Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS94/95
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **Hilfsmittel :** Keine (auch kein Taschenrechner)

Hinweis : Versehen Sie die Assembler-Programme der Aufgaben 1 und 3 mit hinreichenden Kommentaren.

Aufgabe 1 : (60 Punkte)

Schreiben Sie ein vollständiges Assembler-Programm (Ziel : EXE-Datei) für folgenden - in Pascal formulierten - Ablauf :

```
program aufgabe1;

type string80 = string[80];

var s1,s2 : string80;

procedure insert(var ziel,quelle : string80; position : integer);

var i : integer;

begin
  for i := ord(ziel[0]) downto position do
    ziel[i+ord(quelle[0])] := ziel[i];
  for i := 1 to ord(quelle[0]) do
    ziel[position+i-1]:= quelle[i];
  ziel[0] := chr(ord(ziel[0])+ord(quelle[0]))
end;

begin
  s1 := 'Staatsexamen erfolgreich absolviert';
  s2 := 'sfach Prozeßtechnik';
  writeln(s1);
  writeln(s2);
  insert(s1,s2,13);
  writeln(s1)
end.
```

Bedenke : - Die Standardfunktion ORD liefert für Argumente ordinalen Datentyps die zugehörige Ordnungszahl
- Die Standardfunktion CHR liefert für Argumente des Datentyps Integer das zugehörige Zeichen des Rechnerzeichensatzes (z.B. ASCII-Code)

Aufgabe 2 : (20 Punkte)

Nehmen Sie begründet Stellung zur These :

Die Assemblersprache ist die zur Lösung technischer Probleme am besten geeignete Programmiersprache.

Aufgabe 3 : (40 Punkte)

Entwickeln Sie ein Assembler-Programm (Ziel : EXE-Datei) zur Multiplikation zweier Vektoren.

$$\vec{A} * \vec{B} = (a_1,a_2,...,a_n) * (b_1,b_2,...,b_n)$$
$$= a_1*b_1 + a_2*b_2 + ... + a_n*b_n$$

Aufgabe 4 : (45 + 15 Punkte)

Übersetzen Sie das nachfolgende Assembler-Programm in ein COM-Programm :

```
CDSEG     SEGMENT
          ASSUME CS:CDSEG, DS:CDSEG
          ORG   100h

AUFGABE4  PROC NEAR
          MOV   SI,0
          MOV   BX,0
          MOV   CX,2
LOOP1:    MOV   DX,CX
          MOV   CX,8
LOOP2:    MOV   AL,QUELLE[SI]
          MOV   ZIEL[SI],AL
          CALL  UP1
          LOOP  LOOP2
          CALL  UP2
          MOV   CX,DX
          LOOP  LOOP1
          MOV   AH,09h
          MOV   DX,OFFSET FERTIG
          INT   21h
          INT   20h
AUFGABE4  ENDP

UP1       PROC NEAR
          INC   SI
          RET
UP1       ENDP

UP2       PROC NEAR
          INC   BX
          CALL  UP1
          RET
UP2       ENDP

QUELLE    DB    1,2,3,4,5,6,7,8,0,9,10,11,12,13,14,15,16,0
ZIEL      DB    18 DUP (?)
FERTIG    DB    'geschafft...$'

CDSEG     ENDS
          END AUFGABE4
```

Ermitteln Sie per Schreibtischtest den Inhalt der Register SI und BX vor Ausführung des Befehls INT 20h.

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SS95
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **Hilfsmittel :** Keine (auch kein Taschenrechner)

Hinweis : Versehen Sie die Assembler-Programme der Aufgaben 1 und 3 mit hinreichenden Kommentaren.

Aufgabe 1 : (60 Punkte)

Schreiben Sie ein vollständiges Assembler-Programm (Ziel : EXE-Datei) für folgenden - in Pascal formulierten - Ablauf :

```
program aufgabe1;

type string80 = string[80];

var s : string80;

procedure delete(var quelleziel       : string80;
                     position, anzahl : integer);

var i : integer;

begin
  for i := 0 to ord(quelleziel[0])-position-anzahl do
    quelleziel[position+i]:= quelleziel[position+i+anzahl];
  quelleziel[0] := chr(ord(quelleziel[0])-anzahl);
end;

begin
  s := 'Staatsexamensfach Prozeßtechnik bald absolviert';
  writeln(s);
  delete(s,33,5);
  writeln(s);
end.
```

Bedenke : - Die Standardfunktion ORD liefert für Argumente ordinalen Datentyps die zugehörige Ordnungszahl
- Die Standardfunktion CHR liefert für Argumente des Datentyps Integer das zugehörige Zeichen des Rechnerzeichensatzes (z.B. ASCII-Code)

Aufgabe 2 : (20 Punkte)

Sie sind Leiter des Projekts *Entwicklung der Software für die Robotersteuerung ROBBYMAT*. Welche Programmiersprachen und Software-Tools werden Sie auswählen (incl. Begründung) ?

Aufgabe 3 : (40 Punkte)

Entwickeln Sie ein Assembler-Programm (Ziel : EXE-Datei) zur Berechnung des arithmetischen Mittels μ und der mittleren absoluten Abweichung MAD bezogen auf μ. Die Berechnung soll auf folgenden Datenfeldern basieren :

T = Tabelle bestehend aus maximal hundert Integer-Werten
N = tatsächliche Anzahl der Werte in T

Arithmetisches Mittel :

$$\mu = \frac{1}{N} * \sum_{I=1}^{N} T(I) = \frac{1}{N} * (T(1) + T(2) + ... + T(N))$$

Mittlere absolute Abweichung bezogen auf μ :

$$MAD = \frac{1}{N} * \sum_{I=1}^{N} |T(I) - \mu| = \frac{1}{N} * (|T(1) - \mu| + |T(2) - \mu| + ... + |T(N) - \mu|)$$

Die Eingabe der Tabelle und die Ausgabe der Ergebnisse sind *nicht* gefordert.

Aufgabe 4 : (40 Punkte)

Ermitteln Sie per Schreibtischtest den Inhalt der Datenfelder A, B und C vor Ausführung des Befehls INT 20h.

```
CDSEG     SEGMENT
          ASSUME CS:CDSEG,DS:CDSEG
          ORG   100h

AUFGABE4  PROC
          MOV   A,1
          MOV   B,1
          MOV   C,0
          MOV   DX,A
          PUSH  DX
          MOV   DX,OFFSET B
          PUSH  DX
          CALL  UP2
          INT   20h
AUFGABE4  ENDP

UP1       PROC
          PUSH  BP
          MOV   BP,SP
          PUSH  AX
          MOV   AX,[BP+4]
          ADD   C,AX
          POP   AX
          POP   BP
          RET   2
UP1       ENDP

UP2       PROC
          PUSH  BP
          MOV   BP,SP
          PUSH  AX
          PUSH  DX
          PUSH  SI
          MOV   AX,[BP+6]
          ADD   C,AX
          INC   WORD PTR [BP+6]
          MOV   SI,[BP+4]
          MOV   AX,[SI]
          ADD   C,AX
          INC   WORD PTR [SI]
          MOV   DX,[BP+6]
          PUSH  DX
          CALL  UP1
          POP   SI
          POP   DX
          POP   AX
          POP   BP
          RET   4
UP2       ENDP

A         DW    ?
B         DW    ?
C         DW    ?

CDSEG     ENDS
          END AUFGABE4
```

Aufgabe 5 : (20 Punkte)

Übersetzen Sie die nachfolgenden Assembler-Programmausschnitte (Ziel : COM-Dateien) in Maschinencode.

```
100             MOV   BX,TABL[SI]
                MOV   TABL[SI],BX
                MOV   BX,OP
                MOV   OP,BX
```

```
300             CALL  UP
                ...
400  UP         PROC
                ...
     UP         ENDP
```

```
200  UP         PROC
                ...
     UP         ENDP
                ...
300             CALL  UP
```

```
300             JE    MARKE
                ...
330  MARKE:     ...
```

```
2D0  MARKE:     ...
                ...
300             JE    MARKE
```

└ *Offsetadressen*

In allen zugrundeliegenden Assembler-Programmen stehen ab der Offsetadresse 150 jeweils folgende Daten :

```
OP      DW    ?
TABL    DW    10 DUP (?)
```

Hinweis : Alle Offsetadressen sind in hexadezimaler Form angegeben

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS95/96
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

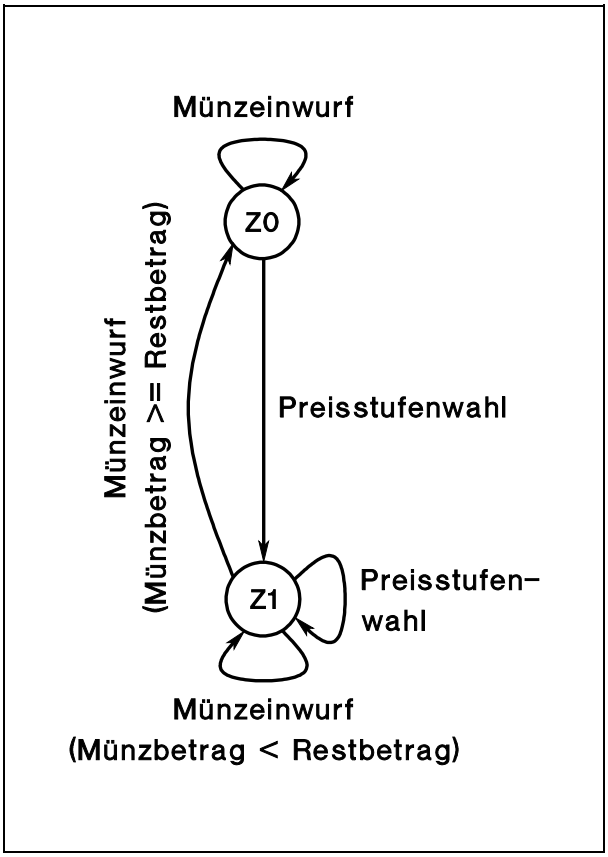
Dauer : 180 Minuten **Hilfsmittel :** Keine (auch kein Taschenrechner)

Hinweis : Versehen Sie die Assembler-Programme der Aufgaben 1, 2 und 3 mit hinreichenden Kommentaren.

Aufgabe 1 : (90 Punkte)

Zur Beschreibung der Funktionsweise technischer Anlagen werden häufig Final-State-Diagramme verwendet. Diese sind durch Zustände und Zustandsübergänge sowie Fehler- und gültige Endzustände gekennzeichnet.

Beispiel : Fahrkartenautomat



Preisstufen : A (2 DM), B (4 DM), C (6 DM)

Münzen : 1 DM, 2 DM, 5 DM

Zustandsübergänge :

- Z0 $\wedge$ Münzeinwurf $\rightarrow$ Z0
  Aktion : Eingeworfene Münze auswerfen
- Z0 $\wedge$ Preisstufenwahl $\rightarrow$ Z1
  Aktion : Restbetrag aus gewählter Preisstufe ermitteln und Restbetrag anzeigen
- Z1 $\wedge$ Preisstufenwahl $\rightarrow$ Z1
  Aktion : Gewählte Preisstufe ignorieren
- Z1 $\wedge$ Münzeinwurf (Münzbetrag < Restbetrag) $\rightarrow$ Z1
  Aktion : Restbetrag um Münzbetrag vermindern und Restbetrag anzeigen
- Z1 $\wedge$ Münzeinwurf (Münzbetrag >= Restbetrag) $\rightarrow$ Z0
  Aktion : Fahrkarte und Wechselgeld auswerfen

Es gibt weder Fehler- noch gültige Endzustände, da der Fahrkartenautomat ständig betriebsbereit sein muß.

Entwickeln Sie ein vollständiges Assemblerprogramm (Ziel : EXE-Datei) unter Beachtung folgender Randbedingungen :

- Leere Münzschächte (d.h. kein Wechselgeld verfügbar) bzw. volle Münzschächte (d.h. kein Platz für neue Münzen verfügbar) müssen nicht berücksichtigt werden.

- Wartungsvorgänge (z.B. eingenommene Münzen entnehmen oder Fahrkarten nachfüllen) müssen ebenfalls nicht berücksichtigt werden.

- Simulation der Preisstufenwahl und des Münzeinwurfs durch Tastatureingabe (A, B, C, 1, 2, 5).

- Simulation des Münz- bzw. Wechselgeldauswurfs sowie des Fahrkartenauswurfs durch entsprechende Bildschirmausgabe (1 DM, 2 DM, 5 DM, Preisstufe A, Preisstufe B, Preisstufe C)

- Die Möglichkeiten der Code- und Datenstrukturierung durch Segmente und Prozeduren sowie Parameterübergaben an Unterprogramme und lokale Variablen in Unterprogrammen sollen ausgenutzt werden.

Dialogbeispiel :

(E) A
(A) Restbetrag : 2 DM
(E) 2
(A) Preisstufe A
(E) A
(A) Restbetrag : 2 DM
(E) 1
(A) Restbetrag : 1 DM
(E) 1
(A) Preisstufe A
(E) A
(A) Restbetrag : 2 DM
(E) 5
(A) Preisstufe A 2 DM 1 DM
(E) 1
(A) 1 DM
(E) A
(A) Restbetrag : 2 DM
(E) B
(A) Restbetrag : 2 DM
(E) 2
(A) Preisstufe A

(E) = Eingabe
(A) = Ausgabe

Aufgabe 2 : (30 Punkte)

Entwickeln Sie ein Assembler-Programm (Ziel : EXE-Datei) zur Addition zweier Matrizen. Die Ein- und Ausgabe der Matrizen ist *nicht* gefordert.

$$\underline{\underline{A}} + \underline{\underline{B}} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & & & \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2n}+b_{2n} \\ \dots & & & \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \dots & a_{mn}+b_{mn} \end{pmatrix}$$

Aufgabe 3 : (30 + 30 Punkte)

a) Schreiben Sie ein vollständiges Assembler-Programm (Ziel : EXE-Datei) für folgenden - in Pascal formulierten - Ablauf. Die Schnittstelle zwischen Haupt- und Unterprogramm soll sich an den Techniken von Borland Turbo-Pascal (d.h. Parameterübergaben via Stack) orientieren.

```
program aufgabe3;

var a,b : integer;

procedure abcxyz(var y : integer; z : integer);

begin
  y := y-1;
  z := z+1;
  writeln(chr(y),chr(z))
end;

begin
  a := 66;
  b := 66;
  writeln(chr(a),chr(b));
  abcxyz(a,b);
  writeln(chr(a),chr(b));
  abcxyz(b,a);
  writeln(chr(a),chr(b))
end.
```

Bedenke : Die Standardfunktion CHR liefert für Argumente des Datentyps Integer das zugehörige Zeichen des Rechnerzeichensatzes (z.B. ASCII-Code)

b) Ermitteln Sie per Schreibtischtest (Basis : Assemblerprogramm !) die Ausgaben der WRITELN-Anweisungen.

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SS96
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **Hilfsmittel :** Keine (auch kein Taschenrechner)

Hinweis : Versehen Sie das Assembler-Programm der Aufgabe 1 mit hinreichenden Kommentaren.

Aufgabe 1 : (90 Punkte)

Entwickeln Sie - zuzüglich eines Hauptprogramms (Ziel : EXE-Datei) als Testumgebung - ein Unterprogramm zum Löschen von Zeichen aus einer Zeichenkette. Die Zeichenkette entspricht den Konventionen von Borland Turbo-Pascal. Das Unterprogramm ist eine Prozedur mit Datenübergabe via Referenzparameter (Zeichenkette) und Wertparametern (Position und Anzahl). Alle lokalen Variablen des Unterprogramms werden auf dem Stack abgelegt. Stellen Sie Ihrer Lösung ein Pseudocodeprogramm voran.

Beispiel :

Zeichenkette : ABCDEFGH Position : 3 Länge : 4

^H	A	B	C	D	E	F	G	H	¿	...	¿

1. Schritt : Kopieren des hinteren Restes (GH) ¿ = undefiniert

^H	A	B	G	H	E	F	G	H	¿	...	¿

2. Schritt : Längenkorrektur

^D	A	B	G	H	E	F	G	H	¿	...	¿

Zeichenkette : ABGH

Aufgabe 2 : (30 Punkte)

Diskutieren Sie die Bedeutung der Assemblersprache. Gehen Sie dabei insbesondere auf die Vor- und Nachteile sowie Einsatzmöglichkeiten der Assemblersprache ein.

Aufgabe 3 : (20 + 40 Punkte)

a) Schreiben Sie ein vollständiges Pascal-Programm für folgenden - in Assembler formulierten - Ablauf.

b) Welche Aktionen führt das Programm AUFGABE3 durch ? Belegen Sie Ihre Aussagen durch eine Kommentierung des Assembler-Programms sowie einen - auf Schlüsselbefehle reduzierten - Schreibtischtest für die Eingabe *BGLQV*.

```
CDSEG     SEGMENT
          ASSUME CS:CDSEG, DS:CDSEG
          ORG   100h

AUFGABE3  PROC NEAR
          MOV   CX,N
          MOV   BX,OFFSET FELD
          MOV   SI,1
M1:       MOV   AH,08
          INT   21h
          CMP   AL,32
          JB    M1
          MOV   [BX+SI],AL
          MOV   AH,02
          MOV   DL,AL
          INT   21h
          INC   SI
          LOOP  M1

          MOV   CX,N
          MOV   BX,OFFSET FELD
          MOV   BYTE PTR [BX],0
          MOV   SI,1
M2:       MOV   AL,[BX+SI]
          ADD   [BX],AL
          INC   SI
          LOOP  M2

          MOV   CX,N
          INC   CX
          MOV   BX,OFFSET FELD
          MOV   SI,0
M3:       MOV   AH,2
          MOV   DL,[BX+SI]
          CMP   DL,32
          JAE   M4
          MOV   DL,32
M4:       INT   21h
          INC   SI
          LOOP  M3

          INT   20h
AUFGABE3  ENDP

          N     EQU 5
          FELD  DB 6 DUP (?)

CDSEG     ENDS
          END AUFGABE3
```

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS96/97
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **externe Hilfsmittel :** keine

Aufgabe 1 : (120 Punkte)

Entwickeln Sie zwei Assembler-Programme (STR.ASM und VAL.ASM) zur Reimplementierung der Borland Turbo-Pascal Prozeduren STR und VAL. Ziel der Entwicklung ist die Einbindung der Objektcodes (STR.OBJ und VAL.OBJ) obiger Assembler-Programme mittels der Compiler-Direktive {$L ...} in ein Borland Turbo-Pascal Hauptprogramm.

Versehen Sie Ihre Assembler-Programme mit hinreichenden Kommentaren (incl. Beschreibung der Algorithmen durch Skizzen und / oder Pseudocodeprogramme).

Beachten Sie die nachfolgenden Hilfsmittel und Hinweise :

(a) Online-Hilfe Borland Turbo-Pascal Version 6.0 (Auszug) :

Str ***Prozedur***

Konvertiert den als x übergebenen Wert in eine Zeichenfolge und speichert das Ergebnis in der als s übergebenen Stringvariablen.

Syntax:
```
Str(x [: width [: decimals ]]; var s: string);
```

Die optionalen Parameter "width" und "decimals" haben dieselbe Funktion wie bei Write.

Val ***Prozedur***

Interpretiert die Zeichenfolge des Strings s als numerischen Wert und speichert das Ergebnis in der als v übergebenen Variablen, die vom Typ Integer oder Real ist.

Syntax:
```
Val(s: string; var v; var code: Integer);
```

"code" enthält nach fehlerfreier Ausführung den Wert 0; ansonsten die Position des Zeichens in s, das den Fehler verursacht hat.

(b) Einschränkungen der Reimplementierung :

- keine optionalen Parameter "width" und "decimals"
- nur Integer´s, keine Real´s
- String´s nur als Referenzparameter

(c) Beispiel eines Borland Turbo-Pascal Hauptprogramms :

```
Program Aufgabe1;

Var S    : String;
    V    : Integer;
    Code : Integer;

Procedure Str (X : Integer; Var S : String); External;
{$L STR}  { STR.OBJ, d.h. Objektcode }

Procedure Val (Var S : String; Var V, Code : Integer); External;
{$L VAL}  { VAL.OBJ, d.h. Objektcode }

Begin
  Str (12345, S); Writeln (S); { 12345 }
  Str (-12345, S); Writeln (S); { -12345 }
  S := ´12345´;
  Val (S, V, Code); Writeln (V, ´ ´, Code); { 12345 0 }
  S := ´-12345´;
  Val (S, V, Code); Writeln (V, ´ ´, Code); { -12345 0 }
  S := ´12.45´;
  Val (S, V, Code); Writeln(V, ´ ´, Code) { 0 3 }
End.
```

Aufgabe 2 : (60 Punkte)

Welche Aktionen führt das Programm AUFGABE2 durch ? Belegen Sie Ihre Aussagen durch eine Kommentierung des Programms sowie einen Schreibtischtest.

```
CDSEG         SEGMENT
              ASSUME CS:CDSEG, DS:CDSEG
              ORG   100h

AUFGABE2      PROC NEAR
              MOV   BX,0
              MOV   CX,16
HEX_LOOP:
              MOV   DL,LINE[BX]
              PUSH  CX
              MOV   CL,4
              SHR   DL,CL
              POP   CX
              CALL  WRITE_DIGIT
              MOV   DL,LINE[BX]
              AND   DL,0Fh
              CALL  WRITE_DIGIT
              MOV   AH,02h
              MOV   DL,' '
              INT   21h
              INC   BX
              LOOP  HEX_LOOP
              MOV   BX,0
              MOV   CX,16
ASCII_LOOP:
              MOV   AH,02h
              MOV   DL,LINE[BX]
              CMP   DL,20h
              JGE   DRUCKBAR
              MOV   DL,'.'
DRUCKBAR:
              INT   21h
              INC   BX
              LOOP  ASCII_LOOP
              INT   20h
AUFGABE2      ENDP

WRITE_DIGIT   PROC NEAR
              ADD   DL,30h
              CMP   DL,3Ah
              JL    DIGIT_OK
              ADD   DL,07h
DIGIT_OK:
              MOV   AH,02h
              INT   21h
              RET
WRITE_DIGIT   ENDP

LINE          DB    30h, 31h, 32h, 33h, 34h, 35h, 36h, 37h
              DB    38h, 39h, 3Ah, 3Bh, 3Ch, 3Dh, 3Eh, 3Fh

CDSEG         ENDS
              END AUFGABE2
```

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SS97
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **externe Hilfsmittel :** keine

Hinweis : Versehen Sie die Assembler-Programme der Aufgaben 1 und 2 mit hinreichenden Kommentaren.

Aufgabe 1 : (90 Punkte)

Entwickeln Sie - zuzüglich eines Hauptprogramms (Ziel : EXE-Datei) als Testumgebung - ein Unterprogramm zum Einfügen einer Zeichenkette in eine Zeichenkette. Die Zeichenketten entsprechen den Konventionen von Borland Turbo-Pascal. Das Unterprogramm ist eine Prozedur mit Datenübergabe via Referenzparametern (Zeichenketten) und Wertparameter (Position). Alle lokalen Variablen des Unterprogramms werden auf dem Stack abgelegt. Stellen Sie Ihrer Lösung ein Pseudocode-, Pascal- oder C-Programm voran.

Beispiel : 1. Zeichenkette : CDEF, 2. Zeichenkette : ABGH, Position : 3

^D	C	D	E	F	¿	¿	¿	¿	¿	...	¿
^D	A	B	G	H	¿	¿	¿	¿	¿	...	¿

1. Schritt : Kopieren des hinteren Teils (GH) der 2. Zeichenkette

^D	A	B	G	H	¿	¿	G	H	¿	...	¿

2. Schritt : Kopieren der 1. Zeichenkette in die 2. Zeichenkette

^D	A	B	C	D	E	F	G	H	¿	...	¿

3. Schritt : Längenkorrektur der 2. Zeichenkette

^H	A	B	G	H	E	F	G	H	¿	...	¿

¿ = undefiniert

Aufgabe 2 : (90 Punkte)

Schreiben Sie ein vollständiges Assembler-Programm (Ziel : EXE-Datei) für folgenden - in Pascal formulierten - Ablauf :

```
Program Aufgabe2;

Uses Crt;  { Bibliothek mit Funktionen und Prozeduren für Bild-
             schirmausgabe und Tastatureingabe }

Const XUpLeft     =  1;
      YUpLeft     =  1;
      XDownRight  = 80;
      YDownRight  = 25;

      SpecialKey  =  #0;
      Esc         = #27;
      CursorUp    = #72;
      CursorDown  = #80;
      CursorLeft  = #75;
      CursorRight = #77;

Type ScreenTyp = Array [1..25,1..80,1..2] of Byte;

Var Screen : ScreenTyp absolute $B800:$0000;

    XPos      : Integer;
    YPos      : Integer;
    ASCii     : Byte;
    Attribute : Byte;
    Key       : Char;

Procedure Up (Var YPos : Integer);

Begin
  If YPos > YUpLeft Then YPos := YPos - 1
End;

Procedure Down (Var YPos : Integer);

Begin
  If YPos < YDownRight Then YPos := YPos + 1
End;

Function Left (XPos : Integer) : Integer;

Begin
  If XPos > XUpLeft Then Left := XPos - 1 Else Left := XPos
End;

Function Right (XPos : Integer) : Integer;

Begin
  If XPos < XDownRight Then Right := XPos + 1 Else Right := XPos
End;
```

```
Function ReadCursorKey : Char;

Var Key : Char;

Begin
  Repeat
    Repeat
      Key := ReadKey  { Zeichen von Tastatur ohne Echo einlesen }
    Until Key in [SpecialKey, Esc];
    If Key = SpecialKey Then Key := ReadKey
  Until Key in [CursorUp, CursorDown, CursorLeft, CursorRight, Esc];
  ReadCursorKey := Key
End;

Begin
  ClrScr;
  XPos := XUpLeft;
  YPos := YUpLeft;
  ASCii := 0;
  Attribute := 0;
  Repeat
    Screen[YPos, XPos, 1] := ASCii + Ord ('A');
    Screen[YPos, XPos, 2] := Attribute;
    Key := ReadCursorKey;
    Case Key of
      CursorUp    : Up (YPos);
      CursorDown  : Down (YPos);
      CursorLeft  : XPos := Left (XPos);
      CursorRight : XPOs := Right (XPOs)
    End;
    ASCii := (ASCii + 1) Mod 26;
    Attribute := (Attribute + 1) mod 128;
  Until Key = Esc;
  ClrScr;
End.
```

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS97/98
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Hinweis : Versehen Sie das Assembler-Programm der Aufgabe 1 mit hinreichenden Kommentaren.

Aufgabe 1 : (90 Punkte)

Übersetzen Sie das nachfolgende Pascal-Programm in ein äquivalentes Assemblerprogramm (8086, EXE).

Die Unterprogramme sollen gemäß den Konventionen von Borland TurboPascal realisiert werden (Parameterübergabe über den Stack, lokale Variablen auf dem Stack).

Hinweis : Das Pascal-Programm analysiert den "Hardware-Zeichengenerator", der für die Zeichen des Standard-ASCII-Codes das Pixelmuster festlegt.

Aufgabe 2 : (30 Punkte)

Vergleichen Sie anhand prägnanter Kriterien maschinenorientierte Programmiersprachen (z.B. 80x86-Assembler) mit höheren Programmiersprachen (z.B. Objekt-Pascal).

```
program aufgabe1;

var a : integer;
    b : array [1..128*8] of byte absolute $F000:$FA6E;
    c : array [1..128*8] of char absolute $F000:$FA6E;
    i : integer;
    j : integer;

procedure writebin(x:byte);

var i : integer;

begin
  for i := 1 to 8 do begin
    if (x and $80) <> 0 then
      write('1')
    else
      write('0');
    x := x shl 1
  end
end;

procedure writehex(x:byte);

var hlx : byte;
    i   : integer;

begin
  for i := 1 to 2 do begin
    if i = 1 then
      hlx := x and $F0 shr 4
    else
      hlx := x and $0F;
    case hlx of
      $0..$9 : write(chr(hlx+ord('0')));
      $A..$F : write(chr(hlx-$A+ord('A')))
    end
  end
end;

begin
  for i := 1 to 128 do begin
    write(i:3,':',' ':3);
    for j := 1 to 8 do begin
      a := (i-1) * 8 + j;
      writehex(b[a]);
      write(' ')
    end;
    write(' ':3);
    for j := 1 to 8 do begin
      a := (i-1) * 8 + j;
      if ord(c[a]) in [32..126] then
        write(c[a])
      else
        write('.')
    end;
    writeln
  end;
  for i := 1 to 128 do begin
    writeln;
    for j := 1 to 8 do begin
      a := (i-1) * 8 + j;
      writebin(b[a]);
      write(' ':3);
      writehex(b[a]);
      writeln
    end;
  end
end.
```

Aufgabe 3 : (60 Punkte)

Übersetzen Sie die beiden nachfolgenden Pascal-Programme in äquivalente Assembler-Programme (8086, EXE). Analog Aufgabe 1 sollen die Unterprogramme den Konventionen von Borland TurboPascal entsprechen. Ermitteln Sie für das zweite Programm per Schreibtischtest (Basis : Assembler-Programm !) den Inhalt der Variablen Z.

```
program dreia;

const n = 5;

type feldtyp = array [0..n] of integer;

var feld : feldtyp;

procedure up(var upfeld : feldtyp; upn : integer);

var i : integer;

begin
  upfeld[0] := 0;
  for i := 1 to upn do upfeld[0] := upfeld[0] + upfeld[i]
end;

begin

  { Eingabe der Variablen Feld nicht gefordert }

  up(feld, n);

  { Ausgabe der Variablen Feld nicht gefordert }

end.
```

```
program dreib;

var z : integer;

function up(x, y : integer) : integer;

begin
  if y > 1 then
    up := up(x, y-1) + x
  else
    up := x
end;

begin

  z := up(4, 4)

  { Ausgabe der Variablen Z nicht gefordert }

end.
```

Das PTL-Team wünscht viel Erfolg

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SS98
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Hinweise :

- Versehen Sie die Assembler-Programme der Aufgaben 1 und 2 mit hinreichenden Kommentaren.
- Die Aufgaben 1 und 2 werden gleichgewichtig gewertet.

Aufgabe 1 :

Entwickeln Sie - zuzüglich der Hauptprogramme FIBOASS (8086-Assembler) und FIBOPAS (TurboPascal) als Testumgebung - ein allgemein verwendbares Unterprogramm FIBO (8086-Assembler) zur Berechnung des n-ten Gliedes der Fibonacci-Zahlenfolge.

Für den eigentlichen Berechnungsalgorithmus sind zwei verschiedene Varianten gesucht : Die erste Variante soll auf Rekursion, die zweite Variante dagegen auf Iteration basieren.

Die Schnittstelle des 8086-Assembler-Unterprogramms muß nur die Aufrufmöglichkeit aus 80x86-Assembler- und TurboPascal-Programmen gewährleisten. Gemäß den Konventionen von Borland TurboPascal erfolgt die Parameterübergabe über den Stack und die Ablage lokaler Variablen auf dem Stack.

Im Modul IOINT stehen die Routinen READDEZ und WRITEDEZ zur Ein- und Ausgabe von Integerzahlen zur Verfügung. Der Datentransfer erfolgt jeweils über das DX-Register.

Die Fibonacci-Zahl F_n (d.h. das n-te Glied der Fibonacci-Zahlenfolge) errechnet sich nach folgender Formel :

$$F_n = \begin{cases} 0 \ , \ n=0 \\ 1 \ , \ n=1 \\ F_{n-2}+F_{n-1} \ , \ n \geq 2 \end{cases}$$

Aufgabe 2 :

Übersetzen Sie das nachfolgende Pascal-Programm in ein äquivalentes Assembler-Programm (8086, EXE).

Im Modul DYNMEM stehen die Routinen NEW und DISPOSE zur Anlage und Löschung dynamischer Variablen zur Verfügung. Die Routine WRITEDEZ des Moduls IOINT (vgl. Aufgabe 1) darf weiterhin benutzt werden.

```
Program Aufgabe2;

Type TItemPtr = ^TItem;
     TItem    = Record
                  Value : Integer;
                  Next  : TItemPtr
                End;

Var Head : TItemPtr;
    Item : TItemPtr;
    I    : Integer;

Function ShowAndAdd(Head:TItemPtr):Integer;

Var Item : TItemPtr;
    Sum  : Integer;

Begin
  Sum := 0;
  Item := Head;
  While Item <> Nil Do Begin
    Writeln(Item^.Value);
    Sum := Sum + Item^.Value;
    Item := Item^.Next
  End;
  ShowAndAdd := Sum
End;

Begin
  Head := Nil;
  For I := 1 To 10 Do Begin
    New(Item);
    Item^.Value := I;
    Item^.Next := Head;
    Head := Item;
  End;
  Writeln(ShowAndAdd(Head));
  While Head <> Nil Do Begin
    Item := Head;
    Head := Head^.Next;
    Dispose(Item)
  End
End.
```

Zusatzfrage :
Welche Werte werden durch die Anweisung
Writeln(ShowAndAdd(Head))
ausgegeben ?

Das PTL-Team wünscht viel Erfolg

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SS98
schriftliche Prüfung im Fach Prozeßtechnik
abgelehnter Vorschlag (verwendet für Nachprüfung)

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Entwickeln Sie für den 8086-Prozessor ein Assembler-Programm (NEXTDATE.ASM) zur Implementierung eines allgemein verwendbaren Unterprogramms NEXTDATE, welches für ein gegebenes Datum das Datum des nachfolgenden Tages berechnet. Ziel der Entwicklung ist die Einbindung des Objektcodes (NEXTDATE.OBJ) obigen Assembler-Programms mittels der Compiler-Direktive {$L ...} in ein Borland TurboPascal Hauptprogramm.

Übersetzen Sie zur Komplettierung der Testumgebung - bestehend aus den Hauptprogrammen DATEPAS (TurboPascal) und DATEASS (8086-Assembler) - das TurboPascal-Programm in ein äquivalentes 8086-Assembler-Programm.

Versehen Sie Ihre Assembler-Programme mit hinreichenden Kommentaren (incl. Beschreibung der Algorithmen durch Skizzen und / oder Pseudocodeprogramme).

Die Schnittstelle des 8086-Assembler-Unterprogramms muß nur die Aufrufmöglichkeit aus TurboPascal- und 80x86-Asssembler-Programmen gewährleisten. Gemäß den Konventionen von Borland TurboPascal erfolgt die Parameterübergabe über den Stack und die Ablage lokaler Variablen auf dem Stack.

Auf der zweiten Seite der Aufgabenstellung sind das Hauptprogramm DATEPAS und dessen Ausgabe aufgeführt.

Das PTL-Team wünscht viel Erfolg

```
Program DATEPAS;

Var Dati : Array [1..2] Of String;
    I, J : Integer;

Procedure NextDate(Var Datum:String); External;
{$L NEXTDATE}   { NEXTDATE.OBJ, d.h. Objektcode }

Begin
  Dati[1] := '19981129';   { 29.11.1998 im Format JJJJMMTT }
  Dati[2] := '19981230';   { 30.12.1998 im Format JJJJMMTT }
  For I := 1 To 2 Do Begin
    Writeln(Dati[I]);
    For J := 1 To 3 Do Begin
      NextDate(Dati[I]);
      Writeln(Dati[I])
    End
  End
End.
```

Hauptprogramm DATEPAS

```
19981129
19981130
19981201
19981202
19981230
19981231
19990101
19990102
```

Ausgabe des Hauptprogramms DATEPAS

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS98/99
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Ziel dieser Klausur ist die Implementierung einer (schnellen) mathematischen Toolbox bestehend aus

- Fibonacci-Zahl,
- größter gemeinsamer Teiler (ggT),
- kleinstes gemeinsames Vielfaches (kgV),
- Fakultät und
- Zahlensystemsumwandlung.

Übersetzen Sie daher das nachfolgende (Seiten 3 und 4) Pascal-Programm in zwei äquivalente Assemblerprogramme (8086, EXE) :

- Das Hauptprogramm nach AUFGABE.ASM und
- Die Funktionen (Fibonacci, ggT, kgV, Fakultaet) und die Prozedur (Zahlensysteme) nach TOOLBOX.ASM

Kommentieren Sie die Assemblerprogramme durch eindeutige Zuordnung der Pascal-Befehle zu den Assembler-Befehlen. Ohne derartige Kommentierung kann Ihre Klausur nicht gewertet werden.

Die Schnittstellen der 8086-Assembler-Unterprogramme der Toolbox müssen nur die Aufrufmöglichkeit aus TurboPascal-Programmen (Stichwort: Compiler-Direktive {$L ...}) und 80x86-Assembler-Programmen (Stichwort: EXTRN-Anweisung) gewährleisten. Gemäß den Konventionen von Borland TurboPascal erfolgt die Parameterübergabe über den Stack, die Ablage lokaler Variablen auf dem Stack und die Rückgabe eines Integer-Funktionswerts über das AX-Register.

Im Modul IOINT steht die Routine WRITEDEZ zur Ausgabe von Integer-Zahlen zur Verfügung. Der Datentransfer erfolgt über das DX-Register.

Die Ausgabe des Pascal-Programms möge die Aufgabenstellung verdeutlichen.

Das PTL-Team wünscht viel Erfolg

```
Program Aufgabe;

Var Q,Z : String;
    I   : Integer;
    B   : Boolean;

Function Fibonacci(X:Integer):Integer;
{ Integer statt LongInt; Klausurvereinfachung ! }

Begin
  If (X=0) Or (X=1) Then
    Fibonacci := X
  Else
    Fibonacci := Fibonacci(X-2) + Fibonacci(X-1)
End;

Function ggT(X,Y:Integer):Integer;

Begin
  If X = Y Then
    ggT := X
  Else If X > Y Then
    ggT := ggT(X-Y,Y)
  Else
    ggT := ggT(X,Y-X)
End;

Function kgV(X,Y:Integer):Integer;

Begin
  kgV := X * Y Div ggT(X,Y)
End;

Function Fakultaet(X:Integer):Integer;
{ Integer statt LongInt; Klausurvereinfachung ! }

Begin
  If (X=0) Or (X=1) Then
    Fakultaet := 1
  Else
    Fakultaet := X * Fakultaet(X-1)
End;
```

Pascal-Programm

```
Procedure Zahlensysteme(    Quellbasis : Integer;
                            Zielbasis  : Integer;
                        Var Quelle     : String;
                        Var Ziel       : String;
                        Var Gueltig    : Boolean);

Const Ziffern : String = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';

Var Dezimal    : Integer; { statt LongInt; Klausurvereinfachung ! }
    Wertigkeit : Integer; { statt LongInt; Klausurvereinfachung ! }
    I          : Integer;

Begin
  Gueltig := Length(Quelle) > 0;
  I := 1;
  While Gueltig And (I <= Length(Quelle)) Do
    If Not (Pos(Quelle[I],Ziffern) In [1..Quellbasis]) Then
      Gueltig := False
    Else
      I := I + 1;
  If Gueltig Then Begin
    Dezimal := 0;
    Wertigkeit := 1;
    For I := Length(Quelle) Downto 1 Do Begin
      Dezimal := Dezimal + (Pos(Quelle[I],Ziffern)-1) * Wertigkeit;
      Wertigkeit := Wertigkeit * Quellbasis
    End;
    Ziel := '';
    Repeat
      Ziel := Ziffern[(Dezimal Mod Zielbasis)+1] + Ziel;
      Dezimal := Dezimal Div Zielbasis;
    Until Dezimal = 0;
  End
End;

Begin
  For I := 1 To 10 Do Writeln(I,',',Fibonacci(I));
  For I := 1 To 20 Do Writeln(I,',',20-I+1,',',ggT(I,20-I+1));
  For I := 1 To 20 Do Writeln(I,',',20-I+1,',',kgV(I,20-I+1));
  For I := 1 To 10 Do Writeln(I,',',Fakultaet(I));
  Q := '255';
  For I := 2 To 36 Do Begin
    Zahlensysteme(10,I,Q,Z,B);
    Writeln(Q,',',I,',',Z)
  End
End.
```

Pascal-Programm (Fort.)

```
1,1
2,1
3,2
4,3
5,5
6,8
7,13
8,21
9,34
10,55
1,20,1
2,19,1
3,18,3
4,17,1
5,16,1
6,15,3
7,14,7
8,13,1
9,12,3
10,11,1
11,10,1
12,9,3
13,8,1
14,7,7
15,6,3
16,5,1
17,4,1
18,3,3
19,2,1
20,1,1
1,20,20
2,19,38
3,18,18
4,17,68
5,16,80
6,15,30
7,14,14
8,13,104
9,12,36
10,11,110
11,10,110
12,9,36
13,8,104
14,7,14
15,6,30
16,5,80
17,4,68
18,3,18
19,2,38
20,1,20
1,1
2,2
3,6
4,24
5,120
6,720
7,5040
8,-25216
9,-30336
10,24320
```

Ausgabe des Pascal-Programms

```
255,2,11111111
255,3,100110
255,4,3333
255,5,2010
255,6,1103
255,7,513
255,8,377
255,9,313
255,10,255
255,11,212
255,12,193
255,13,168
255,14,143
255,15,120
255,16,FF
255,17,F0
255,18,E3
255,19,D8
255,20,CF
255,21,C3
255,22,BD
255,23,B2
255,24,AF
255,25,A5
255,26,9L
255,27,9C
255,28,93
255,29,8N
255,30,8F
255,31,87
255,32,7V
255,33,7O
255,34,7H
255,35,7A
255,36,73
```

Ausgabe des Pascal-Programms (Fort.)

Tips & Tricks

- Die Standardfunktion ORD liefert für Argumente ordinalen Datentyps die zugehörige Ordnungszahl

- Funktionsweise der Funktion LENGTH innerhalb der Prozedur ZAHLEN-SYSTEME :

```
Function Length(S:String):Integer;

Begin
  Length := Ord(S[0])
End;
```

- Funktionsweise der Funktion POS innerhalb der Prozedur ZAHLENSYSTEME :

```
Function Pos(Q:Char;Z:String):Integer;

Var I : Integer;

Begin
  Pos := 0;
  I := 1;
  While I <= Length(Z) Do Begin
    If Q = Z[I] Then Begin
      Pos := I;
      I := Length(Z)
    End;
    I := I + 1
  End
End;
```

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SS99
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Entwickeln Sie einen Viewer ("Betrachter") für den Speicherbereich der Großbuchstaben des Hardware-Zeichensatzes durch Reimplementierung des nachfolgenden Pascal-Programms als 80x86-Assembler-Programm.

Das Assembler-Programm soll sich an den Vorgaben des Pascal-Programms orientieren. Dies gilt insbesondere für die durch die Unterprogramme CopyScreen, WriteChar, WriteHexByte und ReadKey geprägte Struktur des Programms. In Anlehnung an die Konventionen von Borland TurboPascal bietet sich die Parameterübergabe über den Stack, die Ablage lokaler Variablen auf dem Stack und die Rückgabe eines Byte-Funktionswerts über das AL-Register an.

Versehen Sie Ihr Assembler-Programm mit hinreichenden Kommentaren, die den Bezug zum ursprünglichen Pascal-Programm verdeutlichen.

Hinweise :

- Die Standardfunktion CHR liefert für Argumente des Datentyps Integer das zugehörige Zeichen des Rechnerzeichensatzes (z.B. ASCII-Code).
- Die Standardfunktion ORD liefert für Argumente ordinalen Datentyps die zugehörige Ordnungszahl.
- Die Standardfunktion PRED liefert für Argumente ordinalen Datentyps das vorherige Element dieses Datentyps (z.B. PRED(´B´) -> ´A´).
- Die Standardfunktion SUCC liefert für Argumente ordinalen Datentyps das nachfolgende Element dieses Datentyps (z.B. SUCC(´B´) -> ´C´).

Anlage :

- ASCII-Code
- Befehlssatz des 8088

Das PTL-Team wünscht viel Erfolg

```
Program Aufgabe;

Uses Crt;

Const PageUp   = #73;
      PageDown = #81;
      Esc      = #27;
      ScanCode = #0;

Type TScreen = Array[1..25,1..80,1..2] Of Byte;

Var CharPixel : Array[Char,1..8] Of Byte Absolute $F000:$FA6E;
    Screen    : TScreen Absolute $B800:$0000;
    _Screen   : TScreen;
    Zeichen   : Char;
    Zeile     : Integer;
    Spalte    : Integer;
    Maske     : Byte;
    Taste     : Char;

Procedure CopyScreen(Var QScreen,ZScreen:TScreen;QInit:Boolean);

Var Zeile,Spalte : Integer;

Begin
  For Zeile := 1 To 25 Do
    For Spalte := 1 To 80 Do Begin
      ZScreen[Zeile,Spalte] := QScreen[Zeile,Spalte];
      If QInit Then Begin
        QScreen[Zeile,Spalte,1] := Ord(' ');
        QScreen[Zeile,Spalte,2] := 252 { red on white }
      End
    End
End;

Procedure WriteChar(Zeile,Spalte:Integer;Zeichen:Char);

Begin
  Screen[Zeile,Spalte,1] := Ord(Zeichen)
End;

Procedure WriteHexByte(Zeile,Spalte:Integer;Zahl:Byte);

Var Zeichen : Char;

Begin
  Zeichen := Chr((Zahl Shr 4)+Ord('0'));
  If Zeichen > '9' Then Zeichen := Chr(Ord(Zeichen)+7);
  WriteChar(Zeile,Spalte,Zeichen);
  Zeichen := Chr((Zahl And $F)+Ord('0'));
  If Zeichen > '9' Then Zeichen := Chr(Ord(Zeichen)+7);
  WriteChar(Zeile,Spalte+1,Zeichen)
End;

Function ReadKey : Char;

Begin
  Asm
    MOV AH,8
    INT 21h
    MOV @Result,AL
  End
End;
```

Pascal-Programm

```
Begin
  CopyScreen(Screen,_Screen,True);
  Zeichen := 'A';
  Repeat
    For Zeile := 1 To 8 Do Begin
      Maske := $80;
      For Spalte := 1 To 8 Do Begin
        If CharPixel[Zeichen,Zeile] And Maske <> 0 Then
          WriteChar(Zeile,Spalte,Zeichen)
        Else
          WriteChar(Zeile,Spalte,' ');
        Maske := Maske Shr 1
      End;
      WriteHexByte(Zeile,11,CharPixel[Zeichen,Zeile])
    End;
    Repeat
      Taste := Readkey
    Until (Taste = Esc) Or (Taste = ScanCode);
    If Taste = Scancode Then
      Case Readkey Of
        PageUp     : If Zeichen > 'A' Then Zeichen := Pred(Zeichen);
        PageDown   : If Zeichen < 'Z' Then Zeichen := Succ(Zeichen);
      End
  Until Taste = Esc;
  CopyScreen(_Screen,Screen,False)
End.
```

Pascal-Programm (Fort.)

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS99/00
schriftliche Prüfung im Fach Prozeßtechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Für das Prozeßmanagement ist eine effektive Listenverwaltung unverzichtbar.

Entwickeln Sie daher für den 8086-Prozessor ein Assemblerprogramm (TOOLBOX.ASM) zur Implementierung einer allgemein verwendbaren "Listen-Toolbox" bestehend aus Unterprogrammen der Funktionalität

- Element erzeugen,
- Element anzeigen,
- Element löschen,
- Element in Liste einfügen,
- Element in Liste suchen,
- Element aus Liste entfernen und
- Liste anzeigen

basierend auf den Werten

- Prozeßnummer (Sortierkriterium),
- "1.Wert" des Prozeß,
- "2.Wert" des Prozeß und
- "3.Wert" des Prozeß.

Die Einbindung des Objektcodes (TOOLBOX.OBJ) obigen Assemblerprogramms mittels der Compiler-Direktive {$L ...} in ein Borland TurboPascal (Haupt-)Programm zeigt das nachfolgende (Seiten 4 und 5) Pascal-Programm.

Versehen Sie Ihr Assemblerprogramm mit hinreichenden Kommentaren (incl. Skizzierung der Algorithmen, insbesondere der Listenoperationen). Ohne derartige Kommentierung kann Ihre Klausur nicht gewertet werden.

Die Schnittstellen der 8086-Assembler-Unterprogramme der Toolbox müssen neben der Aufrufmöglichkeit aus TurboPascal-Programmen (s.o.) nur noch die Aufrufmöglichkeit aus 80x86-Assembler-Programmen (Stichwort: EXTRN-Anweisung) gewährleisten. Gemäß den Konventionen von Borland TurboPascal erfolgt die Parameterübergabe über den Stack, die Ablage lokaler Variablen auf dem Stack und die Rückgabe eines Pointer-Funktionswerts über das DX:AX-Registerpaar.

Hinweis : Die Übersetzung des Pascal-Hauptprogramms in ein Assemblerprogramm ist nicht gefordert.

In den Modulen DYNMEM und IOINT stehen die Routinen NEW und DISPOSE zur Anlage und Löschung dynamischer Variablen sowie WRITEDEZ zur Ausgabe von Integer-Zahlen zur Verfügung :

- NEW
  - In: CX = Größe der dynamischen Variable in Bytes
  - Out: DX:AX = Pointer auf die dynamische Variable
  - Jede Speicheranforderung ist erfüllbar
- DISPOSE
  - In: DX:AX = Pointer auf die dynamische Variable
    CX = Größe der dynamischen Variable in Bytes
  - Out: -
- WRITEDEZ
  - In: DX = Integer-Zahl
  - Out: -

Die Ausgabe des Pascal-Programms möge die Aufgabenstellung verdeutlichen.

Das PTL-Team wünscht viel Erfolg

```
Program Aufgabe;

Type ItemPtr = ^Items;
     Items   = Record
                 Number : Integer;
                 Next   : ItemPtr;
                 ValueA : Integer;
                 ValueB : Integer;
                 ValueC : Integer;
               End;

Var TopOfList : ItemPtr;
    Item      : ItemPtr;
    I         : Integer;

{$L TOOLBOX}

Function CreateItem(NewNumber : Integer;
                    NewValueA : Integer;
                    NewValueB : Integer;
                    NewValueC : Integer):
                   {Result}     ItemPtr;
                    External;

Procedure ShowItem(Item : ItemPtr); External;

Procedure DeleteItem(Item : ItemPtr); External;

Procedure InsertItemInList(Var TopOfList : ItemPtr;
                               Item      : ItemPtr);
                               External;

Function SearchItemInList(ItemOfList   : ItemPtr;
                          SearchNumber : Integer):
                         {Result}        ItemPtr;
                          External;

Function RemoveItemFromList(Var TopOfList    : Itemptr;
                                RemoveNumber : Integer):
                               {Result}        ItemPtr;
                                External;

Procedure ShowList(ItemOfList : ItemPtr); External;
```

Pascal-Programm

```
Begin
  TopOfList := Nil;
  For I := 1 To 10 Do Begin
    Item := CreateItem(I,I+1,I+2,I+3);
    InsertItemInList(TopOfList,Item)
  End;
  ShowList(TopOfList);
  For I := 1 To 10 Do Begin
    Item := RemoveItemFromList(TopOfList,I);
    DeleteItem(Item)
  End;
  ShowList(TopOfList);
  Item := CreateItem(2,0,0,0);
  InsertItemInList(TopOfList,Item);
  ShowList(TopOfList);
  Item := CreateItem(1,0,0,0);
  InsertItemInList(TopOfList,Item);
  ShowList(TopOfList);
  Item := CreateItem(4,0,0,0);
  InsertItemInList(TopOfList,Item);
  ShowList(TopOfList);
  Item := CreateItem(3,0,0,0);
  InsertItemInList(TopOfList,Item);
  ShowList(TopOfList);
  Writeln('--- ShowItem ---');
  ShowItem(SearchItemInList(TopOfList,1));
  ShowItem(SearchItemInList(TopOfList,2));
  ShowItem(SearchItemInList(TopOfList,3));
  ShowItem(SearchItemInList(TopOfList,4));
  Item := RemoveItemFromList(TopOfList,3);
  DeleteItem(Item);
  ShowList(TopOfList);
  Item := RemoveItemFromList(TopOfList,4);
  DeleteItem(Item);
  ShowList(TopOfList);
  Item := RemoveItemFromList(TopOfList,1);
  DeleteItem(Item);
  ShowList(TopOfList);
  Item := RemoveItemFromList(TopOfList,2);
  DeleteItem(Item);
  ShowList(TopOfList)
End.
```

Pascal-Programm (Fort.)

```
--- ShowList ---
1,2,3,4
2,3,4,5
3,4,5,6
4,5,6,7
5,6,7,8
6,7,8,9
7,8,9,10
8,9,10,11
9,10,11,12
10,11,12,13
--- ShowList ---
--- ShowList ---
2,0,0,0
--- ShowList ---
1,0,0,0
2,0,0,0
--- ShowList ---
1,0,0,0
2,0,0,0
4,0,0,0
--- ShowList ---
1,0,0,0
2,0,0,0
3,0,0,0
4,0,0,0
--- ShowItem ---
1,0,0,0
2,0,0,0
3,0,0,0
4,0,0,0
--- ShowList ---
1,0,0,0
2,0,0,0
4,0,0,0
--- ShowList ---
1,0,0,0
2,0,0,0
--- ShowList ---
2,0,0,0
--- ShowList ---
```

Ausgabe des Pascal-Programms

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SS00
schriftliche Prüfung im Fach Prozesstechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Ein Hauptanwendungsgebiet der Assemblersprache ist die Einbindung von Assembler-Unterprogrammen für zeitkritische Probleme in Hochsprachen-Hauptprogramme. Entwickeln Sie daher - mit dem Ziel der Einbindung in Borland TurboPascal-Programme - folgende zwei Assembler-Unterprogramme :

- Addition maximal 254 stelliger natürlicher Zahlen
- Multiplikation maximal 127 stelliger natürlicher Zahlen

Für die Speicherung der natürlichen Zahlen ist der Datentyp *String* vorgesehen. Entwickeln Sie daher für die Stringverarbeitung - ebenfalls mit dem Ziel der Einbindung in Borland TurboPascal-Programme - zusätzlich folgende drei Assembler-Unterprogramme :

- Hinzufügen einer bestimmten Anzahl führender Nullen
- Hinzufügen einer bestimmten Anzahl nachfolgender Nullen
- Testen auf ausschließlich Zifferninhalt

Die Algorithmen der fünf Unterprogramme sind bereits als Pascal-Code vorgegeben. Falls Ihre Codierung in der 8086-Assemblersprache wesentliche Änderungen aufweist, ist eine zusätzliche Pseudocode-Dokumentation notwendig. Für das beispielhafte Pascal-Hauptprogramm ist keine Assembler-Übersetzung gefordert. Vergessen Sie nicht die hinreichende Kommentierung Ihrer Assembler-Unterprogramme z.B. durch Zuordnung der Pascal-Befehle zu den Assembler-Befehlen.

Hinweise zu Borland TurboPascal :

- Die Technik der Übergabe von Zeichenketten an Unterprogramme ist sowohl für Referenz- als auch Wertparameter ebenso wie für Funktionswerte identisch, d.h. die Adresse des Speicherbereichs der Zeichenkette wird über den Stack an das Unterprogramm übergeben.
- Die Standardfunktion ORD liefert für Argumente ordinalen Datentyps die zugehörige Ordnungszahl.
- Die Standardfunktion CHR liefert für Argumente des Datentyps Integer das zugehörige Zeichen des Rechnerzeichensatzes (z.B. ASCII-Code).

Das PTL-Team wünscht viel Erfolg

```
Function XAdd(S1,S2:String):String;

Var Se    : String;
    I,U,Z : Integer;

Begin
  Se := '';
  U := 0;
  FuehrendeNullen(S1,Ord(S2[0])-Ord(S1[0]));
  FuehrendeNullen(S2,Ord(S1[0])-Ord(S2[0]));
  For I := Ord(S1[0]) DownTo 1 Do Begin
    Z := Ord(S1[I])+Ord(S2[I])-2*Ord('0')+U;
    Se := Chr(Z Mod 10 + Ord('0'))+Se;
    U := Z Div 10
  End;
  If U > 0 Then
    Se := Chr(U+Ord('0'))+Se;
  XAdd := Se
End;
```

Addition maximal 254 stelliger natürlicher Zahlen

```
Function XMult(S1,S2:String):String;

Var Se,Sm   : String;
    I,J,U,Z : Integer;

Begin
  Se := '0';
  For J := Ord(S2[0]) DownTo 1 Do Begin
    Sm := '';
    U := 0;
    For I := Ord(S1[0]) DownTo 1 Do Begin
      Z := (Ord(S2[J])-Ord('0'))*(Ord(S1[I])-Ord('0'))+U;
      Sm := Chr(Z Mod 10 + Ord('0'))+Sm;
      U := Z Div 10
    End;
    If U > 0 Then
      Sm := Chr(U+Ord('0'))+Sm;
    NachfolgendeNullen(Sm,Ord(S2[0])-J);
    Se := XAdd(Se,Sm)
  End;
  XMult := Se
End;
```

Multiplikation maximal 127 stelliger natürlicher Zahlen

```
Procedure FuehrendeNullen(Var S:String;A:Integer);

Var I : Integer;

Begin
  For I := 1 To A Do S := '0'+S
End;
```

Hinzufügen einer bestimmten Anzahl führender Nullen

```
Procedure NachfolgendeNullen(Var S:String;A:Integer);

Var I : Integer;

Begin
  For I := 1 To A Do S := S+'0'
End;
```

Hinzufügen einer bestimmten Anzahl nachfolgender Nullen

```
Function NurZiffern(S:String):Boolean;

Var I : Integer;

Begin
  NurZiffern := Ord(S[0]) > 0;
  For I := 1 To Ord(S[0]) Do
    If Not (S[I] In ['0'..'9']) Then
      NurZiffern := False
End;
```

Testen auf ausschließlich Zifferninhalt

```
Program Aufgabe;

Var S1,S2 : String;

Procedure FuehrendeNullen(Var S:String;A:Integer); External;
{$L FNULLEN}   { FNULLEN.OBJ, d.h. Objektcode }

Procedure NachfolgendeNullen(Var S:String;A:Integer); External;
{$L NNULLEN}   { NNULLEN.OBJ, d.h. Objektcode }

Function XAdd(S1,S2:String):String; External;
{$L XADD}   { XADD.OBJ, d.h. Objektcode }

Function XMult(S1,S2:String):String; External;
{$L XMULT}   { XMULT.OBJ, d.h. Objektcode }

Function NurZiffern(S:String):Boolean; External;
{$L ZIFFERN}   { ZIFFERN.OBJ, d.h. Objektcode }

Begin
  Repeat
    Write('Gib 1.Zahl     : '); ReadLn(S1)
  Until NurZiffern(S1);
  Repeat
    Write('Gib 2.Zahl     : '); ReadLn(S2)
  Until NurZiffern(S2);
  Writeln('Addition       : ',XAdd(S1,S2));
  Writeln('Multiplikation : ',XMult(S1,S2))
End.
```

Pascal-Hauptprogramm (Beispiel)

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS00/01
schriftliche Prüfung im Fach Prozesstechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Entwickeln Sie für einen zum 8086-Prozessor kompatiblen Mikrokontroller das Ratespiel *MasterMind* durch Übersetzung des nachfolgenden Pascal-Programms in ein 8086-Assembler-Programm.

Zur Kommunikation mit dem Spieler bzw. der Spielerin stehen an Peripherie zur Verfügung :

- ein fünfstelliges numerisches LED-Display

```
Procedure DisplayDigit(Stelle:Byte;Digit:Char);
```

- fünf jeweils in den Farben grün, gelb oder rot leuchtende LED´s und

```
Procedure DisplayLED(Stelle:Byte;Farbe:TFarbe);
```

- eine numerische Zehnertastatur

```
Procedure GetDigit(Var Digit:Char);
```

Nicht Gegenstand der Klausur ist der im Pascal-Programm enthaltene Code zur Simulation der Hardware auf der das übersetzte Assembler-Programm ablaufen soll, d.h. alle Programmteile, die nur bei der Definition des Symbols SIMULATION vom TurboPascal-Compiler berücksichtigt werden, sollen nicht nach Assembler übersetzt werden.

Vergessen Sie nicht die hinreichende Kommentierung Ihres Assembler-Programms z.B. durch Zuordnung der Pascal-Befehle zu den Assembler-Befehlen.

Hinweise zu Borland TurboPascal :

- Die Standardfunktion ORD liefert für Argumente ordinalen Datentyps die zugehörige Ordnungszahl.
- Die Standardfunktion CHR liefert für Argumente des Datentyps Integer das zugehörige Zeichen des Rechnerzeichensatzes (z.B. IBM-ASCII-Code).

Das PTL-Team wünscht viel Erfolg

```
{$DEFINE SIMULATION}
Program Aufgabe;

{$IFDEF SIMULATION}
Uses Crt;
{$ENDIF}

Type TFarbe = (Gruen,Gelb,Rot);

Const ZufallsZifferInit : Byte = 1;

Var Rechner : String[5];
    Spieler : String[5];
    Stelle  : Byte;
    Zeichen : Char;

Function SwapTetrade(X:Byte):Byte;

Var High,Low : Byte;

Begin
  High := X And $F0 Shr 4;
  Low  := X And $0F;
  SwapTetrade := Low Shl 4 + High
End;

Function ZufallsZiffer:Char;

Begin
  ZufallsZifferInit :=
    ZufallsZifferInit Shl 1 + ( (ZufallsZifferInit Shr 7) Xor
                                ((ZufallsZifferInit Shr 6) And $01) Xor
                                ((ZufallsZifferInit Shr 4) And $01) Xor
                                ((ZufallsZifferInit Shr 2) And $01)
                              );
  ZufallsZiffer := Chr(ZufallsZifferInit Mod 10 + Ord('0'))
End;

Function FoundChar(C:Char;S:String):Boolean;

Var Stelle : Byte;
    Result : Boolean;

Begin
  Result := False;
  For Stelle := 1 To Ord(S[0]) Do
    If S[Stelle] = C Then
      Result := True;
  FoundChar := Result
End;

{$IFDEF SIMULATION}
Procedure CursorOff;

Begin
  Asm
    MOV AH,01h
    MOV CH,3Fh
    MOV CL,1Fh
    INT 10h
  End
End;
{$ENDIF}
```

```
{$IFDEF SIMULATION}
Procedure DisplayLayout;

Begin
  Writeln('╔═══╤═══╤═══╤═══╤═══╗');
  Writeln('│   │   │   │   │   │');
  Writeln('├───┼───┼───┼───┼───┤');
  Writeln('│   │   │   │   │   │');
  Writeln('├───┼───┼───┼───┼───┤');
  Writeln('│   │ 7 │ 8 │ 9 │   │');
  Writeln('├───┼───┼───┼───┼───┤');
  Writeln('│   │ 4 │ 5 │ 6 │   │');
  Writeln('├───┼───┼───┼───┼───┤');
  Writeln('│   │ 1 │ 2 │ 3 │   │');
  Writeln('├───┼───┼───┼───┼───┤');
  Writeln('│   │   │ 0 │   │   │');
  Writeln('╚═══╧═══╧═══╧═══╧═══╝')
End;
{$ENDIF}

{$IFDEF SIMULATION}
Procedure ConfirmDigit(Digit:Char);

Type TPosition = Array['0'..'9'] Of Record X,Y:Byte End;

Const Position : TPosition =
  ((X:6;Y:12),(X:4;Y:10),(X:6;Y:10),(X:8;Y:10),(X:4;Y:8),
   (X:6;Y:8),(X:8;Y:8),(X:4;Y:6),(X:6;Y:6),(X:8;Y:6));

Var X,Y:Byte;

Begin
  X := Position[Digit].X;
  Y := Position[Digit].Y;
  GotoXY(X,Y);
  TextAttr := SwapTetrade(TextAttr);
  Write(Digit);
  Delay(1000);
  GotoXY(X,Y);
  TextAttr := SwapTetrade(TextAttr);
  Write(Digit)
End;
{$ENDIF}

Procedure DisplayDigit(Stelle:Byte;Digit:Char);

{$IFDEF SIMULATION}
Type TPosition = Array[1..5] Of Record X,Y:Byte End;

Const Position : TPosition =
  ((X:2;Y:2),(X:4;Y:2),(X:6;Y:2),(X:8;Y:2),(X:10;Y:2));

Var X,Y:Byte;

Begin
  X := Position[Stelle].X;
  Y := Position[Stelle].Y;
  GotoXY(X,Y);
  Write(Digit)
End;
{$ENDIF}

{$IFNDEF SIMULATION}
Begin
  { Code der Hardwareversion : Kein Bestandteil der Klausur :-) }
End;
{$ENDIF}
```

```
Procedure DisplayLED(Stelle:Byte;Farbe:TFarbe);

{$IFDEF SIMULATION}
Type TPosition = Array[1..5] Of Record X,Y:Byte End;

Const Position : TPosition =
  ((X:2;Y:4),(X:4;Y:4),(X:6;Y:4),(X:8;Y:4),(X:10;Y:4));

Var X,Y         : Byte;
    OldTextAttr : Byte;

Begin
  OldTextAttr := TextAttr;
  Case Farbe Of
    Gruen : TextColor(Green);
    Gelb  : TextColor(Yellow);
    Rot   : TextColor(Red)
  End;
  X := Position[Stelle].X;
  Y := Position[Stelle].Y;
  GotoXY(X,Y);
  Write('');
  TextAttr := OldTextAttr
End;
{$ENDIF}

{$IFNDEF SIMULATION}
Begin
  { Code der Hardwareversion : Kein Bestandteil der Klausur :-) }
End;
{$ENDIF}

Procedure GetDigit(Var Digit:Char);

{$IFDEF SIMULATION}
Begin
  Repeat
    Digit := ReadKey
  Until Digit In ['0'..'9',#27]
End;
{$ENDIF}

{$IFNDEF SIMULATION}
Begin
  { Code der Hardwareversion : Kein Bestandteil der Klausur :-) }
End;
{$ENDIF}
```

```
Begin
{$IFDEF SIMULATION}
  ClrScr;
  CursorOff;
  DisplayLayout;
{$ENDIF}
  Repeat
    Rechner := '';
    For Stelle := 1 To 5 Do
      Rechner := Rechner+Zufallsziffer;
    Repeat
      Spieler := '';
      For Stelle := 1 To 5 Do Begin
        GetDigit(Zeichen);
{$IFDEF SIMULATION}
        If Zeichen = #27 Then Halt;
{$ENDIF}
        Spieler := Spieler+Zeichen;
        DisplayDigit(Stelle,Zeichen);
{$IFDEF SIMULATION}
        ConfirmDigit(Zeichen)
{$ENDIF}
      End;
      For Stelle := 1 To 5 Do Begin
        If Rechner[Stelle] = Spieler[Stelle] Then
          DisplayLED(Stelle,Gruen)
        Else If FoundChar(Spieler[Stelle],Rechner) Then
               DisplayLED(Stelle,Gelb)
             Else
               DisplayLED(Stelle,Rot)
      End
    Until Rechner = Spieler
  Until False
End.
```