
666         MOV MUL_B_L, #0b
667
668         LCALL mul16
669
670         ; Multipliziere das Ergebnis mit b
671         MOV MUL_B_H, QUAD_B_H
672         MOV MUL_B_L, QUAD_B_L
673
674         LCALL mul16
675
676         ; -- Ergebnis zurueckschreiben -- ;
677         MOV QUAD_B_H, MUL_A_H
678         MOV QUAD_B_L, MUL_A_L
679
680         MOV QUAD_A_H, ADD_A_RE_H
681         MOV QUAD_A_L, ADD_A_RE_L
682
683         ; Zurueck
684         RET
685
686         ; ------------------------------------------------- ;
```

```
687
688
689
690         ; -- [Multiplikation zweier 16 Bit Zahlen] -- ;
691         ; Das Ergebnis findet sich in den Speicherzellen der Zahl A
692     mul16:
693         ; Vorgehen nach folgendem Schema:
694         ;         High  Low
695         ;
696         ;            A1  A2
697         ;     *      B1  B2
698         ; ------------------
699         ;           H22 L22
700         ; +     H21 L21
701         ; +     H12 L12
702         ; + H11 L11
703         ; --------------------
704         ;    P1  P2  P3  P4
705         ;
706         ; Die Zahlen signalisieren, welche Werte multipliziert wurden. z.B. L21: Lowbyte von B2 * A1
707         ; Dabei gilt:
708         ;
709         ; In R1: P1 <= H11 + carry from P2
710         ; In R2: P2 <= H21 + H12 + L11 + carry from P3
711         ; In R3: P3 <= H22 + L21 + L12
712         ; In R4: P4 <= L22
713
714         ; Fallunterscheidung:
715         ; 1) beide Zahlen positiv: normale multiplikation
716         ; 2) A positiv, B negativ --> comp(B), Ergebnis komplementieren ( + * - = -)
717         ; 3) A negativ, B positiv --> comp(A), Ergebnis komplementieren ( + * - = -)
718         ; 4) A negativ, B negativ --> comp(A) und comp(B), Ergebnis nicht komplementieren ( - * - = +)
719
720         ; -- Schauen, ob Zahl negativ -- ;
721         ; Vorzeichen der Zahl A ueberpruefen
722         MOV A, MUL_A_H
723         RL A                ; MSB zu LSB rotieren
724         ANL A, #00000001b   ; Nur Vorzeichenbit betrachten
725         MOV R5, A           ; Zwischenspeichern
726
727         ; Vorzeichen der Zahl B ueberpruefen
728         ; Vorgehen analog zu A
729         MOV A, MUL_B_H
730         RL A
731         ANL A, #00000001b
732         MOV R6, A
733
734         ; Teste das Vorzeichen von A
735         MOV A, R5
736         JNZ mul_A_neg   ; Springen, falls A negativ
737
738         ; Teste das Vorzeichen von B
739         MOV A, R6
740         JNZ mul_B_neg   ; Springen, falls B negativ
741
742         LJMP mul_calc
743
744         mul_A_neg:
745             ; Komplement von A bilden
746             MOV comp_H, MUL_A_H
747             MOV comp_L, MUL_A_L
748
749             LCALL comp
750
751             MOV MUL_A_H, comp_H
752             MOV MUL_A_L, comp_L
753
754             ; Ueberpruefen, ob auch B negativ ist
755             MOV A, R6
756             JNZ mul_B_neg   ; Springen, falls B negativ
757
758             ; Zur Berechnung springen
759             LJMP mul_calc
760
761         mul_B_neg:
762             ; Komplement von B bilden
763             MOV comp_H, MUL_B_H
```

```
764           MOV comp_L, MUL_B_L
765
766           LCALL comp
767
768           MOV MUL_B_H, comp_H
769           MOV MUL_B_L, comp_L
770
771       ; -- Berechnung von A * B -- ;
772       mul_calc:
773           ; Berechung von A2 * B2 (Low-Bytes)
774           MOV A, MUL_A_L
775           MOV B, MUL_B_L
776           MUL AB          ; L22 in A, H22 in B
777
778           ; Werte zwischenspeichern
779           MOV R4,A    ; L22 in R4
780           MOV R3,B    ; H22 in R3
781
782           ; Berechnung von A1 * B2 (High-Byte mit Low-Byte)
783           MOV A, MUL_B_L
784           MOV B, MUL_A_H
785           MUL AB          ; L21 in A, H21 in B
786
787           ; Speichern
788           MOV R2, B   ; H21 in R2
789
790           ; L21 auf P3 addieren
791           ADD A, R3
792
793           ; Zwischenspeichern
794           MOV R3, A
795
796           ; Berechnung von A2 * B1 (Low-Byte mit High-Byte)
797           MOV A, MUL_B_H
798           MOV B, MUL_A_L
799           MUL AB          ; L12 in A, H12 in B
800
801           ; L12 auf R3 addieren und zurueckschreiben
802           CLR C
803           ADD A, R3
804           MOV R3, A
805
806           ; H12 + R2 (inklusive Carry)
807           MOV A, B
808           ADDC A, R2
809           MOV R2, A
810           CLR C
811
812           ; A1 * B1 (High-Bytes)
813           MOV A, MUL_B_H
814           MOV B, MUL_A_H
815           MUL AB          ; L11 in A, H11 in B
816
817           ; L11 auf R2 aufaddieren und zurueckschreiben
818           CLR C
819           ADD A, R2
820           MOV R2, A
821
822           ; L11 + carry
823           MOV A, B
824           ADDC A, #0
825           MOV R1, A
826
827           ; -- Zurueckbringen in ursprüngliche Form -- ;
828           ; Entfernen der hinteren 10 Nachkommastellen und der ersten 6 Vorkommastellen (ignorieren
      der Register)
829           ; Anschliessend um zwei nach rechts rotieren
830           CLR C
831
832           MOV B, #2d
833           rotate_r2r:
834               ; Zuerst High-Byte ueber Carry rotieren
835               MOV A, R2
836               RRC A
837               MOV R2, A
838
839               ; Low-Byte inklusive Carry rotieren
```

```
840                 MOV A, R3
841                 RRC A
842                 MOV R3, A
843 
844                 ; Carry zuruecksetzen
845                 CLR C
846                 DJNZ B, rotate_r2r
847 
848             ; -- Zurueckschreiben an urspruengliche Speicheradressen von A -- M
849             MOV MUL_A_H, R2
850             MOV MUL_A_L, R3
851 
852             ; -- Flippe Ergebnis, falls - * + -- ;
853             ; XOR, da - * - = +
854             MOV A, R6
855             XRL A, R5
856             JNZ flipResult
857 
858             ; Zurueck
859             RET
860 
861             flipResult:
862                 ; Komplement des Ergebnisses bilden
863                 MOV comp_H, MUL_A_H
864                 MOV comp_L, MUL_A_L
865 
866                 LCALL comp
867 
868                 MOV MUL_A_H, comp_H
869                 MOV MUL_A_L, comp_L
870 
871                 ; Zurueck
872                 RET
873 
874         ; ------------------------------------------------ ;
875 
876 
877 
878         ; -- [Berechnung des Zweierkomplements der gesamten 16Bit Zahl] -- ;
879     comp:
880         ; Invertieren des Low-Bytes
881         MOV A, comp_L
882         CPL A
883         ; 1 addieren
884         ADD A, #1d
885         MOV comp_L, A
886 
887         ; Invertieren des High-Bytes
888         ; Uebertrag aus Low-Byte addieren
889         MOV A, comp_H
890         CPL A
891         ADDC A, #0d
892         MOV comp_H, A
893 
894         RET
895 
896         ; ------------------------------------------------ ;
897 
898 
899 
900         ; -- [Berechung eines ASCII - Zeichens abhaengig von n] -- ;
901     calc_ascii:
902         ; n liegt in Register R7
903         MOV A, R7
904 
905         ; Zuerst ueberpruefen, ob n = Nmax gilt
906         SUBB A, #Nmax
907         CLR C
908 
909         ; ACC = 0, wenn n = Nmax
910         JZ set_ascii_nmax
911 
912         ; ACC != 0
913         ; -> Mod 8 rechnen
914         MOV A, R7
915         MOV B, #8d
916         DIV AB
```

```
917
918         ; A enthaelt Ergebnis der Division, B den Rest -> Rest in A schreiben
919         MOV A, B
920
921         ; Nun wird der für den gegebenen Rest das jeweilige UP aufgerufen, welches das ASCII-codierte
    Zeichen in R7 schreibt
922         ; und danach das UP aufruft, welches das ASCII-Zeichen auf die serielle Schnittstelle schreibt
923
924         ; Rest 0
925         JZ set_ascii_mod0
926
927         ; Rest > 0
928         SUBB A, #1
929         JZ set_ascii_mod1
930
931         ; Rest > 1
932         SUBB A, #1
933         JZ set_ascii_mod2
934
935         ; Rest > 2
936         SUBB A, #1
937         JZ set_ascii_mod3
938
939         ; Rest > 3
940         SUBB A, #1
941         JZ set_ascii_mod4
942
943         ; Rest > 4
944         SUBB A, #1
945         JZ set_ascii_mod5
946
947         ; Rest > 5
948         SUBB A, #1
949         JZ set_ascii_mod6
950
951         ; Rest > 6
952         SUBB A, #1
953         JZ set_ascii_mod7
954
955         set_ascii_mod0:
956             MOV R7, #164d
957             LJMP write_ascii
958
959         set_ascii_mod1:
960             MOV R7, #43d
961             LJMP write_ascii
962
963         set_ascii_mod2:
964             MOV R7, #169d
965             LJMP write_ascii
966
967         set_ascii_mod3:
968             MOV R7, #45d
969             LJMP write_ascii
970
971         set_ascii_mod4:
972             MOV R7, #42d
973             LJMP write_ascii
974
975         set_ascii_mod5:
976             MOV R7, #64d
977             LJMP write_ascii
978
979         set_ascii_mod6:
980             MOV R7, #183d
981             LJMP write_ascii
982
983         set_ascii_mod7:
984             MOV R7, #174d
985             LJMP write_ascii
986
987         set_ascii_nmax:
988             MOV R7, #32d
989
990         ; ------------------------------------------------ ;
991
992
```

```
993
994         ; -- [Schreiben von ASCII Wert auf die serielle Schnittstelle] -- ;
995     write_ascii:
996         ; -- Konfiguration der Schnittstelle -- ;
997         ; - Interface 0
998         ; - 1 Startbit, 8 Datenbit, 1 Stopbit
999         ; - keine Parität und kein Handshaking
1000        ; - Baudrate 28800 1/s (XTAL: 12 MHz)
1001
1002        ; S0CON einstellen
1003        ; SM0  = 0,
1004        ; SM1  = 1 -> Mode 1: 8-Bit, var. Baud
1005        ; SM20 = 0 -> RI0 wird aktiviert
1006        ; REN0 = 1 -> Receiver enable
1007        ; TB80 = 0 -> kein 9. Datenbit
1008        ; RB80 = 0 -> Stoppbit
1009        ; TI0  = 0 -> Transmitter interrupt Flag
1010        ; RI0  = 0 -> Receiver interrupt Flag
1011        ; -> Bitfolge ergibt 80 (50h)
1012        MOV S0CON, #80d
1013
1014        ; Baudrate: 2^(SMOD)*OszilatorFrequenz/(64*(1024 - S0REL)) = 28800
1015        ; smod=1, s0rel= 0xfff3
1016        ; -> Baudrate 28846
1017
1018        ; SMOD aktivieren
1019        MOV PCON, #1000$0000b
1020
1021        ; S0REL (S0RELH | S0RELL) einstellen
1022        MOV S0RELL, #0xf3
1023        MOV S0RELH, #0xff
1024
1025        ; Baudgenerator benutzen
1026        SETB BD
1027
1028        ; -- Senden des ASCII Byte -- ;
1029        ; Zeichen liegt in R7
1030        MOV S0BUF, R7
1031
1032        ; Warte auf TI0
1033        wait_for_send:
1034            JNB TI0, wait_for_send
1035
1036        ; Nachrichte wurde gesendet
1037        ; Entferne TI0 Flag
1038        ANL S0CON, #1111$1101
1039
1040        ; -- Springe zurueck zum Hauptprogramm -- ;
1041        RET
1042
1043        ; ------------------------------------------------ ;
1044
1045
1046        ; -- [Beenden des Programms] -- ;
1047    finish:
1048
1049        NOP
1050        NOP
1051
1052        END
```