

033D

TITLE ZEICHENAUSGABE - Programm 1

```

;*****
;Dieses Programm gibt ASCII-Zeichen auf dem Bildschirm aus
;und kehrt dann ins Betriebssystem zurück
;*****

```

```

;*****
;Definition von Systemadressen
;*****

```

```

SYSTEM EQU 00005H      ;Adresse für Aufruf von
                        ;Systemfunktionen

```

```

;*****
;Definition von Systemfunktionen
;*****

```

```

WSTART EQU 0           ;Warmstart
CONAUSF EQU 2          ;Ausgabe auf CON: (=Konsole)

```

```

;*****
;Definition von ASCII-Konstanten
;*****

```

```

CR EQU 0DH             ;Cursor an Zeilenanfang
                        ;(Carriage Return)
LF EQU 0AH             ;Cursor in neue Zeile
                        ;(Line Feed)

```

```

;*****
;Hauptprogramm ZEICHENAUSGABE
;*****

```

```

ORG 0100H              ;Das Programm beginnt bei
                        ;0100H

```

```

START:
    LD  E, 'T'          ;Es werden die Zeichen
    CALL ZEICHEN         ;T, e, x und t geschrieben

```

```

    LD  E, 'e'
    CALL ZEICHEN

```

```

    LD  E, 'x'
    CALL ZEICHEN

```

```

    LD  E, 't'
    CALL ZEICHEN

```

```

    CALL NEUE_ZEILE      ;Der Cursor wird in eine

```

;neue Zeile gesetzt

```
LD    C,WSTARTF
CALL  SYSTEM
```

;Die Warmstartfunktion
;wird aufgerufen

```
;*****
;
;Unterprogramm ZEICHEN
; - schreibt das Zeichen im E-Register auf den Bildschirm
;*****
```

```
ZEICHEN:
LD    C,CONAUSE
CALL  SYSTEM
RET
```

;Der Funktionscode wird ins
;C-Register geladen
;und das Betriebssystem
;aufgerufen
;Rücksprung

```
;*****
;
;Unterprogramm NEUE ZEILE
; - fängt auf dem Bildschirm eine neue Zeile an
;*****
```

```
NEUE ZEILE:
LD    E,CR
CALL  ZEICHEN
LD    E,LF
CALL  ZEICHEN
RET
```

;Der Cursor wird an den
;Anfang der Zeile...
;... und dann eine Zeile
;tiefer gesetzt
;Rücksprung

```
;*****
;
;Programmende
;*****
```

```
END    START
```

;Der Programmzähler wird
;auf den Programmanfang
;gesetzt

TITLE TEXTAUSGABE - Programm 2

```

;*****
;Dieses Programm gibt ASCII-Zeichen und im Speicher abgelegte
;Strings auf dem Bildschirm aus
;und kehrt dann ins Betriebssystem zurück
;*****

```

```

;*****
;Definition von Systemadressen
;*****

```

```

SYSTEM EQU    00005H      ;Adresse für Aufruf von
                          ;Systemfunktionen

```

```

;*****
;Definition von Systemfunktionen
;*****

```

```

WSTARTF EQU    0          ;Warmstart

```

```

CONAUSF EQU    2          ;Ausgabe auf CON: (=Konsole)

```

```

STRAUSF EQU    9          ;Ausgabe von Strings auf CON:

```

```

;*****
;Definition von ASCII-Konstanten
;*****

```

```

CR      EQU    0DH        ;Cursor an Zeilenanfang
                          ;(Carriage Return)
LF      EQU    0AH        ;Cursor in neue Zeile
                          ;(Line Feed)

```

```

STOP    EQU    '$'        ;Markierung für Stringende

```

```

;*****
;Hauptprogramm TEXTAUSGABE
;*****

```

```

ORG     100H              ;Das Programm beginnt bei
                          ;0100H

```

START:

```

LD      E, 'T'            ;Es werden die Zeichen
CALL    ZEICHEN            ;T, e. x und t geschrieben

```

```

LD      E, 'e'
CALL    ZEICHEN

```

```

LD      E, 'x'
CALL    ZEICHEN

```

```

LD      E,'t'
CALL    ZEICHEN

LD      DE,STRING1      ;Der unten definierte String
CALL    STRING          ;wird ausgegeben

CALL    NEUE_ZEILE      ;Der Cursor wird in eine
                        ;neue Zeile gesetzt

LD      C,WSTARTF       ;Die Warmstartfunktion
CALL    SYSTEM          ;wird aufgerufen

```

```

STRING1:
        DB      'verarbeitung'
        DB      CR,LF
        DB      'mit ZEAT'
        DB      STOP

```

```

;eine Leerzeile
;Stringende

```

```

;*****
;
;Unterprogramm ZEICHEN
; - schreibt das Zeichen im E-Register auf den Bildschirm
;*****

```

ZEICHEN:

```

LD      C,CONAUSF       ;Der Funktionskode wird ins
                        ;C-Register geladen
CALL    SYSTEM          ;und das Betriebssystem
                        ;aufgerufen

RET                        ;Rücksprung

```

```

;*****
;
;Unterprogramm NEUE_ZEILE
; - fängt auf dem Bildschirm eine neue Zeile an
;*****

```

NEUE_ZEILE:

```

LD      E,CR            ;Der Cursor wird an den
CALL    ZEICHEN         ;Anfang der Zeile...

LD      E,LF            ;... und dann eine Zeile
CALL    ZEICHEN         ;tiefer gesetzt

RET                        ;Rücksprung

```

```

;*****
;
;Unterprogramm STRING
; - schreibt den String auf den der Pointer DE zeigt
;*****

```


STRING:

```

LD      C,STRAUSF      ;Der Funktionskode wird ins
                        ;C-Register geladen
CALL    SYSTEM         ;und das Betriebssystem
                        ;aufgerufen

RET                                ;Rücksprung

```

```

;*****
;
;Programmende
;*****

```

```

END      START          ;Der Programmzähler wird
                        ;auf den Programmanfang
                        ;gesetzt

```

TITLE TEXTSPEICHER - Programm 3

```

;*****
;Dieses Programm gibt den Inhalt des TLC90 Textspeichers
;auf dem Bildschirm aus
;und kehrt dann ins Betriebssystem zurück
;*****

```

```

;*****
;Definition von Systemadressen
;*****

```

```

SYSTEM EQU      00005H      ;Adresse für Aufruf von
                        ;Systemfunktionen

```

```

;*****
;Definition von Systemfunktionen
;*****

```

```

WSTARTF EQU      0          ;Warmstart

CONAUSF EQU      2          ;Ausgabe auf CON: (=Konsole)

STRAUSF EQU      9          ;Ausgabe von Strings auf CON:

TXTBEGF EQU      247        ;Startadresse des Text-
                        ;speichers abfragen

```

```

;*****
;Definition von ASCII-Konstanten
;*****

```

```

CR      EQU      0DH      ;Cursor an Zeilenanfang
                          ;(Carriage Return)
LF      EQU      0AH      ;Cursor in neue Zeile
                          ;(Line Feed)
EOF     EQU      1AH      ;Markierung für Textende
                          ;(End Of File)
STOP    EQU      '$'      ;Markierung für Stringende

```

```

;*****
;Hauptprogramm TEXTAUSGABE
;*****

```

```

                ORG      100H      ;Das Programm beginnt bei
                                   ;0100H

```

START:

```

                LD      DE,MELDUNG      ;Anfangsmeldung
                CALL    STRING
                CALL    NEUE_ZEILE      ;Der Cursor wird in eine
                                   ;neue Zeile gesetzt
                LD      C,TXTBEGF      ;Der Anfang des Textspeichers
                CALL    SYSTEM          ;wird ins HL-Register geholt

```

SCHLEIFE:

```

                LD      A,(HL)          ;Ein Zeichen ins A-Register..
                INC     HL              ;..und Pointer HL erhöhen
                CP      EOF            ;Ist es END OF FILE ?
                JR      Z,ENDE          ; Ja: nach ENDE springen
                LD      E,A            ; Nein: ins E-Register laden,
                CALL    ZEICHEN        ; ausgeben und...
                JR      SCHLEIFE       ; ... nächstes Zeichen holen

```

ENDE:

```

                CALL    NEUE_ZEILE
                LD      C,WSTARTF      ;Die Warmstartfunktion
                CALL    SYSTEM          ;wird aufgerufen

```

MELDUNG:

DB CR,LF,'Inhalt des Textspeichers:',STOP

```
*****
;
;Unterprogramm ZEICHEN
; - schreibt das Zeichen im E-Register auf den Bildschirm
;*****
```

ZEICHEN:

```
LD C,CONAUSF ;Der Funktionskode wird ins
               ;C-Register geladen
CALL SYSTEM   ;und das Betriebssystem
               ;aufgerufen

RET           ;Rücksprung
```

```
*****
;
;Unterprogramm NEUE ZEILE
; - fängt auf dem Bildschirm eine neue Zeile an
;*****
```

NEUE_ZEILE:

```
LD E,CR        ;Der Cursor wird an den
CALL ZEICHEN    ;Anfang der Zeile...

LD E,LF        ;... und dann eine Zeile
CALL ZEICHEN    ;tiefer gesetzt

RET            ;Rücksprung
```

```
*****
;
;Unterprogramm STRING
; - schreibt den String auf den der Pointer DE zeigt
;*****
```

STRING:

```
LD C,STRAUSF   ;Der Funktionskode wird ins
               ;C-Register geladen
CALL SYSTEM    ;und das Betriebssystem
               ;aufgerufen

RET            ;Rücksprung
```

```
*****
;
;Programmende
;*****
END START      ;Der Programmzähler wird
               ;auf den Programmanfang
               ;gesetzt
```

TITLE TEXTSPEICHER - Verbessertes Programm 3

```

;*****
;Dieses Programm gibt den Inhalt des Textspeichers
;auf dem Bildschirm aus
;und kehrt dann ins Betriebssystem zurück
;*****

;*****
;Definition von Systemadressen
;*****

SYSTEM EQU    00005H           ;Adresse für Aufruf von
                                ;Systemfunktionen

;*****
;Definition von Systemfunktionen
;*****

WSTARTF EQU    0                ;Warmstart

CONAUSF EQU    2                ;Ausgabe auf CON: (=Konsole)

STRAUSF EQU    9                ;Ausgabe von Strings auf CON:

TXTBEGF EQU    247              ;Startadresse des Text-
                                ;speichers abfragen

;*****
;Definition von ASCII-Konstanten
;*****

CR      EQU    0DH              ;Cursor an Zeilenanfang
                                ;(Carriage Return)
LF      EQU    0AH              ;Cursor in neue Zeile
                                ;(Line Feed)
EOF     EQU    1AH              ;Markierung für Textende
                                ;(End Of File)
STOP    EQU    '$'              ;Markierung für Stringende

;*****
;Hauptprogramm TEXTAUSGABE
;*****

ORG     100H                    ;Das Programm beginnt bei
                                ;0100H

```

START:

```

CALL    STRING
DB      CR,LF,'Inhalt des Textspeichers:',STOP

CALL    NEUE_ZEILE      ;Der Cursor wird in eine
                        ;neue Zeile gesetzt

LD      C,TXTBEGF      ;Der Anfang des Textspeichers
CALL    SYSTEM          ;wird ins HL-Register geholt

```

SCHLEIFE:

```

LD      A,(HL)          ;Ein Zeichen ins A-Register..
INC     HL              ;..und Pointer HL erhöhen

CP      EOF             ;Ist es END OF FILE ?
JR      Z,ENDE          ; Ja: nach ENDE springen

LD      E,A             ; Nein: ins E-Register laden,
CALL    ZEICHEN          ; ausgeben und...

JR      SCHLEIFE        ; ... nächstes Zeichen holen

```

ENDE:

```

CALL    NEUE_ZEILE

LD      C,WSTARTF      ;Die Warmstartfunktion
CALL    SYSTEM          ;wird aufgerufen

```

```

;*****
;Unterprogramm ZEICHEN
; - schreibt das Zeichen im E-Register auf den Bildschirm
;*****

```

ZEICHEN:

```

LD      C,CONAUSF      ;Der Funktionscode wird ins
                        ;C-Register geladen,
PUSH    HL              ;HL gerettet
CALL    SYSTEM          ;und das Betriebssystem
                        ;aufgerufen
POP     HL              ;wiederherstellen
RET                     ;Rücksprung

```

```

;*****
;Unterprogramm NEUE ZEILE
; - fängt auf dem Bildschirm eine neue Zeile an
;*****

```

NEUE_ZEILE:

```

LD      E,CR      ;Der Cursor wird an den
CALL    ZEICHEN    ;Anfang der Zeile...

LD      E,LF      ;... und dann eine Zeile
CALL    ZEICHEN    ;tiefer gesetzt

RET      ;Rücksprung

```

```

;*****
;Unterprogramm STRING
; - schreibt den String, der nach dem aufrufenden Programm
;   steht
;*****

```

STRING:

```

POP      DE      ;Rückkehradresse ins DE-
                ;Registerpaar holen und
PUSH     DE      ;wieder auf den Stack zurück-
                ;bringen - DE zeigt jetzt auf
                ;den Stringanfang

LD       C,STRAUSF ;Der Funktionscode wird ins
                ;C-Register geladen
CALL     SYSTEM    ;und das Betriebssystem
                ;aufgerufen

POP      HL      ;Rückkehradresse nochmal vom
                ;Stack holen - jetzt zeigt HL
                ;auf den Stringanfang
LD       BC,OFFFH ;Zähler BC auf Maximalwert,
LD       A,STOP   ;Stringendezeichen in Reg. A
CPIR     ;und Stringende suchend
                ;... HL zeigt nun auf die
                ;Adresse nach dem Stringende
JP       (HL)     ;diese Adresse anspringen

```

```

;*****
;Programmende
;*****

```

```

END      START    ;Der Programmzähler wird
                ;auf den Programmanfang
                ;gesetzt

```

TITLE STATISTIK - Programm 4

```

;*****
;Dieses Programm zählt die Häufigkeit eines bestimmten Buch-
;stabens im Textspeicher
;und kehrt dann ins Betriebssystem zurück
;*****

```

```

;*****
;Definition von Systemadressen
;*****

```

```
WSTART EQU 00000H ;Adresse für Warmstart
```

```
SYSTEM EQU 00005H ;Adresse für Aufruf von
;Systemfunktionen
```

```

;*****
;Definition von Systemfunktionen
;*****

```

```
WSTARTF EQU 0 ;Warmstart
```

```
CONEINF EQU 1 ;Eingabe von CON: (=Tastatur)
```

```
CONAUSF EQU 2 ;Ausgabe auf CON: (=Konsole)
```

```
STRAUSF EQU 9 ;Ausgabe von Strings auf CON:
```

```
PRBYTEF EQU 240 ;Ausgabe eines 'BYTE' als
;zweistellige Sedezimalzahl.
```

```
PRWORDF EQU 241 ;Ausgabe eines 'WORD' als
;vierstellige Sedezimalzahl
```

```
TXTBEGF EQU 247 ;Startadresse des Text-
;speichers abfragen
```

```

;*****
;Definition von ASCII-Konstanten
;*****

```

```
CR EQU 0DH ;Cursor an Zeilenanfang
;(Carriage Return)
```

```
LF EQU 0AH ;Cursor in neue Zeile
;(Line Feed)
```

```
EOF EQU 1AH ;Markierung für Textende
;(End Of File)
```

```
STOP EQU '$' ;Markierung für Stringende
```

```

;*****
;Hauptprogramm STATISTIK
;*****

```

```

      ORG      100H          ;Das Programm beginnt bei
                           ;0100H

```

START:

```

      CALL     STRING
      DB       CR,LF,'Zeichenstatistik: '
      DB       'Bitte Taste drücken - ',STOP

```

```

      CALL     EINGABE      ;ein Zeichen von der
                           ;Tastatur holen, Zeichen
                           ;ist im Register A

```

```

      CP       'C'-40H      :Ist es 'C' ?
      JP       Z,WSTART     ; Ja: Rückkehr ins Betriebs-
                           ; system
                           ; Nein: weiter..

```

```

      CALL     STATISTIK    ;auswerten

```

```

      JR       START       ;Schleife - kann durch
                           ;Eingabe von 'C' abgebrochen
                           ;werden

```

```

;*****
;Unterprogramm EINGABE
; - holt ein Zeichen von der Tastatur ins Register A und
;   schreibt es auf den Bildschirm
;*****

```

EINGABE:

```

      LD       C,CONEINF    ;Der Funktionscode wird ins
                           ;C-Register geladen
      CALL     SYSTEM       ;und das Betriebssystem
                           ;aufgerufen
      RET                      ;Rücksprung

```

```

;*****
;Unterprogramm STATISTIK
; - Der Textspeicher wird nach dem Zeichen im A-Register
;   durchsucht und die Häufigkeit gezählt.
;*****

```

STATISTIK:

```

      PUSH     AF           ;Das zu suchende Zeichen
                           ;wird ins Stack gerettet,

```



```

CALL    NEUE_ZEILE    ;eine neue Zeile begonnen

LD      C,TXTBEGF    ;und der Anfang des Texts-
                  ;peichers ins HL-Register-
CALL    SYSTEM        ;paar geholt

POP     AF            ;Das Zeichen wird ins
LD      C,A           ;A- und C-Register geholt
LD      DE,0          ;und der Zähler (Register-
                  ;paar DE) auf Null gesetzt

```

SCHLEIFE:

```

LD      A,(HL)        ;Ein Zeichen ins A-Register..
INC     HL            ;..und Pointer HL erhöhen

CP      EOF           ;Ist es END OF FILE ?
JR      Z,ERGENIS     ; Ja: nach ERGENIS springen
                  ; Nein: weiter...

CP      C             ; Ist es das gesuchte Zeichen'
JR      NZ,SCHLEIFE   ; Nein: nächstes Zeichen

INC     DE            ; Ja: Zähler erhöhen und
JR      SCHLEIFE      ;   nächstes Zeichen

```

ERGENIS:

```

PUSH    DE            ;Zählerstand und
PUSH    BC            ;gesuchtes Zeichen auf Stack

CALL    STRING
DB      'Zeichen ',STOP

POP     DE            ;gesuchtes Zeichen von Stack
PUSH    DE            ;nach Register E, wieder
                  ;zurück auf Stack

CALL    ZEICHEN        ;und ausgeben

CALL    STRING
DB      ' (=ASCII ',STOP

POP     DE            ;gesuchtes Zeichen von Stack
                  ;wieder ins Register E und
LD      C,PRBYTEF     ;ASCII-Code als Sedezimal-
CALL    SYSTEM        ;zahl ausgeben

CALL    STRING
DB      'H) kommt ',STOP

POP     DE            ;Anzahl von Stack ins
LD      C,PRWORDF     ;Registerpaar DE und als
CALL    SYSTEM        ;Sedezimalzahl ausgeben

```

```
CALL    STRING
DB      'H mal vor',CR,LF,STOP
```

```
RET                                ;fertig: Rücksprung
```

```
;*****
;
;Unterprogramm ZEICHEN
; - schreibt das Zeichen im E-Register auf den Bildschirm
;*****
```

ZEICHEN:

```
LD      C,CONAUSF                ;Der Funktionscode wird ins
                                ;C-Register geladen,
PUSH    HL                      ;HL gerettet
CALL    SYSTEM                  ;und das Betriebssystem
                                ;aufgerufen
POP      HL                     ;wiederherstellen
RET      ;Rücksprung
```

```
;*****
;
;Unterprogramm NEUE_ZEILE
; - fängt auf dem Bildschirm eine neue Zeile an
;*****
```

NEUE_ZEILE:

```
LD      E,CR                    ;Der Cursor wird an den
CALL    ZEICHEN                 ;Anfang der Zeile...

LD      E,LF                    ;... und dann eine Zeile
CALL    ZEICHEN                 ;tiefer gesetzt

RET                                ;Rücksprung
```

```
;*****
;
;Unterprogramm STRING
; - schreibt den String, der nach dem aufrufenden Programm
; steht
;*****
```

STRING:

```
POP      DE                    ;Rückkehradresse ins DE-
                                ;Registerpaar holen und
PUSH     DE                    ;wieder auf den Stack zurück-
                                ;bringen - DE zeigt jetzt auf
                                ;den Stringanfang
```

```
LD      C,STRAUSEF      ;Der Funktionscode wird ins
                        ;C-Register geladen
CALL    SYSTEM          ;und das Betriebssystem
                        ;aufgerufen

POP      HL              ;Rückkehradresse nochmal vom
                        ;Stack holen - jetzt zeigt HL
                        ;auf den Stringanfang
LD       BC,OFFFHH       ;Zähler BC auf Maximalwert,
LD       A,STOP          ;Stringendezeichen in Reg. A
CPIR                      ;und Stringende suchend
                        ;... HL zeigt nun auf die
                        ;Adresse nach dem Stringende
JP       (HL)            ;diese Adresse anspringen
```

```
;*****
;Programmende
;*****
```

```
END      START          ;Der Programmzähler wird
                        ;auf den Programmanfang
                        ;gesetzt
```

```
;*****
```

```
;Hauptprogramm AUTOMATISCHE STATISTIK
```

```
; holt nacheinander alle ASCII-Zeichen von A bis z in das A-Register  
; und zählt ihre Häufigkeit.
```

```
;*****
```

```
ORG 100H ;das Programm beginnt bei 0100H
```

```
START:
```

```
LD A, 'A' ;erstes Zeichen
```

```
LOOP: PUSH AF ;Zeichen im Stack verwahren  
CALL STATISTIK ;auswerten  
POP AF ;Zeichen zurückholen
```

```
INC A ;nächstes Zeichen holen
```

```
CP 'ß'+1 ;ist es noch druckbar?
```

```
JR NZ, LOOP ;ja: Auswerten
```

```
JP WSTART ;nein: Ende, Warmstart
```

```
;*****
```

```
;Unterprogramm STATISTIK
```

```
; Der Textspeicher wird nach dem Zeichen im A-Register durchsucht und  
; die Häufigkeit gezählt.
```

```
;*****
```

```
PUSH AF
```

```
USW.
```

Programm 1 CONWAY's Life-Game

```

;*****
;Definition von Systemadressen
;*****

WSTART EQU    00000H      ;Adresse für Warmstart

SYSTEM EQU    00005H      ;Adresse für Aufruf von
                           ;Systemfunktionen

;*****
;Definition von Systemfunktionen
;*****

WSTARTF EQU    0          ;Warmstart

DIRCONF EQU    6          ;direkte Konsol Ein-/Ausgabe

TXTBEGF EQU    247        ;Startadresse des Text-
                           ;speichers abfragen

;*****
;Definition von ASCII-Konstanten
;*****

CONTROL EQU    1FH        ;Erzeugung von Control-Codes
                           ;mit ... AND CONTROL
CR EQU    0DH            ;Cursor an Zeilenanfang
                           ;(Carriage Return)
LF EQU    0AH            ;Cursor in neue Zeile
                           ;(Line Feed)
EOF EQU    1AH           ;Markierung für Textende
                           ;(End Of File)
ESC EQU    1BH           ;Escape
                           ;(für Cursor-Positionierung)

;*****
;sonstige Konstanten
;*****

ZEILEN EQU    24          ;Anzahl Zeilen
SPALTEN EQU    80         ;Anzahl Spalten
GROESSE EQU    ZEILEN*SPALTEN ;Spielfeldgröße

;*****
;Hauptprogramm LIFE
;*****

ORG    100H              ;Das Programm beginnt bei
                           ;0100H

```

```

START:
      CALL  HOLE_FELD      ;Feld aus Textspeicher holen
      CALL  SCHREIBE_FELD  ;und auf Schirm schreiben

LIFE:
      CALL  DIRIN          ;Tastatureingabe?

      OR     A              ;kein Zeichen (also A=0)?
      JR     Z,LIFE        ;ja: warten
                          ;nein, Zeichen da:
      CP     'C' AND CONTROL ; Control-C ?
      JP     Z,WSTART      ; ja: Warmstart

      JR     LIFE          ;Schleife

```

```

;*****
;
;Unterprogramm HOLE_FELD
; - Holt vom Textspeicher eine Aufstellung in das interne
;   Spielfeld
;*****

```

```

HOLE_FELD:
      LD     C,XTTBEGF     ;Anfang des Textspeichers
      CALL  SYSTEM        ;ins HL-Registerpaar
      LD     DE,FELD       ; .. Spielfeld in DE
      LD     B,SPALTEN     ; und Feldgroesse in BC
      LD     C,ZEILEN

```

```

HOLE_WEITER:
      LD     A,(HL)        ;ein Zeichen holen
      INC    HL            ;Textzeiger weiter
      CP     LF            ;Zeichen = Line Feed?
      JR     Z,HOLE_WEITER ; ja: ignorieren
      CP     EOF           ;Zeichen = Textende?
      JR     Z,LOESCHE_REST ; ja: Feld löschen
      CP     CR            ;Zeichen = neue Zeile?
      JR     Z,NEUE_ZEILE  ; ja: in neue Zeile
      CP     ' '           ;Zeichen = Zwischenraum
      JR     Z,LOESCHE_EL  ; ja: Feldelement loeschen
      LD     A,1           ; nein: Feldelement:=1

```

```

SETZE_EL:
      LD     (DE),A        ;Feldelement setzen
      LD     A,B           ;Zeile schon voll,
      OR     A              ;d.h. Spaltenzähler auf Null?
      JR     Z,HOLE_WEITER ; ja: weiter (bis neue Zeile)
      DEC    B             ; nein: Spaltenzähler zurück
      INC    DE            ;      Feldzeiger weiter
      JR     HOLE_WEITER  ;      und weitermachen

```

```

LOESCHE_EL:
      LD     A,0
      JR     SETZE_EL

```

```

LOESCHE_REST:
    DEC    HL                ;Textzeiger zurück auf EOF,
                        ;dann NEUE_ZEILE
                        ;dadurch so oft NEUE_ZEILE bis
                        ;Feld voll

```

```

NEUE_ZEILE:                ;Rest der Zeile wird gelöscht
    LD     A,B              ;Spaltenzähler in A
    LD     B,SPALTEN        ;Spaltenzähler setzen

```

```

LOESCHE_ZEILE:
    OR     A                ;(alter) Spaltenzähler Null?
    JR     Z,ZEILE_VOLL     ; ja: Zeile schon voll
    EX     DE,HL
    LD     (HL),0           ; nein: Feldelement löschen
    EX     DE,HL
    INC    DE               ; Feldzeiger inkrementieren,
    DEC    A                ; Zähler dekrementieren
    JR     LOESCHE_ZEILE    ; und weiter löschen

ZEILE_VOLL:
    DEC    C                ;Zeilenzähler zurück, Null?
    RET    Z                ; ja: alle Zeilen voll, fertig
    JR     HOLE_WEITER     ; nein: weitermachen

```

```

;*****
;Unterprogramm SCHREIBE_FELD
; - Gibt das interne Spielfeld auf dem Bildschirm aus
;*****

```

```

SCHREIBE_FELD:
    LD     HL,0
    CALL   CURSOR           ;Cursor nach links oben
    LD     HL,FELD          ;Adresse Spielfeld in HL
    LD     BC,GROESSE-1     ;Groesse Spielfeld in BC

```

```

SCHREIBE_EL:
    LD     E,'*'           ;schreibe * falls gesetzt
    LD     A,(HL)           ;hole Feldelement
    OR     A                ;Element gesetzt (nicht 0)?
    JR     NZ,NICHT_LEER   ; ja: weiter, * lassen
    LD     E,' '           ; nein: Leerschritt statt *

```

```

NICHT_LEER:
    PUSH   BC               ;Zeichen ausgeben
    PUSH   HL
    CALL   DIROUT
    POP    HL
    POP    BC
    INC    HL               ;Feldzeiger erhöhen
    DEC    BC               ;Zähler dekrementieren
    LD     A,B              ;und prüfen, ob Null?
    OR     C
    JR     NZ,SCHREIBE_EL   ; nein: weiter
    RET                    ; ja: fertig

```

```

;*****
;
;Unterprogramm CURSOR
; - setzt den Cursor an Zeile L, Spalte H
;*****

```

CURSOR:

```

    PUSH    HL                ;Zeile/Spalte auf Stack
    LD      E,ESC            ;ESC ausgeben
    CALL    DIROUT
    LD      E,'='            ;= ausgeben
    CALL    DIROUT
    POP     HL                ;Zeile/Spalte von Stack
    PUSH    HL                ;holen und wieder zurück
    LD      A,L              ;Zeile ausgeben
    CALL    CURSOR1          ;          mit CURSOR1
    POP     HL                ;Zeile/Spalte von Stack
    LD      A,H              ;Spalte ausgeben

```

CURSOR1:

```

    ADD     A,' '            ;20H addieren
    LD      E,A              ;und dann ausgeben
    CALL    DIROUT
    RET

```

```

;*****
;
;Unterprogramme DIROUT und DIRIN
; - gibt das Zeichen im E-Register mit Systemfunktion 6 aus
;   oder holt Zeichen mit Systemfunktion 6 (DIRCONF)
;*****

```

DIRIN:

```

    LD      E,OFFH

```

DIROUT:

```

    LD      C,DIRCCNF
    CALL    SYSTEM
    RET

```

```

;*****
;
;Variable im RAM
;*****

```

```

FELD:    DS      GROESSE,0    ;Spielfeld

        END      START

```


Programm 2 CONWAY's Life-Game

```

;*****;
;Definition von Systemadressen
;*****;

WSTART EQU    00000H      ;Adresse für Warmstart

SYSTEM EQU    00005H      ;Adresse für Aufruf von
                          ;Systemfunktionen

;*****;
;Definition von Systemfunktionen
;*****;

WSTARTF EQU    0          ;Warmstart

DIRCONF EQU    6          ;direkte Konsole Ein-/Ausgabe

TXTBEGF EQU    247        ;Startadresse des Text-
                          ;speichers abfragen

;*****;
;Definition von ASCII-Konstanten
;*****;

CONTROL EQU    1FH        ;Erzeugung von Control-Codes
                          ;mit ... AND CONTROL
CR EQU    0DH            ;Cursor an Zeilenanfang
                          ;(Carriage Return)
LF EQU    0AH            ;Cursor in neue Zeile
                          ;(Line Feed)
EOF EQU    1AH           ;Markierung für Textende
                          ;(End Of File)
ESC EQU    1BH           ;Escape
                          ;(für Cursor-Positionierung)

;*****;
;sonstige Konstanten
;*****;

ZEILEN EQU    24          ;Anzahl Zeilen
SPALTEN EQU    80         ;Anzahl Spalten
GROESSE EQU    ZEILEN*SPALTEN ;Spielfeldgröße

;*****;
;Hauptprogramm LIFE
;*****;

ORG    100H              ;Das Programm beginnt bei
                          ;0100H

```

START:

```
CALL HOLE_FELD      ;Feld aus Textspeicher holen
CALL SCHREIBE_FELD ;und auf Schirm schreiben
```

LIFE:

```
CALL DIRIN          ;Tastatureingabe?

OR A                ;kein Zeichen (also A=0)?
JR Z,LIFE           ;ja: warten
                   ;nein, Zeichen da:
CP 'C' AND CONTROL ; Control-C ?
JP Z,WSTART         ; ja: Warmstart

CP CR              ; Carriage Return?
CALL Z,GENERATION

JR LIFE            ;Schleife
```

```
*****
;
;Unterprogramm GENERATION
; - Rechnet eine neue Generation aus
*****
```

```
VON_FELD:      DW FELD1
NACH_FELD:     DW FELD2
```

GENERATION:

```
LD HL,0            ;Cursor nach links oben
CALL CURSOR
LD IX,(VON_FELD)   ;IX := Zeiger Ausgangsfeld
LD IY,(NACH_FELD)  ;IY := Zeiger Ergebnisfeld
LD (VON_FELD),IY   ;Ausgangsfeld und Ergebnis-
LD (NACH_FELD),IX  ;feld für nächste Generation
                   ;tauschen
LD BC,GROESSE-1    ;BC := Anzahl der Feldelemente
                   ;die zu berechnen sind
```

GENERATION_1:

```
CALL PRUEFE_ELEMENT ;prüfe, ob Feldelement in
                   ;nächster Gen. besetzt ist
CALL SETZE_ELEMENT  ;setze Feldelement in Ergebnis-
                   ;feld entsprechend
CALL SCHREIBE_ELEMENT;schreibe * oder Leerzeichen
                   ;auf Schirm
INC IX              ;erhöhe Zeiger Ausgangsfeld
INC IY              ;erhöhe Zeiger Ergebnisfeld
DEC BC              ;dekrementiere Zähler
LD A,B              ;und prüfe, ob Null?
OR C
JR NZ,GENERATION_1 ; nein: weiter
RET                 ; ja: fertig
```

```

;*****
;Unterprogramm PRUEFE_ELEMENT
; - Prüft, ein einzelnes Feldelement in der nächsten
;   Generation belegt ist
;*****

```

PRUEFE_ELEMENT:

```

    LD      A,(IX)          ;Feldelement laden
    ADD     A,A             ;und 4 mal zu sich selbst
    ADD     A,A             ;addieren, d.h. mit 16
    ADD     A,A             ;multiplizieren
    ADD     A,A             ;dann Nachbarelemente addieren:
    ADD     A,(IX-81)       ;links oben
    ADD     A,(IX-80)       ;oben
    ADD     A,(IX-79)       ;rechts oben
    ADD     A,(IX-1)        ;links
    ADD     A,(IX+1)        ;rechts
    ADD     A,(IX+79)       ;links unten
    ADD     A,(IX+80)       ;unten
    ADD     A,(IX+81)       ;rechts unten

    LD      HL,REGELN       ;Adresse Regeltabelle nach HL
    LD      D,0             ;und Feldelement*16+Anzahl der
    LD      E,A             ;Nachbarn addieren
    ADD     HL,DE
    LD      A,(HL)          ;Tabellenwert laden
    OR      A               ;und verknüpfen für Flags
    RET

```

REGELN:

```

    DB      0,0,0,1,0,0,0,0 ;Spielregeln, erste Hälfte
    DB      0,0,0,0,0,0,0,0 ; (16 Byte) für bisher
                           ;leeres Feldelement

    DB      0,0,1,1,0,0,0,0 ;zweite für bisher belegtes
    DB      0,0,0,0,0,0,0,0 ;Feldelement

```

```

;*****
;Unterprogramm SETZE_ELEMENT
; - Setzt ein einzelnen Feldelement im neuen Feld
;*****

```

SETZE_ELEMENT:

```

    LD      (IX),A
    RET

```

```

;*****
;Unterprogramm SCHREIBE_ELEMENT
; - Schreibt ein einzelnes Feldelement auf Schirm
;*****

```

SCHREIBE_ELEMENT:

```

        LD      E, '*'
        OR      A
        JR      NZ, NO_SPACE
        LD      E, ' '
NO_SPACE:
        PUSH    IX
        PUSH    IY
        PUSH    BC
        CALL    DIROUT
        POP     BC
        POP     IY
        POP     IX
        RET

```

```

;*****
;Unterprogramm HOLE_FELD
; - Holt vom Textspeicher eine Aufstellung in das interne
;   Spielfeld
;*****

```

HOLE_FELD:

```

        LD      C, TXTBEGF      ;Anfang des Textspeichers
        CALL    SYSTEM          ;ins HL-Registerpaar
        LD      DE, (VON_FELD)  ; .. Spielfeld in DE
        LD      B, SPALTEN      ; und Feldgroesse in BC
        LD      C, ZEILEN

```

HOLE_WEITER:

```

        LD      A, (HL)         ;ein Zeichen holen
        INC     HL              ;Textzeiger weiter
        CP      LF              ;Zeichen = Line Feed?
        JR      Z, HOLE_WEITER  ; ja: ignorieren
        CP      EOF             ;Zeichen = Textende?
        JR      Z, LOESCHE_REST ; ja: Feld löschen
        CP      CR              ;Zeichen = neue Zeile?
        JR      Z, NEUE_ZEILE    ; ja: in neue Zeile
        CP      ' '             ;Zeichen = Zwischenraum
        JR      Z, LOESCHE_EL    ; ja: Feldelement loeschen
        LD      A, 1            ; nein: Feldelement:=1

```

SETZE_EL:

```

        LD      (DE), A         ;Feldelement setzen
        LD      A, B            ;Zeile schon voll,
        OR      A               ;d.h. Spaltenzähler auf Null?
        JR      Z, HOLE_WEITER  ; ja: weiter (bis neue Zeile)
        DEC     B               ; nein: Spaltenzähler zurück
        INC     DE              ;      Feldzeiger weiter
        JR      HOLE_WEITER     ;      und weitermachen

```

```

LOESCHE_EL:
    LD    A,0
    JR    SETZE_EL
LOESCHE_REST:
    DEC    HL                ;Textzeiger zurück auf EOF,
                            ;dann NEUE_ZEILE
                            ;dadurch so oft NEUE_ZEILE bis
                            ;Feld voll
NEUE_ZEILE:
                            ;Rest der Zeile wird gelöscht
    LD    A,B                ;Spaltenzähler in A
    LD    B,SPALTEN          ;Spaltenzähler setzen
LOESCHE_ZEILE:
    OR    A                ;(alter) Spaltenzähler Null?
    JR    Z,ZEILE_VOLL      ; ja: Zeile schon voll
    EX    DE,HL
    LD    (HL),0            ; nein: Feldelement löschen
    EX    DE,HL
    INC    DE                ; Feldzeiger inkrementieren,
    DEC    A                ; Zähler dekrementieren
    JR    LOESCHE_ZEILE     ; und weiter loeschen
ZEILE_VOLL:
    DEC    C                ;Zeilenzähler zurück, Null?
    RET    Z                ; ja: alle Zeilen voll, fertig
    JR    HOLE_WEITER      ; nein: weitermachen

```

```

;*****
;Unterprogramm SCHREIBE_FELD
;- Gibt das interne Spielfeld auf dem Bildschirm aus
;*****

```

```

SCHREIBE_FELD:
    LD    HL,0
    CALL  CURSOR            ;Cursor nach links oben
    LD    HL,(VON_FELD)     ;Adresse Spielfeld in HL
    LD    BC,GROESSE-1     ;Groesse Spielfeld in BC
SCHREIBE_EL:
    LD    E,'*'            ;schreibe * falls gesetzt
    LD    A,(HL)            ;hole Feldelement
    OR    A                ;Element gesetzt (nicht 0)?
    JR    NZ,NICHT_LEER    ; ja: weiter, * lassen
    LD    E,' '            ; nein: Leerschritt statt *
NICHT_LEER:
    PUSH  BC                ;Zeichen ausgeben
    PUSH  HL
    CALL  DIROUT
    POP   HL
    POP   BC
    INC   HL                ;Feldzeiger erhöhen
    DEC   BC                ;Zähler dekrementieren
    LD    A,B              ;und prüfen, ob Null?
    OR    C
    JR    NZ,SCHREIBE_EL   ; nein: weiter
    RET                    ; ja: fertig

```

```

;*****
;Unterprogramm CURSOR
; - setzt den Cursor an Zeile L, Spalte H
;*****

```

CURSOR:

```

    PUSH    HL                ;Zeile/Spalte auf Stack
    LD      E,ESC             ;ESC ausgeben
    CALL    DIROUT
    LD      E,'='              ;= ausgeben
    CALL    DIROUT
    POP     HL                ;Zeile/Spalte von Stack
    PUSH    HL                ;holen und wieder zurück
    LD      A,L               ;Zeile ausgeben
    CALL    CURSOR1           ;          mit CURSOR1
    POP     HL                ;Zeile/Spalte von Stack
    LD      A,H               ;Spalte ausgeben

```

CURSOR1:

```

    ADD     A,' '              ;20H addieren
    LD      E,A               ;und dann ausgeben
    CALL    DIROUT
    RET

```

```

;*****
;Unterprogramme DIROUT und DIRIN
; - gibt das Zeichen im E-Register mit Systemfunktion 6 aus
;   oder holt Zeichen mit Systemfunktion 6 (DIRCONF)
;*****

```

DIRIN:

```

    LD      E,OFFH

```

DIROUT:

```

    LD      C,DIRCONF
    CALL    SYSTEM
    RET

```

```

;*****
;Variable im RAM
;*****

```

```

    DS      81,0
FELD1:  DS      GROESSE,0      ;Spielfeld 1
        DS      2*81,0
FELD2:  DS      GROESSE,0      ;Spielfeld 2
        DS      81,0

    END     START

```

TITLE CONWAY's Life-Game schneller Programm 1

```
*****
;
;Definition von Systemadressen
*****
```

```
WSTART EQU 00000H ;Adresse für Warmstart
```

```
SYSTEM EQU 00005H ;Adresse für Aufruf von
;Systemfunktionen
```

```
*****
;
;Definition von Systemfunktionen
*****
```

```
WSTARTF EQU 0 ;Warmstart
```

```
DIRCONF EQU 6 ;direkte Konsole Ein-/Ausgabe
```

```
TXTBEGF EQU 247 ;Startadresse des Text-
;speichers abfragen
```

```
*****
;
;Definition von ASCII-Konstanten
*****
```

```
CONTROL EQU 1FH ;Erzeugung von Control-Codes
;mit ... AND CONTROL
```

```
CR EQU 0DH ;Cursor an Zeilenanfang
;(Carriage Return)
```

```
LF EQU 0AH ;Cursor in neue Zeile
;(Line Feed)
```

```
EOF EQU 1AH ;Markierung für Textende
;(End Of File)
```

```
ESC EQU 1BH ;Escape
;(für Cursor-Positionierung)
```

```
*****
;
;sonstige Konstanten
*****
```

```
ZEILEN EQU 24 ;Anzahl Zeilen
```

```
SPALTEN EQU 80 ;Anzahl Spalten
```

```
GROESSE EQU ZEILEN*SPALTEN ;Spielfeldgröße
```

```

;*****
;Hauptprogramm LIFE-GAME
;*****

```

```

      ORG      100H          ;Das Programm beginnt bei
                           ;0100H

```

```
START:
```

```

      CALL     HOLE_FELD      ;Feld aus Textspeicher holen
      CALL     SCHREIBE_FELD  ;und auf Schirm schreiben

```

```
LIFE:
```

```

      CALL     DIRIN          ;Tastatureingabe?

      OR       A              ;kein Zeichen (also A=0)?
      JR       Z,LIFE         ;ja: warten
                           ;nein, Zeichen da:
      CP       'C' AND CONTROL ; Control-C ?
      JP       Z,WSTART       ; ja: Warmstart

      CP       CR              ; Carriage Return?
      CALL     Z,GENERATION

      JR       LIFE           ;Schleife

```

```

;*****
;Unterprogramm GENERATION
; - Rechnet eine neue Generation aus
;*****

```

```

VON_FELD:      DW      FELD1
NACH_FELD:     DW      FELD2

```

```
GENERATION:
```

```

      LD       HL,0           ;Cursor nach links oben
      CALL     CURSOR
      LD       IX,(VON_FELD)  ;IX := Zeiger Ausgangsfeld
      LD       IY,(NACH_FELD) ;IY := Zeiger Ergebnisfeld
      LD       (VON_FELD),IY  ;Ausgangsfeld und Ergebnis-
      LD       (NACH_FELD),IX ;feld für nächste Generation
                           ;tauschen
      LD       BC,GROESSE-1   ;BC := Anzahl der Feldelemente
                           ;die zu berechnen sind

      EXX
      LD       HL,0           ;Spalte/Zeile auf Schirm und
      LD       DE,0           ;Zähler für ausgelassene
                           ;Zeichen
      EXX                     ;im 2. Registersatz

```


GENERATION_1:

```

CALL  PRUEFE_ELEMENT ;prüfe, ob Feldelement in
                        ;nächster Gen. besetzt ist
CALL  SETZE_ELEMENT  ;setze Feldelement in Ergebnis-
                        ;feld entsprechend
CALL  SCHREIBE_ELEMENT;schreibe * oder Leerzeichen
                        ;auf Schirm
LD     (IX),0         ;lösche altes Ausgangsfeld
INC    IX             ;erhöhe Zeiger Ausgangsfeld
INC    IY             ;erhöhe Zeiger Ergebnisfeld
DEC    BC             ;dekrementiere Zähler
LD     A,B            ;und prüfe, ob Null?
OR     C
JR     NZ,GENERATION_1 ; nein: weiter
RET                                ; ja: fertig

```

```

;*****
;Unterprogramm PRUEFE_ELEMENT
; - Prüft, ein einzelnes Feldelement in der nächsten
;   Generation belegt ist
;*****

```

PRUEFE_ELEMENT:

```

LD     E,(IX)         ;Feldelement laden
LD     D,0
LD     HL,REGELN      ;Adresse Regeltabelle nach HL
ADD    HL,DE
LD     A,(HL)         ;Tabellenwert laden
RET

```

REGELN:

```

DB     0,0,0,1,0,0,0,0 ;Spielregeln, erste Hälfte
DB     0,0,0,0,0,0,0,0 ;(16 Byte) für bisher
                        ;leeres Feldelement

DB     0,0,1,1,0,0,0,0 ;zweite für bisher belegtes
DB     0,0,0,0,0,0,0,0 ;Feldelement

```

```

;*****
;Unterprogramm SETZE_ELEMENT
; - Setzt ein einzelnen Feldelement im neuen Feld
;*****

```

SETZE_ELEMENT:

```

OR     A
RET    Z              ;A=0: schon fertig!
LD     A,10H
ADD    A,(IY)         ;belegtes Feld markieren
LD     (IY),A
INC    (IY-81)
INC    (IY-80)
INC    (IY-79)        ;und Nachbarmfelder erhöhen
INC    (IY-1)

```

```

INC      (IY+1)
INC      (IY+79)
INC      (IY+80)
INC      (IY+81)
RET

```

```

;*****
;Unterprogramm SCHREIBE_ELEMENT
; - Schreibt ein einzelnes Feldelement auf Schirm
;*****

```

```

;die Befehle, die den zweiten Registersatz benutzen, sind
;zur besseren Übersicht eingerückt!

```

SCHREIBE_ELEMENT:

```

XOR      (IX)          ;Vergleich mit
AND      10H           ;   Ausgangsfeld
CALL     Z,NO_CHANGE   ;gleich: nichts ändern
CALL     NZ,CHANGE     ;ungleich: neu schreiben

```

```

EXX                      ;neue Spalte/Zeile
INC      H              ;in HL' berechnen
LD       A,H
CP       SPALTEN        ;Zeile zu Ende?
JR       NZ,SCHREIBE_RET ; nein: Zeile bleibt
LD       H,0            ; ja:  Spalte=0 und
INC      L              ;   Zeile erhöhen

```

SCHREIBE_RET:

```

EXX                      ;zurück in 1. Registersatz
RET

```

NO_CHANGE:

```

EXX
INC      DE              ;Zähler in DE' erhöhen
EXX
RET

```

CHANGE:

```

EXX
LD       A,D             ;Zähler für nicht geschr.
OR       A               ;Elemente, D -> Z-Flag
LD       A,E             ;   E -> A
LD       DE,0            ;und Zähler=0
EXX

```

```

JR       NZ,NEU_CURSOR   ;D>0: Cursor neu setzen
DEC      A               ;D=0: E=1?
JR       NZ,NICHT_EINS
LD       E,9             ;ja: mit 09H Cursor 1 rechts
CALL     NO_SPACE
JR       SCHREIBE

```

```

NICHT_EINS:
    INC     A                ;E=0?

NEU_CURSOR:
    EXX
    CALL    NZ,CURSOR        ;nein: Cursor setzen
    EXX

SCHREIBE:
    LD      A,(IY)
    AND     10H
    LD      E,'*'
    JR      NZ,NO_SPACE
    LD      E,' '

NO_SPACE:
    PUSH    BC
    CALL    DIROUT
    POP     BC
    RET

```

```

;*****
;Unterprogramm HOLE_FELD
; - Holt vom Textspeicher eine Aufstellung in das interne
;   Spielfeld
;*****

```

```

HOLE_FELD:
    LD      C,TEXTBEGF        ;Anfang des Textspeichers
    CALL    SYSTEM            ;ins HL-Registerpaar
    LD      IY,(VON_FELD)     ; .. Spielfeld in IY
    LD      B,SPALTEN         ; und Feldgroesse in BC
    LD      C,ZEILEN

```

```

HOLE_WEITER:
    LD      A,(HL)            ;ein Zeichen holen
    INC     HL                ;Textzeiger weiter
    CP      LF                ;Zeichen = Line Feed?
    JR      Z,HOLE_WEITER     ; ja: ignorieren
    CP      EOF               ;Zeichen = Textende?
    JR      Z,LOESCHE_REST    ; ja: Feld löschen
    CP      CR                ;Zeichen = neue Zeile?
    JR      Z,NEUE_ZEILE      ; ja: in neue Zeile
    CP      ' '               ;Zeichen = Zwischenraum
    CALL    NZ,SETZE_ELEMENT; nein: Feldelement setzen

    LD      A,B                ;Zeile schon voll,
    OR      A                 ;d.h. Spaltenzähler auf Null?
    JR      Z,HOLE_WEITER     ; ja: weiter (bis neue Zeile)
    DEC     B                 ; nein: Spaltenzähler zurück
    INC     IY                ; Feldzeiger weiter
    JR      HOLE_WEITER       ; und weitermachen

```

```

LOESCHE_REST:
    DEC    HL                ;Textzeiger zurück auf EOF,
                                ;dann NEUE_ZEILE
                                ;dadurch so oft NEUE_ZEILE bis
                                ;Feld voll

NEUE_ZEILE:
    LD     A,B                ;Rest der Zeile wird gelöscht
                                ;Spaltenzähler in A
    LD     B,SPALTEN          ;Spaltenzähler setzen

LOESCHE_ZEILE:
    OR     A                  ;(alter) Spaltenzähler Null?
    JR     Z,ZEILE_VOLL       ; ja: Zeile schon voll
    INC    IY                 ; nein: Feldzeiger inkrementieren,
    DEC    A                  ; Zähler dekrementieren
    JR     LOESCHE_ZEILE      ; und weiter

ZEILE_VOLL:
    DEC    C                  ;Zeilenzähler zurück, Null?
    RET    Z                  ; ja: alle Zeilen voll, fertig
    JR     HOLE_WEITER        ; nein: weitermachen

```

```

;*****
;Unterprogramm SCHREIBE_FELD
; - Gibt das interne Spielfeld auf dem Bildschirm aus
;*****

```

```

SCHREIBE_FELD:
    LD     HL,0
    CALL   CURSOR             ;Cursor nach links oben
    LD     HL,(VON_FELD)      ;Adresse Spielfeld in HL
    LD     BC,GROESSE-1       ;Groesse Spielfeld in BC

SCHREIBE_EL:
    LD     E,'*'              ;schreibe * falls gesetzt
    LD     A,(HL)              ;hole Feldelement
    AND    10H                 ;Element gesetzt?
    JR     NZ,NICHT_LEER      ; ja: weiter, * lassen
    LD     E,' '               ; nein: Leerschritt statt *

NICHT_LEER:
    PUSH   BC                  ;Zeichen ausgeben
    PUSH   HL
    CALL   DIROUT
    POP    HL
    POP    BC
    INC    HL                  ;Feldzeiger erhöhen
    DEC    BC                  ;Zähler dekrementieren
    LD     A,B
    OR     C                    ;und prüfen, ob Null?
    JR     NZ,SCHREIBE_EL     ; nein: weiter
    RET                        ; ja: fertig

```

```

;*****
;Unterprogramm CURSOR
; - setzt den Cursor an Zeile L, Spalte H
;*****

```

CURSOR:

```

    PUSH    HL                ;Zeile/Spalte auf Stack
    LD      E,ESC             ;ESC ausgeben
    CALL    DIROUT
    LD      E,'='             ;= ausgeben
    CALL    DIROUT
    POP     HL                ;Zeile/Spalte von Stack
    LD      A,L               ;Zeile ausgeben
    CALL    CURSOR1           ;      mit CURSOR1
    LD      A,H               ;Spalte ausgeben

```

CURSOR1:

```

    PUSH    HL                ;Z/S zurück auf Stack
    ADD     A,' '             ;20H addieren
    LD      E,A               ;und dann ausgeben
    CALL    DIROUT
    POP     HL                ;Zeile/Spalte von Stack
    RET                      ;und zurück

```

```

;*****
;Unterprogramme DIROUT und DIRIN
; - gibt das Zeichen im E-Register mit Systemfunktion 6 aus
;   oder holt Zeichen mit Systemfunktion 6 (DIRCONF)
;*****

```

DIRIN:

```

    LD      E,OFFH

```

DIROUT:

```

    LD      C,DIRCONF
    CALL    SYSTEM
    RET

```

```

;*****
;Variable im RAM
;*****

```

```

    DS      81,0
FELD1: DS    GROESSE,0        ;Spielfeld 1
    DS      2*81,0
FELD2: DS    GROESSE,0        ;Spielfeld 2
    DS      81,0

    END     START

```

>asm

Z80-Assembler

pass 1:

+++++

pass 2:

+++++

no fatal error(s)

SUM=815D

CRC=9705

F

34

TITLE Fakultäetenberechnung - Programm 2

```
;*****
;Definition von Systemadressen
;*****
```

```
SYSTEM EQU      00005H      ;Adresse für Aufruf von
                           ;Systemfunktionen
```

```
;*****
;Definition von Systemfunktionen
;*****
```

```
CONAUSF EQU      2          ;Konsol-Ausgabe
```

```
STRAUSF EQU      9          ;String-Ausgabe
```

```
STREINF EQU      10         ;String-Eingabe
```

```
;*****
;Definition von ASCII-Konstanten
;*****
```

```
CR      EQU      0DH        ;Cursor an Zeilenanfang
                           ;(Carriage Return)
```

```
LF      EQU      0AH        ;Cursor in neue Zeile
                           ;(Line Feed)
```

```
;*****
;Definition von TRUE/FALSE (WAHR/FALSCH)
;*****
```

```
FALSE   EQU      0
TRUE    EQU      NOT FALSE
```

```

;*****
;Sonstige Konstanten
;*****

```

```

MAX_LAENGE      EQU      10      ;Länge des Eingabepuffers

MAX_FAKT        EQU      57      ;größte mögliche Eingabe
                                ;bei
DEZ_STELL       EQU      78      ;maximal 78 Dezimalstellen
                                ;(57! = 4*10 hoch 76) bzw.
BIN_STELL       EQU      256     ;256-Bit Binärzahl
                                ;(2 hoch 256 = 1*10 hoch 77)
BIN_BYTE        EQU      BIN_STELL/8
                                ;Anzahl Bytes d. Binärzahlen
CHECK           EQU      TRUE    ;prüfen, ob Speicher für
                                ;Datenbereich frei

```

```

;*****
;Datenbereich
;*****

```

```

DSEG            ;Datenbereich
ORG             ;Beginn bei 1000H
1000H

```

```

PUFFER:         ;Puffer für Eingabezeile
DB              ;Wert der maximalen Länge
1               ;Platz f. tatsächliche Länge
DS              ;Platz für Eingabe
MAX_LAENGE

```

```

DEZ_REG:        ;
DS              ;Speicher für Dezimalzahl
DEZ_STELL/2

```

```

BIN_REG1:       ;
DS              ;Speicher für Binärzahlen
BIN_BYTE

```

```

BIN_REG2:       ;
DS              ;
BIN_BYTE

```

```

IF              ;erzeugt beim assemblieren
CHECK           ;I-Fehler falls in ge-
0               ;schütztem Bereich
ENDIF

```

```

;*****
;Hauptprogramm FAKULTAET
;*****

```

```

CSEG            ;Programmbereich
ORG             ;Das Programm beginnt bei
100H             ;0100H

```

```

START:
    CALL    EINGABE            ;Eingabezeile von der
                                ;Tastatur holen
    CALL    HOLE_ZAHL          ;Eingabezeile auswerten und
    JR      C,START            ;Neueingabe bei Fehler

    CALL    FAKT               ;Fakultät berechnen

    CALL    WANDLE_DEZ         ;in Dezimalzahl wandeln

    CALL    AUSGABE            ;ausgeben

    JR      START              ;und zurück zur Eingabe

;*****
;Unterprogramm EINGABE
;    holt eine Eingabezeile von der Tastatur in den Puffer
;*****

EINGABE_TEXT:
    DB      CR,LF,'? $'

EINGABE:
    LD      DE,EINGABE_TEXT    ;? in neuer Zeile ausgeben
    LD      C,STRAUSF
    CALL    SYSTEM

    LD      DE,PUFFER          ;und Eingabezeile einlesen
    LD      C,STREINF
    CALL    SYSTEM
    RET

;*****
;Unterprogramm HOLE_ZAHL
;    wertet die Eingabe aus und erzeugt Binärzahl
;    prüft, ob Eingabe zu groß
;*****

HOLE_ZAHL:
    LD      HL,PUFFER+1        ;HL auf Länge d. Eingabe
    LD      B,(HL)             ;dieselbe nach B
    INC     HL                 ;HL auf Eingabe
    EX      DE,HL              ;DE auf Eingabe und
    LD      HL,0               ;HL := 0

HOL_SCHLEIFE:
    LD      A,(DE)             ; Zeichen von Eingabe holen
    INC     DE                 ; und Zeiger weiter
    SUB     '0'                ; ASCII -> binär wandeln
    RET     C                  ; Fehler falls kleiner 0
    CP      10                 ; oder größer 9
    CCF
    RET     C                  ; (Rücksprung mit Carry bei
                                ; Fehler)

```



```

        PUSH    BC                ; Schleifenzähler und
        PUSH    DE                ; Zeiger retten,
        EX      DE,HL             ; DE := HL
        LD      B,10
        LD      HL,0              ; HL := 0
HOL_SCHLEIFE1:                ; wiederhole 10 mal:
        ADD     HL,DE              ; HL := HL+DE
        DJNZ    HOL_SCHLEIFE1     ; (d.h. HL := HL*10)
        LD      E,A
        LD      D,0
        ADD     HL,DE              ; HL := HL+Ziffer
        POP     DE                ; Zeiger und
        POP     BC                ; Schleifenzähler holen

        DJNZ    HOL_SCHLEIFE      ; Zähler (B) dekrementieren
                                   ; bis Zähler=0
        EX      DE,HL             ; Ergebnis nach DE
        LD      HL,MAX_FAKT
        AND     A
        SBC     HL,DE              ; MAX_FACT-Eingabe:
                                   ; CY=1 falls Eingabe>MAX_FACT

        RET

```

```

;*****
;Unterprogramm FAKT
;    errechnet Fakultät der Eingabe
;    Argument wird in DE übergeben
;*****

```

```

FAKT:
    RET

```

```

;*****
;Unterprogramm WANDLE_DEZ
;*****

```

```

WANDLE_DEZ:
    RET

```

```

;*****
;Unterprogramm AUSGABE
;*****

```

```

PRWORDF EQU    241

```

```

AUSGABE:
    LD      C,PRWORDF
    CALL    SYSTEM

    RET

    END     START

```

>asm

Z80-Assembler

pass 1:

++++++

pass 2:

++++++

no fatal error(s)

SUM=1FE8

CRC=793F

TITLE Fakultaetenberechnung - Programm 3

```

;*****
;Definition von Systemadressen
;*****

```

```

SYSTEM EQU      00005H          ;Adresse für Aufruf von
                                ;Systemfunktionen

```

```

;*****
;Definition von Systemfunktionen
;*****

```

```

CONAUSF EQU     2                ;Konsol-Ausgabe

```

```

STRAUSF EQU     9                ;String-Ausgabe

```

```

STREINF EQU     10               ;String-Eingabe

```

```

;*****
;Definition von ASCII-Konstanten
;*****

```

```

CR      EQU     0DH              ;Cursor an Zeilenanfang
                                ;(Carriage Return)

```

```

LF      EQU     0AH              ;Cursor in neue Zeile
                                ;(Line Feed)

```

```

;*****
;Definition von TRUE/FALSE (WAHR/FALSCH)
;*****

```

```

FALSE   EQU     0
TRUE    EQU     NOT FALSE

```

```

;*****
;
;Sonstige Konstanten
;*****

MAX_LAENGE      EQU      10      ;Länge des Eingabepuffers

MAX_FAKT        EQU      57      ;größte mögliche Eingabe
                                ;bei
DEZ_STELL       EQU      78      ;maximal 78 Dezimalstellen
                                ;(57! = 4*10 hoch 76) bzw.
BIN_STELL       EQU      256     ;256-Bit Binärzahl
                                ;(2 hoch 256 = 1*10 hoch 77)
BIN_BYTE        EQU      BIN_STELL/8
                                ;Anzahl Bytes d. Binärzahlen
CHECK           EQU      TRUE    ;prüfen, ob Speicher für
                                ;Datenbereich frei

;*****
;
;Datenbereich
;*****

        DSEG                      ;Datenbereich
        ORG      1000H            ;Beginn bei 1000H

PUFFER:
        DB      MAX_LAENGE        ;Wert der maximalen Länge
        DS      1                 ;Platz f. tatsächliche Länge
        DS      MAX_LAENGE        ;Platz für Eingabe

DEZ_REG:
        DS      DEZ_STELL/2       ;Speicher für Dezimalzahl
BIN_REG1:
        DS      BIN_BYTE          ;Speicher für Binärzahlen
BIN_REG2:
        DS      BIN_BYTE

        IF      CHECK              ;erzeugt beim assemblieren
        DB      0                  ;I-Fehler falls in ge-
        ENDIF                      ;schütztem Bereich

;*****
;
;Hauptprogramm FAKULTAET
;*****

        CSEG                      ;Programmbereich
        ORG      100H              ;Das Programm beginnt bei
                                ;0100H

START:
        CALL    EINGABE            ;Eingabezeile von der
                                ;Tastatur holen
        CALL    HOLE_ZAHL          ;Eingabezeile auswerten und
        JR      C,START            ;Neueingabe bei Fehler

```

```

        CALL    FAKT                ;Fakultät berechnen
        CALL    WANDLE_DEZ         ;in Dezimalzahl wandeln
        CALL    AUSGABE            ;ausgeben
R        START                ;und zurück zur Eingabe

;*****
;Unterprogramm EINGABE
;    holt eine Eingabezeile von der Tastatur in den Puffer
;*****

EINGABE_TEXT:
        DB      CR,LF,'? $'

EINGABE:
        LD      DE,EINGABE_TEXT ;? in neuer Zeile ausgeben
        LD      C,STRAUSF
        CALL    SYSTEM

        LD      DE,PUFFER        ;und Eingabezeile einlesen
        LD      C,STREINF
LL       SYSTEM
        RET

;*****
;Unterprogramm HOLE_ZAHL
;    wertet die Eingabe aus und erzeugt Binärzahl
;    prüft, ob Eingabe zu groß
;*****

HOLE_ZAHL:
        LD      HL,PUFFER+1      ;HL auf Länge d. Eingabe
        LD      B,(HL)           ;dieselbe nach B
        INC     HL               ;HL auf Eingabe
        EX      DE,HL            ;DE auf Eingabe und
        LD      HL,0             ;HL := 0
HOL_SCHLEIFE:
        LD      A,(DE)           ; Zeichen von Eingabe holen
        INC     DE               ; und Zeiger weiter

        SUB     '0'              ; ASCII -> binär wandeln
        RET     C                ; Fehler falls kleiner 0
        CP      10               ; oder größer 9
        CCF
        RET     C                ; (Rücksprung mit Carry bei
                                ; Fehler)
        PUSH    BC               ; Schleifenzähler und
        PUSH    DE               ; Zeiger retten,
        EX      DE,HL            ; DE := HL
        LD      B,10
        LD      HL,0             ; HL := 0
HOL_SCHLEIFE1:
        ADD     HL,DE            ; wiederhole 10 mal:
        DJNZ    HOL_SCHLEIFE1   ; HL := HL+DE
                                ; (d.h. HL := HL*10)

```

```

LD      E,A
LD      D,0
ADD     HL,DE      ; HL := HL+Ziffer
POP     DE         ; Zeiger und
POP     BC         ; Schleifenzähler holen
DJNZ    HOL_SCHLEIFE ; Zähler (B) dekrementieren
                        ;bis Zähler=0
EX      DE,HL      ;Ergebnis nach DE
LD      HL,MAX_FAKT
AND     A
SBC     HL,DE      ;MAX_FACT-Eingabe:
                        ;CY=1 falls Eingabe>MAX_FACT
RET

```

```

;*****
;Unterprogramm FAKT
;   errechnet Fakultät der Eingabe
;   Argument wird in DE übergeben
;*****

```

FAKT:

```

LD      HL,BIN_REG1 ;BIN_REG1 := 1
CALL    CLEAR       ;BIN_REG1 wird gelöscht
DEC     HL           ;und das letzte Byte auf 1
INC     (HL)         ;inkrementiert
LD      B,E          ;Argument ins B-Register
FAKT_SCHLEIFE:      ;wiederhole:
CALL    MULT         ; BIN_REG2 := BIN_REG1*B
CALL    MOV12        ; BIN_REG1 := BIN_REG2
DJNZ    FAKT_SCHLEIFE ; Zähler (B) dekrementieren
RET      ;bis Zähler=0

```

```

;*****
;Unterprogramm MULT
;   multipliziert BIN_REG1 mit B, Ergebnis in BIN_REG2
;*****

```

MULT:

```

LD      HL,BIN_REG2
CALL    CLEAR       ;BIN_REG2 := 0

LD      C,B         ;Faktor ins C-Register

MULT_SCHLEIFE:      ;wiederhole:
SRL     C           ; C :=C/2, CY := Rest
CALL    C,ADDIERE   ; falls Rest (Carry)=1:
                        ;   BIN_REG2 := BIN_REG2
                        ;   +BIN_REG1
CALL    SCHIEBE     ; BIN_REG1 := BIN_REG1*2
LD      A,C
OR      A           ;bis C=0
JR      NZ,MULT_SCHLEIFE
RET

```

```

;*****
;Unterprogramm ADDIERE
;   addiert BIN_REG1 zu BIN_REG2, Ergebnis in BIN_REG2
;*****

```

ADDIERE:

```

        PUSH    BC                ;Vorbereitung:
        LD      HL,BIN_REG2 + BIN_BYTE ;Pointer auf
        LD      DE,BIN_REG1 + BIN_BYTE ;niedrigstes Byte+1
        LD      B,BIN_BYTE        ;Zähler laden
        AND     A                  ;und Carry löschen
ADDIER_SCHLEIFE:
        DEC     HL                ; wiederhole:
        DEC     DE                ; Zeiger dekrementieren
        LD      A,(DE)            ; Byte von BIN_REG1 laden
        ADC     A,(HL)            ; zu BIN_REG2 addieren
        LD      (HL),A            ; und in BIN_REG2 speichern
        DJNZ    ADDIER_SCHLEIFE ; Zähler (B) dekrementieren
        POP     BC                ;bis Zähler=0
        RET

```

```

;*****
;Unterprogramm SCHIEBE
;   schiebt BIN_REG1 eine Binärstelle nach links
;*****

```

SCHIEBE:

```

        PUSH    BC                ;Vorbereitung:
        LD      HL,BIN_REG1 + BIN_BYTE ;Pointer und
        LD      B,BIN_BYTE        ;Zähler und
        AND     A                  ;Carry wie b. ADDIERE
SCHIEBE_SCHLEIFE:
        DEC     HL                ; wiederhole:
        RL      (HL)              ; Zeiger dekrement.,
        DJNZ    SCHIEBE_SCHLEIFE ; byteweise schieben
        POP     BC                ; Zähler (B) dekrem.
        RET                      ;bis Zähler=0

```

```

;*****
;Unterprogramm CLEAR
;   löscht eine Binärzahl
;*****

```

CLEAR:

```

        PUSH    BC
        LD      B,BIN_BYTE        ;Zähler auf Anzahl Byte
CLEAR1:
        LD      (HL),0            ;wiederhole:
        INC     HL                ; ein Byte löschen,
        DJNZ    CLEAR1            ; Zeiger inkrementieren,
        POP     BC                ; Zähler (B) dekrementieren
        RET                      ;bis Zähler=0

```

```

;*****
;Unterprogramm MOV12
;   kopiert BIN_REG2 nach BIN_REG1
;*****

```

MOV12:

```

    PUSH    BC           ;Faktor (für MULT!) retten
    LD      DE,BIN_REG1  ;Ziel = BIN_REG1
    LD      HL,BIN_REG2  ;Quelle = BIN_REG2
    LD      BC,BIN_BYTE  ;Anzahl Byte = BIN_BYTE
    LDIR                     ;und schieben
    POP     BC
    RET

```

```

;*****
;Unterprogramm WANDLE_DEZ
;   wandelt BIN_REG1 in eine Dezimalzahl (nach DEZ_REG)
;*****

```

WANDLE_DEZ:

```

    LD      HL,DEZ_REG    ;alte Dezimalzahl löschen:
    LD      B,DEZ_STELL/2

```

CLEAR_DEZ:

```

;wiederhole:
    LD      (HL),0        ; ein Byte löschen
    INC     HL             ; Zeiger inkrementieren
    DJNZ    CLEAR_DEZ     ; Zähler (B) dekrementieren
;bis Zähler=0

```

```

    LD      DE,BIN_STELL  ;alle Binärbits wandeln:

```

WANDL_SCHLEIFE:

```

;wiederhole:
    CALL    SCHIEBE       ; BIN_REG1 := BIN_REG1*2
; DEZ_REG := DEZ_REG*2
    LD      B,DEZ_STELL/2 ; + Carry

```

WANDL_SCHLEIFE1:

```

; wiederhole:
    DEC     HL            ; Zeiger dekrementieren
    LD      A,(HL)        ; Byte laden
    ADC     A,(HL)        ; mit Carry addieren
    DAA                     ; dezimal korrigieren!
    LD      (HL),A        ; speichern
    DJNZ    WANDL_SCHLEIFE1 ; Zähler (B) dekrement.
; bis Zähler=0

```

```

    DEC     DE            ; Zähler (DE) dekremen.

```

```

    LD      A,D

```

```

    OR      E             ;bis Zähler=0

```

```

    JR      NZ,WANDL_SCHLEIFE

```

```

    RET

```

```

;*****
;Unterprogramm AUSGABE
;*****

```

```

FLAG    EQU    0
AUSGABE:
    RES    FLAG,C        ;Flag löschen,
    LD     HL,DEZ_REG    ;Zeiger und
    LD     B,DEZ_STELL/2 ;Zähler setzen
AUSGABE_SCHLEIFE:        ;wiederhole:
    RLD                     ; Dezimaldigit (Halbbyte)
    CALL   EINE_STELLE    ; holen und ausgeben
    RLD                     ; dto. für anderes Halb-
    CALL   EINE_STELLE    ; byte (Digit)
    INC    HL              ; Zeiger auf nächstes Byte
    DJNZ   AUSGABE_SCHLEIFE; Zähler (B) dekrementieren
    RET                      ;bis Zähler=0

EINE_STELLE:              ;Digit ausgeben:
    AND    0FH            ;rechtes Halbbyte maskieren
    JR     NZ,SCHREIBE    ;Digit=0? nein: ausgeben
    BIT    FLAG,C         ; ja: Flag gesetzt?
    JR     Z,              ; nein: zurück (führende
                          ; Nullen unterdrücken)
    JR     NZ,              ; ja: ausgeben (normale
                          ; Null)
SCHREIBE:
    SET    0,C            ;Flag setzen
    PUSH   BC              ;Zähler und
    PUSH   HL              ;Zeiger retten,
    ADD    A,'0'           ;Digit -> ASCII
    LD     E,A
    LD     C,CONAUSF       ;ausgeben
    CALL   SYSTEM         ;mit Systemfunktion
    POP    HL              ;Zeiger und
    POP    BC              ;Zähler wiederherstellen
    RET

    END    START

```

>asm

Z80-Assembler

pass 1:

+++++

pass 2:

+++++

no fatal error(s)

SUM=5F8F

CRC=C2B9

TITLE Primzahlen - Programm 1

```

;*****
;Definition von Systemadressen
;*****

SYSTEM EQU      00005H      ;Adresse für Aufruf von
                             ;Systemfunktionen

;*****
;Definition von Systemfunktionen
;*****

KONAUSF EQU      2          ;Konsol-Ausgabe

DIRCONF EQU      6          ;Direkte Konsol-Ein/Ausgabe

STRAUSF EQU      9          ;Stringausgabefunktion

;*****
;Definition von ASCII-Konstanten
;*****

CR      EQU      0DH        ;Cursor an Zeilenanfang
                             ;(Carriage Return)
LF      EQU      0AH        ;Cursor in neue Zeile
                             ;(Line Feed)
CTRL_C  EQU      'C' AND 1FH ;für Abbruchbedingung

;*****
;Definition von TRUE/FALSE (WAHR/FALSCH)
;*****

FALSE   EQU      0
TRUE    EQU      NOT FALSE

;*****
;Sonstige Konstanten
;*****

DEZ_STELL EQU      10      ;maximal 10 Dezimalstellen
                             ;2 hoch 32 = 4*10 hoch 9
BIN_STELL EQU      32      ;32-Bit Binärzahl

BIN_BYTE EQU      BIN_STELL/8
                             ;Anzahl Bytes d. Binärzahlen

```

```

;*****
;
;Hauptprogramm PRIMZAHLEN
;*****

```

START:

```

LD      HL,PRIM      ;Setze PRIM = 1
CALL    SETZE1

```

SCHLEIFE:

```

;wiederhole:
; wiederhole:

```

```

LD      HL,PRIM
CALL    PLUS2        ; erhöhe PRIM um 2

CALL    PRUEFE        ; Prüfe ob PRIM Primzahl

JR      NC,SCHLEIFE   ; bis Primzahl gefunden

CALL    WANDLE_DEZ    ; Wandle PRIM in Dezimalzahl

CALL    AUSGABE       ; Gib PRIM auf Konsole aus

LD      DE,NEUE_ZEILE ; Gib CR/LF auf Konsole aus
LD      C,STRAUSF
CALL    SYSTEM

LD      C,DIRCONF
LD      E,OFFH
CALL    SYSTEM
CP      CTRL_C
JR      NZ,SCHLEIFE
JP      0              ;bis Abbruch mit CTRL-C

```

NEUE_ZEILE:

```

DB      CR,LF,'$'

```

```

;*****
;
;Unterprogramm PRUEFE
;
; Prüft, ob PRIM eine Primzahl ist
;*****

```

PRUEFE:

```

LD      HL,TEILER    ;Setze TEILER = 1
CALL    SETZE1

```

PRUEFE_SCHL:

```

;wiederhole:

```

```

LD      HL,TEILER
CALL    PLUS2        ; erhöhe TEILER um 2
CALL    DIV           ; Dividiere PRIM/TEILER
                        ; Ergebnis: QUOTIENT, REST
LD      HL,REST       ;bis REST gleich Null
CALL    NULL          ; --> keine Primzahl
RET     Z             ; --> Carry=0

```

```

LD      IX,BIN_REG      ;oder TEILER >= QUOTIENT
LD      DE,TEILER       ; TEILER - QUOTIENT
LD      HL,QUOTIENT     ; Carry=Vorzeichen Ergebnis
CALL    SUB             ; Carry=1: TEILER<QUOTIENT
                        ; Carry=0: TEILER>=QUOTIENT
JR      C,PRUEFE_SCHL   ; --> Primzahl
SCF                                           ; --> Carry:=1
RET

```

```

;*****
;Unterprogramm DIV
;   dividiert PRIM durch TEILER
;   Ergebnis in QUOTIENT, Rest in REST
;*****

```

DIV:

```

LD      B,1             ;Setze ZAEHL = 1
DIV_SCHL1:              ;wiederhole:
INC      B              ; erhöhe ZAEHL um 1
LD      HL,TEILER
CALL    SCHIEBE         ; schiebe TEILER links
LD      IX,BIN_REG
LD      DE,PRIM
LD      HL,TEILER
CALL    SUB
JR      NC,DIV_SCHL1    ;bis TEILER > PRIM

LD      HL,PRIM
LD      DE,REST
CALL    MOV             ;Kopiere PRIM nach REST
LD      HL,QUOTIENT
CALL    SETZE1
DEC      (HL)           ;Setze QUOTIENT gleich Null

```

```

DIV_SCHL2:              ;wiederhole:
LD      IX,BIN_REG      ; BIN_REG :=
LD      DE,REST         ; REST
LD      HL,TEILER       ; -TEILER
CALL    SUB             ; Ergebnis positiv?
LD      HL,BIN_REG
LD      DE,REST
CALL    NC,MOV          ; ja: BIN_REG nach REST
CCF                                           ; komplementiere Carry-Flag
LD      HL,QUOTIENT
CALL    SCHIEBE_CARRY   ; Carry-Flag in QUOTIENT
LD      HL,TEILER
CALL    SCHIEBE_RECHTS  ; schiebe TEILER rechts
DJNZ    DIV_SCHL2       ; dekrementiere ZAEHL (B)
                        ;bis ZAEHL=0

LD      HL,TEILER
CALL    SCHIEBE_CARRY   ;Teiler wiederherstellen
RET

```

```

;*****
;Unterprogramm SUB
;   addiert Register (HL) von Register (DE), Ergebnis
;   nach Register (IX)
;*****

```

SUB:

```

        PUSH    BC
        LD      BC,BIN_BYTE
        EX      DE,HL
        ADD     HL,BC           ;alle Zeiger ans Registerende
        EX      DE,HL           ;(+BIN_BYTE)
        ADD     HL,BC
        ADD     IX,BC           ;Zähler setzen und
        LD      B,C
        AND     A               ;Carry löschen
SUB_SCHLEIFE:
        DEC     DE               ;wiederhole:
        DEC     HL
        DEC     IX               ; Zeiger dekrement.,
        LD      A,(DE)           ; Byte laden,
        SBC     A,(HL)           ; subtrahieren und
        LD      (IX),A           ; Ergebnis speichern
        DJNZ    SUB_SCHLEIFE     ; Zähler (B) dekrem.
        POP     BC               ;bis Zähler=0
        RET

```

```

;*****
;Unterprogramm SETZE1
;   setzt REGISTER (HL) auf 1
;*****

```

SETZE1:

```

        PUSH    BC
        LD      B,BIN_BYTE      ;Zähler auf Anzahl Byte
SETZE1_SCHLEIFE:
        LD      (HL),0           ; ein Byte löschen,
        INC     HL               ; Zeiger inkrementieren,
        DJNZ    SETZE1_SCHLEIFE ; Zähler (B) dekrementieren
                                   ;bis Zähler=0
        DEC     HL               ;Zeiger inkrementieren
        INC     (HL)             ;und Byte auf 1 erhöhen
        POP     BC
        RET

```

```

;*****
;Unterprogramm PLUS2
;   addiert 2 zu Register (HL)
;*****

```

PLUS2:

```

        PUSH    BC
        LD      BC,BIN_BYTE      ;Zeiger ans Registerende

```

```

        ADD    HL,BC          ;(+BIN_BYTE) (kein Carry!)
        LD     B,C           ;Zähler setzen
        LD     A,2           ;Summand = 2
PLUS2_SCHLEIFE:                ;wiederhole:
        DEC    HL            ; Zeiger dekrementieren
        ADC    A,(HL)        ; ein Byte addieren
        LD     (HL),A        ; und wieder speichern
        LD     A,0           ; Akku=0 für folgende Bytes
        DJNZ   PLUS2_SCHLEIFE ; Zähler (B) dekrementieren
        POP    BC            ;bis Zähler=0
        RET

```

```

;*****
;Unterprogramme SCHIEBE, SCHIEBE_CARRY
;    schiebt Register (HL) eine Binärstelle nach links
;*****

```

```

SCHIEBE_CARRY:
    PUSH    BC
    PUSH    AF              ;Carry retten
    LD      BC,BIN_BYTE     ;Zeiger ans Registerende
    ADD     HL,BC           ;(+BIN_BYTE)
    POP     AF              ;Carry restaurieren
    JR      SCHIEBE1        ;und schieben

```

```

SCHIEBE:
    PUSH    BC
    LD      BC,BIN_BYTE     ;Zeiger ans Registerende
    ADD     HL,BC           ;(+BIN_BYTE) (kein Carry!)

```

```

SCHIEBE1:
    LD      B,C             ;Zähler (B) setzen
SCHIEBE_SCHL:                ;wiederhole:
    DEC     HL              ; Zeiger dekrement.,
    RL      (HL)            ; byteweise schieben
    DJNZ    SCHIEBE_SCHL    ; Zähler (B) dekrem.
    POP     BC              ;bis Zähler=0
    RET

```

```

;*****
;Unterprogramm SCHIEBE_RECHTS
;    schiebt Register (HL) eine Binärstelle nach rechts
;*****

```

```

SCHIEBE_RECHTS:
    PUSH    BC
    LD      B,BIN_BYTE     ;Zähler setzen
    AND     A              ;Carry löschen
SCHIEBE_R:                ;wiederhole:
    RR      (HL)           ; byteweise schieben
    INC     HL              ; Zeiger inkrement.,
    DJNZ    SCHIEBE_R      ; Zähler (B) dekrem.
    POP     BC              ;bis Zähler=0
    RET

```

```

;*****
;Unterprogramm MOV
;      kopiert Register (HL) nach Register (DE)
;*****

```

MOV:

```

      PUSH    BC
      LD      BC,BIN_BYTE      ;Anzahl Byte = BIN_BYTE
      LDIR                      ;und schieben
      POP     BC
      RET

```

```

;*****
;Unterprogramm NULL
;      Prüft, ob Register (HL) gleich Null
;*****

```

NULL:

```

      PUSH    BC
      LD      B,BIN_BYTE      ;Anzahl Byte = BIN_BYTE
      XOR     A                ;A=0
NULL_SCHLEIFE:
      OR      (HL)            ; ODER-Verkn mit Byte
      INC     HL               ; Zeiger inkrementieren
      DJNZ    NULL_SCHLEIFE   ; Zähler (B) dekrement.
                                ;bis Zähler=0
      AND     A                ;Bedingungsflags setzen
      POP     BC
      RET

```

```

;*****
;Unterprogramm WANDLE_DEZ
;      wandelt PRIM in eine Dezimalzahl (nach DEZ_REG)
;*****

```

WANDLE_DEZ:

```

      LD      HL,PRIM
      LD      DE,BIN_REG
      CALL    MOV              ;PRIM nach BIN_REG
      LD      HL,DEZ_REG       ;alte Dezimalzahl löschen:
      LD      B,DEZ_STELL/2
CLEAR_DEZ:
      LD      (HL),0           ;wiederhole:
                                ; ein Byte löschen
      INC     HL               ; Zeiger inkrementieren
      DJNZ    CLEAR_DEZ       ; Zähler (B) dekrementieren
                                ;bis Zähler=0
      LD      DE,BIN_STELL     ;alle Binärbits wandeln:
WANDL_SCHLEIFE:
      LD      HL,BIN_REG
      CALL    SCHIEBE          ; BIN_REG := BIN_REG*2
                                ; DEZ_REG := DEZ_REG*2
      LD      B,DEZ_STELL/2    ;
                                ;      + Carry

```

```

WANDL_SCHLEIFE1:      ; wiederhole:
    DEC    HL          ; Zeiger dekrementieren
    LD     A,(HL)       ; Byte laden
    ADC    A,(HL)       ; mit Carry addieren
    DAA          ; dezimal korrigieren!
    LD     (HL),A       ; speichern
    DJNZ   WANDL_SCHLEIFE1 ; Zähler (B) dekrement.
                    ; bis Zähler=0
    DEC    DE          ; Zähler (DE) dekrement.
    LD     A,D
    OR     E            ;bis Zähler=0
    JR     NZ,WANDL_SCHLEIFE
    RET

```

```

;*****
;Unterprogramm AUSGABE
;*****

```

```

FLAG    EQU    0

```

```

AUSGABE:
    RES    FLAG,C       ;Flag löschen,
    LD     HL,DEZ_REG    ;Zeiger und
    LD     B,DEZ_STELL/2 ;Zähler setzen

```

```

AUSGABE_SCHLEIFE:      ;wiederhole:
    RLD          ; Dezimaldigit (Halbbyte)
    CALL    EINE_STELLE ; holen und ausgeben
    RLD          ; dto für anderes Halb-
    CALL    EINE_STELLE ; byte (Digit)
    INC     HL       ; Zeiger auf nächstes Byte
    DJNZ    AUSGABE_SCHLEIFE; Zähler (B) dekrementieren
    RET          ;bis Zähler=0

```

```

EINE_STELLE:           ;Digit ausgeben:
    AND     OFH        ;rechtes Halbbyte maskieren
    JR     NZ,SCHREIBE  ;Digit=0? nein: ausgeben
    BIT     FLAG,C      ; ja: Flag gesetzt?
    RET     Z          ; nein: zurück (führende
                    ; Nullen unterdrücken)
                    ; ja: ausgeben (normale
                    ; Null)

```

```

SCHREIBE:
    SET     0,C         ;Flag setzen
    PUSH    BC          ;Zähler und
    PUSH    HL          ;Zeiger retten,
    ADD     A, 0'       ;Digit -> ASCII
    LD     E,A
    LD     C,KONAUSF    ;ausgeben
    CALL    SYSTEM      ;mit Systemfunktion
    POP     HL          ;Zeiger und
    POP     BC          ;Zähler wiederherstellen
    RET

```

```

;*****
;Datenbereich
;*****

DEZ_REG:
    DS      DEZ_STELL/2      ;Speicher für Dezimalzahl
BIN_REG:
    DS      BIN_BYTE
PRIM:
    DS      BIN_BYTE      ;Speicher für Binärzahlen
TEILER:
    DS      BIN_BYTE
QUOTIENT:
    DS      BIN_BYTE
REST:
    DS      BIN_BYTE

    END      START

```

```

Z80-Assembler
pass 1:
+++++
pass 2:
+++++

no fatal error(s)

```

```

SUM=817C
CRC=2533

```

TITLE Primzahlen - Programm 2

```

;*****
;Definition von Systemadressen
;*****

SYSTEM EQU    00005H      ;Adresse für Aufruf von
                           ;Systemfunktionen

;*****
;Definition von Systemfunktionen
;*****

KONAUSF EQU    2          ;Konsol-Ausgabe
DIRCONF EQU    6          ;Direkte Konsol-Ein/Ausgabe
STRAUSF EQU    9          ;Stringausgabefunktion

```



```

;*****
;Definition von ASCII-Konstanten
;*****

```

```

CR      EQU      ODH          ;Cursor an Zeilenanfang
                                ;(Carriage Return)
LF      EQU      OAH          ;Cursor in neue Zeile
                                ;(Line Feed)
CTRL_C  EQU      'C' AND 1FH  ;für Abbruchbedingung

```

```

;*****
;Definition von TRUE/FALSE (WAHR/FALSCH)
;*****

```

```

FALSE   EQU      0
TRUE    EQU      NOT FALSE

```

```

;*****
;Sonstige Konstanten
;*****

```

```

DEZ_STELL      EQU      10      ;maximal 10 Dezimalstellen
                                ;2 hoch 32 = 4*10 hoch 9
BIN_STELL      EQU      32      ;32-Bit Binärzahl
BIN_BYTE       EQU      BIN_STELL/8
                                ;Anzahl Bytes d. Binärzahlen

```

```

;*****
;Hauptprogramm PRIMZAHLEN
;*****

```

START:

```

        LD      HL,PRIM      ;Setze PRIM = 1
        CALL    SETZE1

```

```

SCHLEIFE0:      ;wiederhole:
        LD      B,7          ; Zähler für 7 Zahlen/Zeile

```

```

SCHLEIFE:      ; wiederhole:
                ; wiederhole:
        PUSH    BC          ; Zähler retten

        LD      HL,PRIM
        CALL    PLUS2      ; erhöhe PRIM um 2

        CALL    PRUEFE      ; Prüfe ob PRIM Primzahl

        POP     BC          ; Zähler restaurieren

```

```

JR      NC,SCHLEIFE      ;   bis Primzahl gefunden
PUSH    BC                ;   Zähler retten
CALL    WANDLE_DEZ       ;   Wandle PRIM in Dezimalzahl
CALL    AUSGABE          ;   Gib PRIM auf Konsole aus
POP      BC               ;   Zähler restaurieren
DJNZ     SCHLEIFE        ;   und dekrementieren
                        ;   bis Zähler (B) gleich Null
LD       DE,NEUE_ZEILE   ;   Gib CR/LF auf Konsole aus
LD       C,STRAUSF
CALL     SYSTEM

LD       C,DIRCONF
LD       E,OFFH
CALL     SYSTEM
CP       CTRL_C
JR       NZ,SCHLEIFE0
JP       0                ;bis Abbruch mit CTRL-C

```

NEUE_ZEILE:

```

DB      CR,LF,'$'

```

```

;*****
;Unterprogramm PRUEFE
;   Prüft, ob PRIM eine Primzahl ist
;*****

```

PRUEFE:

```

LD      HL,TEILER        ;Setze TEILER = 1
CALL    SETZE1
PRUEFE_SCHL:              ;wiederhole:
LD      HL,TEILER
CALL    PLUS2             ; erhöhe TEILER um 2
CALL    DIV               ; Dividiere PRIM/TEILER
                        ; Ergebnis: QUOTIENT, REST
LD      HL,REST           ;bis REST gleich Null
CALL    NULL              ; --> keine Primzahl
RET     Z                 ; --> Carry=0
LD      IX,BIN_REG        ;oder TEILER >= QUOTIENT
LD      DE,TEILER         ; TEILER - QUOTIENT
LD      HL,QUOTIENT       ; Carry=Vorzeichen Ergebnis
CALL    SUB               ; Carry=1: TEILER<QUOTIENT
                        ; Carry=0: TEILER>=QUOTIENT
JR      C,PRUEFE_SCHL    ; --> Primzahl
SCF
RET     1                 ; --> Carry:=1

```

```

;*****
;Unterprogramm DIV
;   dividiert PRIM durch TEILER
;   Ergebnis in QUOTIENT, Rest in REST
;*****

```

DIV:

```

    LD      B,1          ;Setze ZAEHL = 1

```

```

DIV_SCHL1:                ;wiederhole:
    INC     B            ; erhöhe ZAEHL um 1
    LD      HL,TEILER
    CALL    SCHIEBE      ; schiebe TEILER links
    LD      IX,BIN_REG
    LD      DE,PRIM
    LD      HL,TEILER
    CALL    SUB
    JR      NC,DIV_SCHL1 ;bis TEILER > PRIM

    LD      HL,PRIM
    LD      DE,REST
    CALL    MOV          ;Kopiere PRIM nach REST
    LD      HL,QUOTIENT
    CALL    SETZE1
    DEC     (HL)         ;Setze QUOTIENT gleich Null

```

```

DIV_SCHL2:                ;wiederhole:
    LD      IX,BIN_REG   ; BIN_REG :=
    LD      DE,REST      ; REST
    LD      HL,TEILER    ; -TEILER
    CALL    SUB          ; Ergebnis positiv?
    LD      HL,BIN_REG
    LD      DE,REST
    CALL    NC,MOV        ; ja: BIN_REG nach REST
    CCF                ; komplementiere Carry-Flag
    LD      HL,QUOTIENT
    CALL    SCHIEBE_CARRY ; Carry-Flag in QUOTIENT
    LD      HL,TEILER
    CALL    SCHIEBE_RECHTS ; schiebe TEILER rechts
    DJNZ    DIV_SCHL2    ; dekrementiere ZAEHL (B)
                    ;bis ZAEHL=0

    LD      HL,TEILER
    CALL    SCHIEBE_CARRY ;Teiler wiederherstellen
    RET

```

```

;*****
;Unterprogramm SUB
;   addiert Register (HL) von Register (DE), Ergebnis
;   nach Register (IX)
;*****

```

SUB:

```

        PUSH    BC
        LD      BC,BIN_BYTE
        EX      DE,HL
        ADD     HL,BC           ;alle Zeiger ans Registerende
        EX      DE,HL           ;(+BIN_BYTE)
        ADD     HL,BC
        ADD     IX,BC           ;Zähler setzen und
        LD      B,C
        AND     A               ;Carry löschen
SUB_SCHLEIFE:
        ;wiederhole:
        DEC     DE
        DEC     HL
        DEC     IX             ; Zeiger dekrement.,
        LD      A,(DE)         ; Byte laden,
        SBC     A,(HL)         ; subtrahieren und
        LD      (IX),A         ; Ergebnis speichern
        DJNZ    SUB_SCHLEIFE   ; Zähler (B) dekrem.
        POP     BC             ;bis Zähler=0
        RET

```

```

;*****
;Unterprogramm SETZE1
;   setzt REGISTER (HL) auf 1
;*****

```

SETZE1:

```

        PUSH    BC
        LD      B,BIN_BYTE     ;Zähler auf Anzahl Byte
SETZE1_SCHLEIFE:
        ;wiederhole:
        LD      (HL),0         ; ein Byte löschen
        INC     HL             ; Zeiger inkrementieren,
        DJNZ    SETZE1_SCHLEIFE ; Zähler (B) dekrementieren
        ;bis Zähler=0
        DEC     HL             ;Zeiger inkrementieren
        INC     (HL)           ;und Byte auf 1 erhöhen
        POP     BC
        RET

```

```

;*****
;Unterprogramm PLUS2
;   addiert 2 zu Register (HL)
;*****

```

PLUS2:

```

        PUSH    BC
        LD      BC,BIN_BYTE     ;Zeiger ans Registerende

```

```

        ADD    HL,BC          ;(+BIN_BYTE) (kein Carry!)
        LD     B,C           ;Zähler setzen
        LD     A,2           ;Summand = 2
PLUS2_SCHLEIFE:
        DEC    HL            ; Zeiger dekrementieren
        ADC    A,(HL)        ; ein Byte addieren
        LD     (HL),A        ; und wieder speichern
        LD     A,0           ; Akku=0 für folgende Bytes
        DJNZ   PLUS2_SCHLEIFE ; Zähler (B) dekrementieren
        POP    BC            ;bis Zähler=0
        RET

```

```

;*****
;Unterprogramme SCHIEBE, SCHIEBE_CARRY
; schiebt Register (HL) eine Binärstelle nach links
;*****

```

```

SCHIEBE_CARRY:
        PUSH   BC
        PUSH   AF            ;Carry retten
        LD     BC,BIN_BYTE   ;Zeiger ans Registerende
        ADD    HL,BC         ;(+BIN_BYTE)
        POP    AF            ;Carry restaurieren
        JR     SCHIEBE1      ;und schieben

SCHIEBE:
        PUSH   BC
        LD     BC,BIN_BYTE   ;Zeiger ans Registerende
        ADD    HL,BC         ;(+BIN_BYTE) (kein Carry!)

SCHIEBE1:
        LD     B,C           ;Zähler (B) setzen

SCHIEBE_SCHL:
        DEC    HL            ; Zeiger dekrement.,
        RL     (HL)          ; byteweise schieben
        DJNZ   SCHIEBE_SCHL ; Zähler (B) dekrem.
        POP    BC            ;bis Zähler=0
        RET

```

```

;*****
;Unterprogramm SCHIEBE_RECHTS
; schiebt Register (HL) eine Binärstelle nach rechts
;*****

```

```

SCHIEBE_RECHTS:
        PUSH   BC
        LD     B,BIN_BYTE   ;Zähler setzen
        AND    A             ;Carry löschen

SCHIEBE_R:
        RR     (HL)          ; byteweise schieben
        INC    HL            ; Zeiger inkrement.
        DJNZ   SCHIEBE_R    ; Zähler (B) dekrem.
        POP    BC            ;bis Zähler=0
        RET

```

```

;*****
;Unterprogramm MOV
;      kopiert Register (HL) nach Register (DE)
;*****

```

MOV:

```

      PUSH    BC
      LD      BC,BIN_BYTE      ;Anzahl Byte = BIN_BYTE
      LDIR                    ;und schieben
      POP     BC
      RET

```

```

;*****
;Unterprogramm NULL
;      Prüft, ob Register (HL) gleich Null
;*****

```

NULL:

```

      PUSH    BC
      LD      B,BIN_BYTE      ;Anzahl Byte = BIN_BYTE
      XOR     A                ;A=0

```

NULL_SCHLEIFE:

```

;wiederhole:
      OR      (HL)            ; ODER-Verkn. mit Byte
      INC     HL              ; Zeiger inkrementieren
      DJNZ    NULL_SCHLEIFE  ; Zähler (B) dekrement.
                                ;bis Zähler=0
      AND     A                ;Bedingungsflags setzen
      POP     BC
      RET

```

```

;*****
;Unterprogramm WANDLE_DEZ
;      wandelt PRIM in eine Dezimalzahl (nach DEZ_REG)
;*****

```

WANDLE_DEZ:

```

      LD      HL,PRIM
      LD      DE,BIN_REG
      CALL    MOV              ;PRIM nach BIN_REG
      LD      HL,DEZ_REG       ;alte Dezimalzahl löschen:
      LD      B,DEZ_STELL/2

```

CLEAR_DEZ:

```

;wiederhole:
      LD      (HL),0           ; ein Byte löschen
      INC     HL              ; Zeiger inkrementieren
      DJNZ    CLEAR_DEZ       ; Zähler (B) dekrementieren
                                ;bis Zähler=0

```

WANDL_SCHLEIFE:

```

      LD      DE,BIN_STELL     ;alle Binärbits wandeln:
                                ;wiederhole:
      LD      HL,BIN_REG
      CALL    SCHIEBE          ; BIN_REG := BIN_REG*2
                                ; DEZ_REG := DEZ_REG*2
      LD      B,DEZ_STELL/2    ;
                                ; + Carry

```

```

WANDL_SCHLEIFE1:      ; wiederhole:
    DEC    HL          ; Zeiger dekrementieren
    LD     A,(HL)      ; Byte laden
    ADC    A,(HL)      ; mit Carry addieren
    DAA                ; dezimal korrigieren!
    LD     (HL),A       ; speichern
    DJNZ   WANDL_SCHLEIFE1 ; Zähler (B) dekrement.
                        ; bis Zähler=0
    DEC    DE          ; Zähler (DE) dekrement.
    LD     A,D
    OR     E            ; bis Zähler=0
    JR     NZ,WANDL_SCHLEIFE
    RET

```

```

;*****
;Unterprogramm AUSGABE
;*****

```

```

FLAG    EQU    0

```

```

AUSGABE:

```

```

    RES    FLAG,C      ;Flag löschen,
    LD     HL,DEZ_REG   ;Zeiger und
    LD     B,DEZ_STELL/2 ;Zähler setzen

```

```

AUSGABE_SCHLEIFE:

```

```

    RLD                ; Dezimaldigit (Halbbyte)
    CALL   EINE_STELLE ; holen und ausgeben
    RLD                ; dto. für anderes Halb-
    CALL   EINE_STELLE ; byte (Digit)
    INC    HL          ; Zeiger auf nächstes Byte
    DJNZ   AUSGABE_SCHLEIFE; Zähler (B) dekrementieren
    RET                ;bis Zähler=0

```

```

EINE_STELLE:

```

```

    AND    OFH          ;Digit ausgeben:
                        ;rechtes Halbbyte maskieren
    JR     NZ,SCHREIBE   ;Digit=0? nein: ausgeben
    BIT    FLAG,C        ; ja: Flag gesetzt?
    JR     NZ,SCHREIBE   ; ja: ausgeben (normale
                        ;                               Null)
    LD     E,' '         ; nein: Zwischenraum aus-
    PUSH   BC            ; geben
    PUSH   HL
    LD     C,KONAUSF
    CALL   SYSTEM
    POP    HL
    POP    BC
    RET

```

```

SCHREIBE:

```

```

    SET    0,C          ;Flag setzen
    PUSH   BC           ;Zähler und
    PUSH   HL           ;Zeiger retten.
    ADD    A,'0'        ;Digit -> ASCII

```

```

LD      E,A
LD      C,KONAUSF      ;ausgeben
CALL    SYSTEM          ;mit Systemfunktion
POP     HL              ;Zeiger und
PCP     BC              ;Zähler wiederherstellen
RET

```

```

;*****
;
;Datenbereich
;*****

```

```

DEZ_REG:
        DS      DEZ_STELL/2      ;Speicher für Dezimalzahl
BIN_REG:
        DS      BIN_BYTE
PRIM:
        DS      BIN_BYTE      ;Speicher für Binärzahlen
TEILER:
        DS      BIN_BYTE
QUOTIENT:
        DS      BIN_BYTE
REST:
        DS      BIN_BYTE

        END      START

```

Z80-Assembler

pass 1:

+++++

pass 2:

+++++

no fatal error(s)

SUM=8D36

CRC=1EDB

TITLE Primzahlen - Multi-Tasking, Programm 3

```

;*****
;
;Definition von Systemadressen
;*****

```

```

WSTART EQU      0              ;Warmstartadresse

SYSTEM EQU      00005H        ;Adresse für Aufruf von
                               ;Systemfunktionen

```



```

;*****
;Definition von Systemfunktionen
;*****

DRUCKF EQU 5 ;Drucker-Ausgabe

;*****
;Definition von ASCII-Konstanten
;*****

CR EQU ODH ;Cursor an Zeilenanfang
; (Carriage Return)
LF EQU OAH ;Cursor in neue Zeile
; (Line Feed)

;*****
;Definition von TRUE/FALSE (WAHR/FALSCH)
;*****

FALSE EQU 0
TRUE EQU NOT FALSE

;*****
;Sonstige Konstanten
;*****

DEZ_STELL EQU 10 ;maximal 10 Dezimalstellen
; 2 hoch 32 = 4*10 hoch 9
BIN_STELL EQU 32 ;32-Bit Binärzahl

BIN_BYTE EQU BIN_STELL/8 ;Anzahl Bytes d. Binärzahlen

;*****
;Hauptprogramm für Multitasking
; ändert BIOS-Einsprung für Konsolstatus und -eingabe
;*****
HAUPT:
    LD HL,(WSTART+1) ;Adresse Warmstart in BIOS-
    REPT 3 ;tabelle + 3 = Adresse
    INC HL ;Konsolstatus
    ENDM
    LD DE,BIOS_ST ;Zeiger DE auf interne
    ; Sprungtabelle
    LD B,6 ;Zähler für 6 Byte setzen
HAUPT_SCHLEIFE:
    ; wiederhole:
    LD C,(HL) ; Byte aus BIOS-Tabelle
    LD A,(DE) ; merken, ersetzen durch
    LD (HL),A ; Byte aus interner Tab.
    LD A,C ; und in interner Tab.
    LD (DE),A ; speichern
    INC DE ; Zeiger für interne und
    INC HL ; Biostabelle erhöhen
    DJNZ HAUPT_SCHLEIFE ; Zähler (B) dekrementieren
    ; bis Zähler=0

```

```

LD      SP,STACK      ;Stack und Stackpointer für
                      ;Primzahlenprogramm vorber-
                      ;reiten:
LD      HL,START      ;Startadresse auf Stack
PUSH    HL             ; (RET=Sprung nach START!)
PUSH    AF             ;4 Registerpaare auf Stack
PUSH    BC
PUSH    DE
PUSH    HL
LD      (SP_PRIM),SP  ;Stackpointer ablegen
JP      WSTART        ;Warmstart

```

```

;*****
;Unterprogramme STATUS und INPUT
;      ersetzen BIOS-PROGRAMME für Konsolstatus und -eingabe
;*****

```

```

BIOS_ST:              ;Diese beiden Sprungbefehle
JP      STATUS        ;werden von Haupt gegen die
BIOS_IN:              ;beiden der BIOS-Sprung-
JP      INPUT          ;tabelle getauscht
STATUS:
CALL    BIOS_ST       ;Konsolstatus abfragen

OR      A              ;Rücksprung wenn Zeichen
RET     NZ             ;bereit
TRANSFER PRIM:        ;sonst Primzahlenprogramm:
PUSH    BC             ;Register retten
PUSH    DE
PUSH    HL
PUSH    IX
LD      (SP_SYS),SP   ;Stackpointer (System) retten

LD      SP,(SP_PRIM)  ;Stackpointer (PRIM) holen
POP     HL             ;4 Registerpaare laden
POP     DE
POP     BC
POP     AF
RET                     ;und Rücksprung an letzte
                      ;Adresse

TRANSFER SYS:
PUSH    AF             ;4 Registerpaare
PUSH    BC             ;für PRIM retten
PUSH    DE
PUSH    HL
LD      (SP_PRIM),SP  ;dto. Stackpointer für PRIM
LD      SP,(SP_SYS)   ;Stackpointer für System
POP     IX
POP     HL             ;und Register für System
POP     DE             ;holen
POP     BC
XOR     A              ;=keine Taste bereit
RET                     ;zurück ins System

```

```

INPUT:                                ;wiederhole:
      CALL    STATUS                  ; Status abfragen (mit PRIM)
      JR      Z,INPUT                ;bis Zeichen bereit
      JP      BIOS_IN                ;Zeichen holen

```

```

;*****
;Hauptprogramm PRIMZAHLEN
;*****

```

```

START:
      LD      HL,PRIM                ;Setze PRIM = 1
      CALL    SETZE1

```

```

SCHLEIFE0:
      LD      B,7                    ; wiederhole:
                                      ; Zähler für 7 Zahlen/Zeile

```

```

SCHLEIFE:
      ; wiederhole:
      ; wiederhole:
      PUSH    BC                    ; Zähler retten

      LD      HL,PRIM
      CALL    PLUS2                ; erhöhe PRIM um 2

      CALL    PRUEFE                ; Prüfe ob PRIM Primzahl

      POP     BC                    ; Zähler restaurieren

      JR      NC,SCHLEIFE          ; bis Primzahl gefunden

      PUSH    BC                    ; Zähler retten

      CALL    WANDLE_DEZ           ; Wandle PRIM in Dezimalzahl

      CALL    AUSGABE              ; Gib PRIM auf Drucker aus

      POP     BC                    ; Zähler restaurieren

      DJNZ    SCHLEIFE             ; und dekrementieren
                                      ; bis Zähler (B) gleich Null

      LD      E,CR                  ; Gib CR/LF auf Drucker aus
      LD      C,DRUCKF
      CALL    SYSTEM

      LD      E,LF
      LD      C,DRUCKF
      CALL    SYSTEM

      JR      SCHLEIFE0
                                      ;bis RESET

```

```

;*****
;Unterprogramm PRUEFE
;   Prüft, ob PRIM eine Primzahl ist
;*****

```

PRUEFE:

```

        LD      HL,TEILER      ;Setze TEILER = 1
        CALL    SETZE1

```

PRUEFE_SCHL:

;wiederhole:

```

        LD      HL,TEILER
        CALL    PLUS2          ; erhöhe TEILER um 2
        CALL    DIV            ; Dividiere PRIM/TEILER
                                ; Ergebnis: QUOTIENT, REST
        LD      HL,REST        ;bis REST gleich Null
        CALL    NULL          ; --> keine Primzahl
        RET      Z             ; --> Carry=0

        LD      IX,BIN_REG     ;oder TEILER >= QUOTIENT
        LD      DE,TEILER      ; TEILER - QUOTIENT
        LD      HL,QUOTIENT     ; Carry=Vorzeichen Ergebnis
        CALL    SUB            ; Carry=1: TEILER<QUOTIENT
                                ; Carry=0: TEILER>=QUOTIENT
        JR      C,PRUEFE_SCHL  ; --> Primzahl
        SCF                    ; --> Carry:=1
        RET

```

```

;*****
;Unterprogramm DIV
;   dividiert PRIM durch TEILER
;   Ergebnis in QUOTIENT, Rest in REST
;*****

```

DIV:

```

        LD      B,1            ;Setze ZAEHL = 1
DIV_SCHL1:
        INC     B              ; erhöhe ZAEHL um 1
        LD      HL,TEILER
        CALL    SCHIEBE        ; schiebe TEILER links
        LD      IX,BIN_REG
        LD      DE,PRIM
        LD      HL,TEILER
        CALL    SUB
        JR      NC,DIV_SCHL1    ;bis TEILER > PRIM

        LD      HL,PRIM
        LD      DE,REST
        CALL    MOV            ;Kopiere PRIM nach REST
        LD      HL,QUOTIENT
        CALL    SETZE1
        DEC     (HL)           ;Setze QUOTIENT gleich Null

```

```

DIV_SCHL2:                                ;wiederhole:
LD      IX,BIN_REG                        ; BIN_REG :=
LD      DE,REST                          ; REST
LD      HL,TEILER                        ; -TEILER
CALL    SUB                              ; Ergebnis positiv?
LD      HL,BIN_REG
LD      DE,REST
CALL    NC,MOV                          ; ja: BIN_REG nach REST
CCF                                          ; komplementiere Carry-Flag
LD      HL,QUOTIENT
CALL    SCHIEBE_CARRY                    ; Carry-Flag in QUOTIENT
LD      HL,TEILER
CALL    SCHIEBE_RECHTS                  ; schiebe TEILER rechts
DJNZ    DIV_SCHL2                      ; dekrementiere ZAEHL (B)
                                           ;bis ZAEHL=0

LD      HL,TEILER
CALL    SCHIEBE_CARRY                    ;Teiler wiederherstellen
RET

```

```

;*****
;Unterprogramm SUB
;      addiert Register (HL) von Register (DE), Ergebnis
;      nach Register (IX)
;*****
SUB:

```

```

      PUSH    BC
      LD      BC,BIN_BYTE
      EX      DE,HL
      ADD     HL,BC                      ;alle Zeiger ans Registerende
      EX      DE,HL                      ;(+BIN_BYTE)
      ADD     HL,BC
      ADD     IX,BC                      ;Zähler setzen und
      LD      B,C
      AND     A                          ;Carry löschen
SUB_SCHLEIFE:                          ;wiederhole:
      DEC     DE
      DEC     HL
      DEC     IX                        ; Zeiger dekrement.,
      LD      A,(DE)                    ; Byte laden,
      SBC     A,(HL)                    ; subtrahieren und
      LD      (IX),A                    ; Ergebnis speichern
      DJNZ    SUB_SCHLEIFE              ; Zähler (B) dekrem.
      POP     BC                        ;bis Zähler=0
      RET

```

```

;*****
;Unterprogramm SETZE1
;      setzt REGISTER (HL) auf 1
;*****

```

```

SETZE1:
      PUSH    BC
      LD      B,BIN_BYTE                ;Zähler auf Anzahl Byte

```

```

SETZE1_SCHLEIFE:                ;wiederhole:
    LD      (HL),0                ; ein Byte löschen.
    INC     HL                    ; Zeiger inkrementieren,
    DJNZ    SETZE1_SCHLEIFE      ; Zähler (B) dekrementieren
                                ;bis Zähler=0
    DEC     HL                    ;Zeiger inkrementieren
    INC     (HL)                  ;und Byte auf 1 erhöhen
    POP     BC
    RET

;*****
;Unterprogramm PLUS2
;    addiert 2 zu Register (HL)
;*****
PLUS2:
    PUSH    BC
    LD      BC,BIN_BYTE           ;Zeiger ans Registerende
    ADD     HL,BC                 ;(+BIN_BYTE) (kein Carry!)
    LD      B,C                   ;Zähler setzen
    LD      A,2                   ;Summand = 2
PLUS2_SCHLEIFE:                ;wiederhole:
    DEC     HL                    ; Zeiger dekrementieren
    ADC     A,(HL)                ; ein Byte addieren
    LD      (HL),A                ; und wieder speichern
    LD      A,0                   ; Akku=0 für folgende Bytes
    DJNZ    PLUS2_SCHLEIFE       ; Zähler (B) dekrementieren
    POP     BC                    ;bis Zähler=0
    RET

;*****
;Unterprogramme SCHIEBE, SCHIEBE_CARRY
;    schiebt Register (HL) eine Binärstelle nach links
;*****
SCHIEBE_CARRY:
    PUSH    BC
    PUSH    AF                    ;Carry retten
    LD      BC,BIN_BYTE           ;Zeiger ans Registerende
    ADD     HL,BC                 ;(+BIN_BYTE)
    POP     AF                    ;Carry restaurieren
    JR      SCHIEBE1              ;und schieben
SCHIEBE:
    PUSH    BC
    LD      BC,BIN_BYTE           ;Zeiger ans Registerende
    ADD     HL,BC                 ;(+BIN_BYTE) (kein Carry!)
SCHIEBE1:
    LD      B,C                   ;Zähler (B) setzen
SCHIEBE_SCHL:
    DEC     HL                    ; Zeiger dekrement.,
    RL      (HL)                  ; byteweise schieben
    DJNZ    SCHIEBE_SCHL         ; Zähler (B) dekrem.
    POP     BC                    ;bis Zähler=0
    CALL    TRANSFER_SYS
    RET

```

```

;*****
;Unterprogramm SCHIEBE_RECHTS
;    schiebt Register (HL) eine Binärstelle nach rechts
;*****

```

SCHIEBE_RECHTS:

```

    PUSH    BC
    LD      B,BIN_BYTE    ;Zähler setzen
    AND     A              ;Carry löschen
SCHIEBE_R:    ;wiederhole:
    RR      (HL)          ; byteweise schieben
    INC     HL             ; Zeiger inkrement.,
    DJNZ    SCHIEBE_R     ; Zähler (B) dekrem.
    POP     BC             ;bis Zähler=0
    RET

```

```

;*****
;Unterprogramm MOV
;    kopiert Register (HL) nach Register (DE)
;*****

```

MOV:

```

    PUSH    BC
    LD      BC,BIN_BYTE    ;Anzahl Byte = BIN_BYTE
    LDIR                      ;und schieben
    POP     BC
    RET

```

```

;*****
;Unterprogramm NULL
;    Prüft, ob Register (HL) gleich Null
;*****

```

NULL:

```

    PUSH    BC
    LD      B,BIN_BYTE    ;Anzahl Byte = BIN_BYTE
    XOR     A              ;A=0
NULL_SCHLEIFE:    ;wiederhole:
    OR      (HL)          ; ODER-Verkn. mit Byte
    INC     HL             ; Zeiger inkrementieren
    DJNZ    NULL_SCHLEIFE ; Zähler (B) dekrement.
                    ;bis Zähler=0
    AND     A              ;Bedingungsflags setzen
    POP     BC
    RET

```

```

;*****
;
;Unterprogramm WANDLE_DEZ
;   wandelt PRIM in eine Dezimalzahl (nach DEZ_REG)
;*****

```

WANDLE_DEZ:

```

    LD    HL,PRIM
    LD    DE,BIN_REG
    CALL  MOV        ;PRIM nach BIN_REG
    LD    HL,DEZ_REG ;alte Dezimalzahl löschen:
    LD    B,DEZ_STELL/2

```

CLEAR_DEZ:

```

;wiederhole:
    LD    (HL),0      ; ein Byte löschen
    INC   HL           ; Zeiger inkrementieren
    DJNZ  CLEAR_DEZ   ; Zähler (B) dekrementieren
                    ;bis Zähler=0

```

```

    LD    DE,BIN_STELL ;alle Binärbits wandeln:

```

WANDL_SCHLEIFE:

```

;wiederhole:
    LD    HL,BIN_REG
    CALL  SCHIEBE      ; BIN_REG := BIN_REG*2
                    ; DEZ_REG := DEZ_REG*2
    LD    B,DEZ_STELL/2 ;          + Carry

```

WANDL_SCHLEIFE1:

```

; wiederhole:
    DEC   HL           ; Zeiger dekrementieren
    LD    A,(HL)       ; Byte laden
    ADC   A,(HL)       ; mit Carry addieren
    DAA                    ; dezimal korrigieren!
    LD    (HL),A       ; speichern
    DJNZ  WANDL_SCHLEIFE1 ; Zähler (B) dekrement.
                    ; bis Zähler=0
    DEC   DE           ; Zähler (DE) dekremen.
    LD    A,D
    OR    E             ;bis Zähler=0
    JR    NZ,WANDL_SCHLEIFE
    RET

```

```

;*****
;
;Unterprogramm AUSGABE
;*****

```

FLAG EQU 0

AUSGABE:

```

    RES   FLAG,C       ;Flag löschen,
    LD    HL,DEZ_REG   ;Zeiger und
    LD    B,DEZ_STELL/2 ;Zähler setzen

```



```

AUSGABE_SCHLEIFE:                ;wiederhole:
    RLD                          ; Dezimaldigit (Halbbyte)
    CALL    EINE_STELLE          ; holen und ausgeben
    RLD                          ; dto. für anderes Halb-
    CALL    EINE_STELLE          ; byte (Digit)
    INC     HL                   ; Zeiger auf nächstes Byte
    DJNZ    AUSGABE_SCHLEIFE; Zähler (B) dekrementieren
    RET                          ;bis Zähler=0

```

```

EINE_STELLE:                      ;Digit ausgeben:
    AND     OFH                  ;rechtes Halbbyte maskieren
    JR      NZ,SCHREIBE          ;Digit=0? nein: ausgeben
    BIT     FLAG,C               ; ja: Flag gesetzt?
    JR      NZ,SCHREIBE          ; ja: ausgeben (normale
                                ;                               Null)
    LD      E,' '                ; nein: Zwischenraum aus-
    PUSH    BC                   ; geben
    PUSH    HL
    LD      C,DRUCKF
    CALL    SYSTEM
    POP     HL
    POP     BC
    RET

```

```

SCHREIBE:
    SET     0,C                  ;Flag setzen
    PUSH    BC                   ;Zähler und
    PUSH    HL                   ;Zeiger retten,
    ADD     A,'0'                ;Digit -> ASCII
    LD      E,A
    LD      C,DRUCKF             ;ausgeben
    CALL    SYSTEM               ;mit Systemfunktion
    POP     HL                   ;Zeiger und
    POP     BC                   ;Zähler wiederherstellen
    RET

```

```

;*****
;Datenbereich
;*****

```

```

SP_PRIM:                          ;Stackpointer für Primzahlen-
    DS      2                    ;programm
SP_SYS:                          ;Stackpointer für System
    DS      2                    ;programme

```

```

DEZ_REG:
    DS      DEZ_STELL/2          ;Speicher für Dezimalzahl
BIN_REG:
    DS      BIN_BYTE
PRIM:
    DS      BIN_BYTE             ;Speicher für Binärzahlen
TEILER:
    DS      BIN_BYTE

```

```
QUOTIENT:
      DS      BIN_BYTE
REST:
      DS      BIN_BYTE

      DS      100           ;viel Platz für Stack!
STACK:
      END      HAUPT
```

Z80-Assembler

pass 1:

+++++

pass 2:

+++++

no fatal error(s)

SUM=BDF6

CRC=6074

TITLE BUCHSTABENWANDLUNG - Programm 1

```

;*****
;Dieses Programm wandelt alle Buchstaben im Textspeicher von
;Klein- auf Großschreibung, außer in Kommentaren oder in An-
;führungsstrichen stehenden Strings
;*****

```

```

;*****
;Definition von Systemadressen
;*****

```

```

SYSTEM EQU     00005H           ;Adresse für Aufruf von
                                ;Systemfunktionen

```

```

;*****
;Definition von Systemfunktionen
;*****

```

```

WSTARTF EQU     0               ;Warmstart

DIRCONF EQU     6               ;Direkte Konsol Ein/Ausgabe

STRAUSF EQU     9               ;Ausgabe von Strings auf CON:

TXTBEGF EQU     247             ;Startadresse des Text-
                                ;speichers abfragen

```

```

;*****
;Definition von ASCII-Konstanten
;*****

```

```

CR     EQU     0DH               ;Cursor an Zeilenanfang
                                ;(Carriage Return)
LF     EQU     0AH               ;Cursor in neue Zeile
                                ;(Line Feed)
EOF    EQU     1AH               ;Markierung für Textende
                                ;(End Of File)

```

```

;*****
;Definition von Flags in Register E
;*****

```

```

KOM     EQU     0               ;Kommentar: Bit 0
STR     EQU     1               ;String:     Bit 1

```

```

;*****
;Hauptprogramm BUCHSTABENWANDLUNG
;*****

```

```

START:  JP      START1

```

```

TEXT:

```

```

        DB      'Klein-Grossbuchstabenwandlung '
        DB      'fuer Programmtext',CR,LF,'$'

```

```

START1:

```

```

        LD      SP,(SYSTEM+1)    ;Stackpointer laden

```

```

        LD      DE,TEXT          ;Meldung ausgeben

```

```

        LD      C,STRAUSF

```

```

        CALL    SYSTEM

```

```

        CALL    VORBEREITUNG     ;alles vorbereiten

```

```

        JR      C,FEHLER         ;Abbruch bei Fehler

```

```

        RES     KOM,E            ;Flag für Kommentar Nullsetzen

```

```

        RES     STR,E            ;Flag für String Nullsetzen

```

```

SCHLEIFE:

```

```

        CALL    HOLE_ZEICHEN     ;ein Zeichen holen,
        JR      C,FEHLER         ;Abbruch bei Fehler und
        AND     7FH              ;höchstes Bit wegmaskieren

```

```

        BIT     KOM,E            ;im Kommentar?
        JR      NZ,KOMMENTAR     ; ja: weiter dort

```

```

        BIT     STR,E            ;im String?
        JR      NZ,STRING        ; ja: dto.

```

```

PRUEFE_KOMM:                                ; nein:

```

```

        CP      ';'              ;Semikolon?

```

```

        JR      NZ,PRUEFE_STR     ; nein: weiter

```

```

KOMM_ANFANG:                                ; ja: Kommentar beginnt

```

```

        SET     KOM,E            ;Flag setzen

```

```

        JR      WANDLE_NICHT     ;und weiter ohne wandeln

```

```

PRUEFE_STR:

```

```

        CP      ' '              ;Anführungsstrich?

```

```

        JR      NZ,WANDLE        ; nein: weiter mit wandeln

```

```

STRING_ANFANG:                              ; ja: String beginnt

```

```

        SET     STR,E            ;Flag setzen

```

```

        JR      WANDLE_NICHT     ;und weiter ohne wandeln

```

```

KOMMENTAR:

```

```

        CP      CR               ;Zeile zu Ende (CR)?

```

```

        JR      NZ,WANDLE_NICHT ; nein: weiter im Komm.

```

```

KOMM_ENDE:
    RES    KOM,E      ; ja: Kommentar zu Ende
    JR     WANDLE     ; Flag rücksetzen
                    ; und weiter mit wandeln

```

```

STRING:
    CP     CR          ; Zeile zu Ende (CR)?
    JR     Z,STRING_ENDE ; ja: String endet
    CP     ' '         ; Anführungsstrich?
    JR     NZ,WANDLE_NICHT ; nein: weiter im String

```

```

STRING_ENDE:
    RES    STR,E      ; Flag rücksetzen
                    ; und weiter mit wandeln

```

```

WANDLE:
    CP     60H         ; kleiner Buchstabe ?
    JR     C,WANDLE_NICHT ; nein: nicht wandeln
    SUB    20H         ; ja: wandeln

```

```

WANDLE_NICHT:
    CALL   SCHREIBE_ZEICHEN ; Zeichen wegschreiben
    JR     C,FEHLER        ; Abbruch bei Fehler

    CP     EOF             ; Textende?
    JR     NZ,SCHLEIFE     ; nein: weiter

```

```

FERTIG:
    CALL   ABSCHLUSS      ; ja: fertig

```

```

FEHLER:
    LD     C,WSTARTF      ; Warmstart
    CALL   SYSTEM

```

```

;*****
;Unterprogramm VORBEREITUNG
; - lädt IX und IY mit Textspeicheranfang
;*****

```

```

VORBEREITUNG:
    LD     C,XTXBEGF      ; Textspeicheranfang
    CALL   SYSTEM        ; holen nach HL
    PUSH   HL             ; und HL
    POP    IX             ; nach IX
    PUSH   HL
    POP    IY             ; und IY
    AND    A              ; Carry Nullsetzen
    RET

```

```

;*****
;Unterprogramm Abschluss
;*****

```

```

ABSCHLUSS:
    RET

```

```

;*****
;
;Unterprogramm HOLE_ZEICHEN
;*****

```

HOLE_ZEICHEN:

```

        LD      A,(IX)
        INC     IX
        AND     A                ;Carry Nullsetzen
        RET

```

```

;*****
;
;Unterprogramm SCHREIBE_ZEICHEN
;*****

```

SCHREIBE_ZEICHEN:

```

        LD      (IY),A
        INC     IY
        AND     A                ;Carry Nullsetzen
        RET

```

PROGRAMM_ENDE:

```

        END     START

```

TITLE BUCHSTABENWANDLUNG - Programm 2

```

;*****
;
;Dieses Programm wandelt alle Buchstaben einer Datei von
;Klein- auf Großschreibung, außer in Kommentaren oder in An-
;führungsstrichen stehenden Strings
;*****

```

```

;*****
;
;Definition von Systemadressen
;*****

```

```

SYSTEM EQU      00005H                ;Adresse für Aufruf von
                                         ;Systemfunktionen

```

```

;*****
;
;Definition von Systemfunktionen
;*****

```

```

WSTARTF EQU     0                    ;Warmstart

DIRCONF EQU     6                    ;Direkte Konsol Ein/Ausgabe

STRAUSF EQU     9                    ;Ausgabe von Strings auf CON:

```

```

STREINF EQU    10           ;Eingabe von Strings mit CON:

OPENF  EQU    15           ;Datei oeffnen

CLOSEF EQU    16           ;Datei schliessen

DELETF EQU    19           ;Datei löschen

READF  EQU    20           ;aus Datei lesen

WRITEF EQU    21           ;in Datei schreiben

MAKEF  EQU    22           ;Datei anlegen

SETDMF EQU    26           ;DMA-Adresse setzen

MAKEFCB EQU    250         ;FCB aus String erzeugen

```

```

;*****
;Definition von ASCII-Konstanten
;*****

```

```

CR      EQU    0DH          ;Cursor an Zeilenanfang
                          ;(Carriage Return)
LF      EQU    0AH          ;Cursor in neue Zeile
                          ;(Line Feed)
EOF     EQU    1AH          ;Markierung für Textende
                          ;(End Of File)

```

```

;*****
;Definition von Flags in Register E
;*****

```

```

KOM     EQU    0           ;Kommentar: Bit 0
STR     EQU    1           ;String:   Bit 1

```

```

;*****
;Hauptprogramm BUCHSTABENWANDLUNG
;*****

```

```

START:  JP      START1

```

```

TEXT:

```

```

    DB      'Klein-Grossbuchstabenwandlung '
    DB      'fuer Programtext',CR,LF,'$'

```

```

START1:

```

```

    LD      SP,(SYSTEM+1)  ;Stackpointer laden

    LD      DE,TEXT        ;Meldung ausgeben
    LD      C,STRAUSF
    CALL    SYSTEM

```

```

CALL    VORBEREITUNG    ;alles vorbereiten
JR      C,FEHLER        ;Abbruch bei Fehler

RES     KOM,E           ;Flag für Kommentar Nullsetzen
RES     STR,E           ;Flag für String Nullsetzen

SCHLEIFE:
CALL    HOLE_ZEICHEN    ;ein Zeichen holen,
JR      C,FEHLER        ;Abbruch bei Fehler und
AND     7FH             ;höchstes Bit wegmaskieren

BIT     KOM,E           ;im Kommentar?
JR      NZ,KOMMENTAR    ; ja: weiter dort

BIT     STR,E           ;im String?
JR      NZ,STRING       ; ja: dto.

PRUEFE_KOMM:           ; nein:
CP      ';'             ;Semikolon?
JR      NZ,PRUEFE_STR   ; nein: weiter
KOMM_ANFANG:           ; ja: Kommentar beginnt
SET     KOM,E           ;Flag setzen
JR      WANDLE_NICHT    ;und weiter ohne wandeln

PRUEFE_STR:
CP      ''''           ;Anführungsstrich?
JR      NZ,WANDLE       ; nein: weiter mit wandeln
STRING_ANFANG:         ; ja: String beginnt
SET     STR,E           ;Flag setzen
JR      WANDLE_NICHT    ;und weiter ohne wandeln

KOMMENTAR:
CP      CR              ;Zeile zu Ende (CR)?
JR      NZ,WANDLE_NICHT ; nein: weiter im Komm.

KOMM_ENDE:             ; ja: Kommentar zu Ende
RES     KOM,E           ;Flag rücksetzen
JR      WANDLE          ;und weiter mit wandeln

STRING:
CP      CR              ;Zeile zu Ende (CR)?
JR      Z,STRING_ENDE   ; ja: String endet
CP      ''''           ;Anführungsstrich?
JR      NZ,WANDLE_NICHT ; nein: weiter im String

STRING_ENDE:
RES     STR,E           ;Flag rücksetzen
                        ;und weiter mit wandeln

WANDLE:
CP      60H             ; kleiner Buchstabe ?
JR      C,WANDLE_NICHT ; nein: nicht wandeln
SUB     20H             ; ja: wandeln

```


WANDLE_NICHT:

CALL SCHREIBE_ZEICHEN ;Zeichen wegschreiben
JR C,FEHLER ;Abbruch bei Fehler

CP EOF ;Textende?
JR NZ,SCHLEIFE ; nein: weiter

FERTIG:

CALL ABSCHLUSS ; ja: fertig

FEHLER:

LD C,WSTARTF ;Warmstart
CALL SYSTEM

```
;*****
;Unterprogramm VORBEREITUNG
;- lädt Datei in Speicher
;- lädt IX und IY mit Textanfang
;*****
```

VORB_FRAGE: DB 'Name der Eingabe-Datei ? \$'
VORB_FEHLER1: DB 'Datei nicht gefunden! ',CR,LF,'\$'
VORB_FEHLER2: DB 'Speicher voll! ',CR,LF,'\$'

SATZ_ZAHL: DW 0 ;Anzahl gelesene Datensätze

PUFFER: DB 16,0 ;Eingabe-Puffer für Datei-
DS 16 ;name, max. 16 Zeichen

EINGABE_FCB: DS 36,0 ;Dateisteuerblock Eingabe

VORBEREITUNG:

LD DE,VORB_FRAGE ;nach Dateiname fragen,
LD C,STRAUSF
CALL SYSTEM

LD DE,PUFFER ;Name in Puffer lesen
LD C,STREINF
CALL SYSTEM

LD DE,PUFFER+2 ;und File-Control-Block
LD IY,PUFFER ;erzeugen
LD IX,EINGABE_FCB
LD C,MAKEFCB
CALL SYSTEM ;(Carry=0 => Fehler)
CCF ;Carry komplementieren
RET C ;Rücksprung mit Carry=1
;bei Fehler

```

LD      DE,EINGABE_FCB  ;Datei öffnen
LD      C,OPENF
CALL    SYSTEM          ;A=OFFH falls Datei
CP      OFFH            ;      nicht gefunden
JR      NZ,VORBEREITUNG1

LD      DE,VORB_FEHLER1 ;Textausgabe "Datei
LD      C,STRAUSF       ;      nicht gefunden"
CALL    SYSTEM
SCF                                ;und mit Carry=1 für Fehler
RET                                ;zurück ins Hauptprogramm

```

VORBEREITUNG1:

```

LD      BC,-1           ;Zähler auf -1
LD      DE,PROGRAMM_ENDE ;Anfangsadresse zum
                                ;Einlesen

```

LESE_SCHLEIFE:

```

LD      HL,(SYSTEM+1)   ;Test ob Speicher voll
DEC      H
SBC      HL,DE
JR      NC,LESE_SCHLEIFE1

```

```

LD      DE,VORB_FEHLER2 ;Textausgabe "Speicher voll"
LD      C,STRAUSF
CALL    SYSTEM
SCF                                ;und mit Carry=1 für Fehler
RET                                ;zurück ins Hauptprogramm

```

LESE_SCHLEIFE1:

```

LD      HL,128          ;DE+128 ergibt die Adresse
ADD      HL,DE          ;für den nächsten Durchlauf
PUSH     HL             ;diese Adresse retten,
INC      BC             ;Zähler inkrementieren,
PUSH     BC             ;retten

```

```

LD      C,SETDMAF       ;und momentane Transfer-
CALL     SYSTEM         ;Adresse (DMA-Adr.) setzen

```

```

LD      DE,EINGABE_FCB  ;Record (Datensatz) von
LD      C,READF         ;Diskette einlesen
CALL    SYSTEM          ;A=0 falls erfolgreich,
                                ;sonst Dateiende
POP      BC             ;Zähler restaurieren,
POP      DE             ;nächste DMA-Adresse in DE
AND      A
JR      Z,LESE_SCHLEIFE ;und weiter falls nicht Ende

```

```

LD      (SATZ_ZAHL),BC  ;sonst Zähler weglegen,

```

```

LD      IX,PROGRAMM_ENDE ;Textanfang nach IX
LD      IY,PROGRAMM_ENDE ;und IY,
AND      A               ;Carry Nullsetzen,
RET                                ;fertig

```

```

;*****
;Unterprogramm Abschluss
;      - speichert Datei auf Diskette
;*****

```

```

ABSCH_FRAGE:  DB      'Name der Ausgabe-Datei ? $'
ABSCH_FEHLER1: DB      'Datei existiert!',CR,LF,'$'
ABSCH_FEHLER2: DB      'Inhaltsverzeichnis voll!',CR,LF,'$'
ABSCH_FEHLER3: DB      'Diskette voll!',CR,LF,'$'

```

```

AUSGABE_FCB:  DS      36,0      ;Dateisteuerblock Ausgabe

```

ABSCHLUSS:

```

      LD      DE,ABSCH_FRAGE  ;nach Dateiname fragen
      LD      C,STRAUSF
      CALL    SYSTEM

      LD      DE,PUFFER      ;Name in Puffer lesen
      LD      C,STREINF
      CALL    SYSTEM

      LD      DE,PUFFER+2    ;und File-Control-Block
      LD      IX,PUFFER      ;erzeugen
      LD      IX,AUSGABE_FCB
      LD      C,MAKEFCB
      CALL    SYSTEM        ;(Carry=0 => Fehler)
      RET      NC           ;Rücksprung bei Fehler

      LD      DE,AUSGABE_FCB ;Datei öffnen
      LD      C,OPENF
      CALL    SYSTEM        ;A=OFFH falls Datei
      CP      OFFH          ;      nicht existiert!
      JR      Z,ABSCHLUSS1

      LD      DE,AUSGABE_FCB ;falls Datei existiert:
      LD      C,CLOSEF      ;Datei schliessen,
      CALL    SYSTEM

      LD      DE,ABSCH_FEHLER1 ;Fehlertext ausgeben
      LD      C,STRAUSF
      CALL    SYSTEM
      RET                      ;und Rücksprung

```

ABSCHLUSS1:

```

      LD      DE,AUSGABE_FCB ;Datei anlegen
      LD      C,MAKEF
      CALL    SYSTEM        ;A=OFFH bei Fehler
      CP      OFFH
      JR      NZ,ABSCHLUSS2

```

```
LD    DE,ABSCH_FEHLER2 ;Fehlertext ausgeben
LD    C,STRAUSF
CALL  SYSTEM
```

```
RET                                ;und Rücksprung
```

ABSCHLUSS2:

```
LD    DE,PROGRAMM_ENDE ;Anfangsadresse zum
                                ;Abspeichern
LD    BC,(SATZ_ZAHL)    ;und Anzahl Datensätze
                                ;laden
```

SCHREIB_SCHLEIFE:

```
LD    A,B                ;BC=0: kein Datensatz mehr
OR     C                  ;zu schreiben
JR     Z,SCHREIB_ENDE
```

```
LD    HL,128              ;DE+128 ergibt die Adresse
ADD    HL,DE              ;für den nächsten Durchlauf
PUSH   HL                 ;diese Adresse retten
```

```
DEC    BC                 ;Zähler dekrementieren
PUSH   BC                 ;und retten,
```

```
LD    C,SETDMAF           ;momentane Transfer-
CALL  SYSTEM              ;Adresse (DMA-Adr.) setzen
```

```
LD    DE,AUSGABE_FCB     ;Record (Datensatz) auf
LD    C,WRITEF            ;Diskette schreiben
CALL  SYSTEM              ;A=0 falls erfolgreich,
                                ;sonst Diskette voll
```

```
POP    BC                 ;Zähler restaurieren und
POP    DE                 ;nächste DMA-Adresse in DE
AND    A
JR     Z,SCHREIB_SCHLEIFE ;und weiter falls nicht voll
```

```
LD    DE,AUSGABE_FCB     ;sonst Datei löschen,
LD    C,DELETEF
CALL  SYSTEM
```

```
LD    DE,ABSCH_FEHLER3 ;Fehlertext ausgeben
LD    C,STRAUSF
CALL  SYSTEM
```

```
RET                                ;und Rücksprung
```

SCHREIB_ENDE:

```
LD    DE,AUSGABE_FCB     ;Datei schliessen
LD    C,CLOSEF
CALL  SYSTEM
```

```
RET                                ;und Rücksprung
```

```

;*****
;
;Unterprogramm HOLE_ZEICHEN
;*****

```

HOLE_ZEICHEN:

```

    LD      A,(IX)
    INC     IX
    AND     A           ;Carry Nullsetzen
    RET

```

```

;*****
;
;Unterprogramm SCHREIBE_ZEICHEN
;*****

```

SCHREIBE_ZEICHEN:

```

    LD      (IY),A
    INC     IY
    AND     A           ;Carry Nullsetzen
    RET

```

PROGRAMM_ENDE:

```

    END     START

```

TITLE BUCHSTABENWANDLUNG - PROGRAMM 3

```

;*****
;
;Dieses Programm wandelt alle Buchstaben einer Datei von
;Klein- auf Großschreibung, außer in Kommentaren oder in An-
;führungsstrichen stehenden Strings
;*****

```

```

;*****
;
;Definition von Systemadressen
;*****

```

```

SYSTEM EQU     00005H             ;Adresse für Aufruf von
                                  ;Systemfunktionen

```

```

;*****
;
;Definition von Systemfunktionen
;*****

```

```

WSTARTF EQU    0                 ;Warmstart

DIRCONF EQU    6                 ;Direkte Konsol Ein/Ausgabe

STRAUSF EQU    9                 ;Ausgabe von Strings auf CON:

OPENF EQU      15                ;Datei oeffnen

```

```

CLOSEF EQU 16 ;Datei schliessen
DELETEF EQU 19 ;Datei löschen
READF EQU 20 ;aus Datei lesen
WRITEF EQU 21 ;in Datei schreiben
MAKEF EQU 22 ;Datei anlegen
RENAMEF EQU 23 ;Datei umbenennen
SETDMAF EQU 26 ;DMA-Adresse setzen

```

```

;*****
;Definition von ASCII-Konstanten
;*****

```

```

CR EQU 0DH ;Cursor an Zeilenanfang
; (Carriage Return)
LF EQU 0AH ;Cursor in neue Zeile
; (Line Feed)
EOF EQU 1AH ;Markierung für Textende
; (End Of File)

```

```

;*****
;Definition von Flags in Register E
;*****

```

```

KOM EQU 0 ;Kommentar: Bit 0
STR EQU 1 ;String: Bit 1

```

```

;*****
;Hauptprogramm BUCHSTABENWANDLUNG
;*****

```

```

START: JP START1

```

```

TEXT:

```

```

DB 'Klein-Grossbuchstabenwandlung '
DB 'fuer Programmtext',CR,LF,'$'

```

```

START1:

```

```

LD SP,(SYSTEM+1) ;Stackpointer laden

LD DE,TEXT ;Meldung ausgeben
LD C,STRAUSF
CALL SYSTEM

```

```

CALL VORBEREITUNG ;alles vorbereiten
JR C,FEHLER ;Abbruch bei Fehler

```

```

RES KOM,E ;Flag für Kommentar Nullsetzen
RES STR,E ;Flag für String Nullsetzen

```

SCHLEIFE:

| | | |
|------|--------------|----------------------------|
| CALL | HOLE_ZEICHEN | ;ein Zeichen holen, |
| JR | C,FEHLER | ;Abbruch bei Fehler und |
| AND | 7FH | ;höchstes Bit wegmaskieren |

| | | |
|-----|--------------|-------------------|
| BIT | KOM,E | ;im Kommentar? |
| JR | NZ,KOMMENTAR | ; ja: weiter dort |

```

BIT      STR,E          ;im String?
JR        NZ,STRING     ; ja: dto.

```

| | | |
|--------------|---------------|--------------------------|
| PRUEFE_KOMM: | | ; nein: |
| CP | ',' | ;Semikolon? |
| JR | NZ,PRUEFE_STR | ; nein: weiter |
| KOMM_ANFANG: | | ; ja: Kommentar beginnt |
| SET | KOM,E | ;Flag setzen |
| JR | WANDLE NICHT | ;und weiter ohne wandeln |

```

PRUEFE_STR:
    CP            ' '           ;Anführungsstrich?
    JR            NZ,WANDLE     ; nein: weiter mit wandeln
STRING_ANFANG:
    SET          STR,E         ; ja: String beginnt
    JR            WANDLE NICHT ;und weiter ohne wandeln

```

```

KOMMENTAR:
      CP      CR              ;Zeile zu Ende (CR)?
      JR      NZ,WANDLE NICHT ; nein: weiter im Komm.

```

| | | | |
|------|--------|--|-------------------------|
| KOMM | ENDE: | | ; ja: Kommentar zu Ende |
| RES | KOM,E | | ;Flag rücksetzen |
| JR | WANDLE | | ;und weiter mit wandeln |

```

STRING:
    CP      CR          ;Zeile zu Ende (CR)?
    JR      Z,STRING_ENDE ; ja: String endet
    CP      ' '         ;Anführungsstrich?
    JR      NZ,WANDLE NICHT ; nein: weiter im String

```

```

STRING_ENDE:
    RES     STR,E           ;Flag rücksetzen
                                ;und weiter mit wandeln

```

```
WANDLE:
    CP      60H      ; kleiner Buchstabe ?
    JR      C,WANDLE_NICHT ; nein: nicht wandeln
    SUB     20H      ; ja: wandeln
```

```

WANDLE_NICHT:
CALL    SCHREIBE_ZEICHEN ;Zeichen wegschreiben
JR      C,FEHLER          ;Abbruch bei Fehler

CP      EOF                ;Textende?
JR      NZ,SCHLEIFE       ; nein: weiter

```


FERTIG:

CALL ABSCHLUSS ; ja: fertig

FEHLER:

LD DE,AUSGABE_FCB ; ".\$\$\$\$" FCB löschen falls

LD C,DELETF ; vorhanden

CALL SYSTEM

LD C,WSTARTF ; Warmstart

CALL SYSTEM

```

;*****
;Unterprogramm VORBEREITUNG
;      - lädt Datei in Speicher
;      - lädt IX und IY mit Textanfang
;*****

```

VORB_FEHLER1: DB 'Datei nicht gefunden!',CR,LF,'\$'

VORB_FEHLER2: DB 'Inhaltsverzeichnis voll!',CR,LF,'\$'

```

STANDARD_FCB EQU 005CH ;Standard-Dateisteuerblock
;vom Betriebssystem

```

VORBEREITUNG:

```

LD HL,STANDARD_FCB ;Dateiname für
LD DE,EINGABE_FCB ;Eingabedatei
LD BC,12 ;(Laufwerk+Name+Typ=12)
LDIR

```

```

LD HL,STANDARD_FCB ;Dateiname für
LD DE,AUSGABE_FCB ;Ausgabedatei ( ".$$$$" )
LD BC,9 ;(Laufwerk+Name=9)
LDIR

```

```

LD HL,STANDARD_FCB ;Dateiname für
LD DE,BACKUP_FCB ;Sicherungsdatei ( ".BAK" )
LD BC,9 ;(Laufwerk+Name=9)
LDIR

```

```

LD DE,EINGABE_FCB ;Datei öffnen
LD C,OPENF
CALL SYSTEM ;A=OFFH falls Datei
CP OFFH ; nicht gefunden
JR NZ,VORBEREITUNG1
LD DE,VORB_FEHLER1 ;Textausgabe "Datei
LD C,STRAUSF ; nicht gefunden"
CALL SYSTEM
SCF ;und mit Carry=1 für Fehler
RET ;zurück ins Hauptprogramm

```

VORBEREITUNG1:

```

LD DE,AUSGABE_FCB ;Datei löschen falls
LD C,DELETF ;vorhanden
CALL SYSTEM

```



```

LD      DE,AUSGABE_FCB ;Datei anlegen
LD      C,MAKEF
CALL    SYSTEM          ;A=OFFH bei Fehler
CP      OFFH
JR      Z,VORBEREITUNG2
AND     A                ;ok: Carry löschen und zurück
RET     ;ins Hauptprogramm, sonst

```

VORBEREITUNG2:

```

LD      DE,VORB_FEHLER2 ;Fehlertext ausgeben
LD      C,STRAUSF
CALL    SYSTEM
SCF     ;und mit Carry=1 für Fehler
RET     ;zurück ins Hauptprogramm

```

```

;*****
;Unterprogramm Abschluss
;      - speichert Datei auf Diskette
;*****

```

BACKUP_FCB:

```

DB      0,'          BAK' ;Dateisteuerblock Sicherungs-
DS      24,0            ;datei (backup)

```

ABSCHLUSS:

```

CALL    SCHREIBE_DS      ;letzten Datensatz schreiben
RET     C                ;und zurück falls Fehler

```

```

LD      DE,AUSGABE_FCB ;sonst Ausgabedatei
LD      C,CLOSEF       ;schliessen
CALL    SYSTEM

```

```

LD      DE,BACKUP_FCB  ;alte Backup-Datei löschen
LD      C,DELETEF     ;falls vorhanden
CALL    SYSTEM

```

```

LD      HL,BACKUP_FCB+1 ;Eingabedatei in ".BAK" umbe-
LD      DE,EINGABE_FCB+16+1 ;nennen
LD      BC,11          ;(Name+Typ=11)
LDIR
LD      DE,EINGABE_FCB
LD      C,RENAMEF
CALL    SYSTEM

```

```

LD      HL,EINGABE_FCB+1 ;Ausgabedatei ".$$$"
                        ;in alte Eingabedatei umbe-
LD      DE,AUSGABE_FCB+16+1 ;nennen
LD      BC,11          ;(Name+Typ=11)
LDIR
LD      DE,AUSGABE_FCB
LD      C,RENAMEF
CALL    SYSTEM
RET     ;und Rücksprung

```

```

;*****
;Unterprogramm HOLE_ZEICHEN
;*****

```

```

EINGABE_FCB:
    DS      36,0           ;Dateisteuerblock Eingabe

```

```

EINGABE_ZEIGER:
    DB      0             ;Zeiger im Eingabepuffer

```

```

EINGABE_PUFFER:
    DS      128           ;Puffer für einen Datensatz

```

```

HOLE_ZEICHEN:

```

```

    LD      A,(EINGABE_ZEIGER)
    LD      C,A           ;Pufferposition berechnen
    LD      B,0
    LD      HL,EINGABE_PUFFER
    ADD     HL,BC

```

```

    INC     A             ;Zeiger inkrementieren,
    AND     7FH           ;höchstes Bit wegmaskieren
    LD      (EINGABE_ZEIGER),A ;und Zeiger speichern

```

```

    LD      A,C
    AND     A             ;wenn Zeiger auf Null, dann
    PUSH    HL
    CALL    Z,LESE_DS     ;zuerst Datensatz einlesen
    POP     HL
    RET     C             ;zurück bei Fehler

```

```

    LD      A,(HL)        ;Zeichen aus Puffer laden
    AND     A             ;Carry=Null (kein Fehler)
    RET                     ;und zurück

```

```

LESE_DS:                               ;Datensatz lesen:

```

```

    PUSH    DE
    LD      DE,EINGABE_PUFFER
    LD      C,SETDMAF      ;Transferadresse setzen
    CALL    SYSTEM

```

```

    LD      DE,EINGABE_FCB ;und Datensatz einlesen
    LD      C,READF
    CALL    SYSTEM

```

```

    POP     DE
    AND     A
    RET     Z             ;zurück falls ok, sonst
    SCF                     ;mit Carry zurück
    RET

```

F

86

```

;*****
;Unterprogramm SCHREIBE_ZEICHEN
;*****

```

AUSGABE_FCB:

```

    DB    0,'      $$$' ;Dateisteuerblock Ausgabe
    DS    24,0

```

AUSGABE_ZEIGER:

```

    DB    0          ;Zeiger im Ausgabepuffer

```

AUSGABE_PUFFER:

```

    DS    128          ;Puffer für einen Datensatz

```

SCHREIBE_ZEICHEN:

```

    PUSH   AF          ;Zeichen retten
    LD     A,(AUSGABE_ZEIGER)

    LD     C,A          ;Pufferposition berechnen
    LD     B,0
    LD     HL,AUSGABE_PUFFER
    ADD    HL,BC

    POP    AF          ;Zeichen holen und
    PUSH   AF          ;wieder retten,
    LD     (HL),A       ;Zeichen in Puffer legen
    LD     A,C

    INC    A            ;Zeiger inkrementieren,
    AND    7FH          ;höchstes Bit wegmaskieren
    LD     (AUSGABE_ZEIGER),A ;und Zeiger speichern

    AND    A            ;wenn Zeiger auf Null, dann
    CALL   Z,SCHREIBE_DS ;Datensatz wegschreiben

    POP    BC          ;Zeichen holen, aber Flags
    LD     A,B          ;lassen!
    RET

```

SCHREIBE_DS: ;Datensatz schreiben:

```

    PUSH   DE
    LD     DE,AUSGABE_PUFFER

    LD     C,SETDMAF    ;Transferadresse setzen
    CALL   SYSTEM
    LD     DE,AUSGABE_FCB ;und Datensatz schreiben

```

```
LD      C,WRITEF
CALL    SYSTEM
```

```
POP     DE
```

```
AND     A
```

```
RET     Z
```

```
;zurück falls ok, sonst
```

```
SCF
```

```
;mit Carry zurück
```

```
RET
```

```
PROGRAMM_ENDE:
```

```
END     START
```