Einfache Übungen zum Assembler und zum Simulator

Hinweis. Alle genannten Dateien finden Sie im Unterverzeichnis „Programs“ des Stammverzeichnisses.

Aufgabe 1:

In dieser Aufgabe soll die Arbeit mit dem Assembler und dem Terminalprogramm geübt werden. In der Vorlagedatei 1_VORL.ASM finden Sie ein Programm, welches das folgende Problem löst. Die Datei enthält jedoch Assemblerfehler wie auch logische Fehler. Ihre Aufgabe ist es, das Programm so zu ändern, daß es fehlerfrei assembliert wird, und dazu ein paar Fragen zu beantworten.

Einige Fehler werden Ihnen vielleicht sofort ins Auge fallen. „Selbstgemachte“ Fehler haben jedoch vor allem die Eigenschaft, nicht sofort aufzufallen. Daher sollen Sie die Fehler anhand der Assemblerausgaben finden und berichtigen.

Das Problem: *Schreiben Sie ein Programm, welches in der Anzeige eine Uhr simuliert. Dabei sollen an den Stellen S5..S4 der Siebensegmentanzeige die Stunden angezeigt werden und an den Stellen S3..S2 die Minuten. Die Weiterschaltung der Uhr soll im Sekundentakt erfolgen, d.h. eine Minute wird auf eine Sekunde verkürzt. Solange die Taste „+“ gedrückt wird, soll das Weiterschalten ohne Zeitverzögerung, also so schnell wie möglich geschehen. Alle anderen Tasten sollen keine Bedeutung haben.*

Diese fehlerhafte Lösung finden Sie in der Datei 1_VORL.ASM:

```
*********************************************************************
*         fehlerhaftes Programm folgt
*********************************************************************
          ORG  $0400    ;Beginn des Programmbereichs

          JSR  CLRDSP   ;Anzeige löschen
          LDY  #3E8     ;Y mit 1 Sek. Pause laden
          CLR  STUNDE   ;Stundenzähler auf Null setzen
          CLR  MINUTE   ;Minutenzähler auf Null setzen

ANZEIGE   LDB  STUNDEN  ;Stundenzähler laden...
          LDX  #4       ;...und an S5-S4...
          JSR  SHOWB7SG;...ausgeben
          LDB  MINUTEN  ;Minutenzähler laden...
          LDX  #2       ;...und an S3-S2...
          JSR  SHOWB7SG;...ausgeben

INCMIN    TFR  B,A      ;Minuten nach A kopieren,
          INC           ;inkrementieren,
          DAA           ;und BCD-Korrektur.
          CMP  A,#$60   ;60 Minuten erreicht?
          BEQ  ERHÖHE   ;ja, Stunde erhöhen
          STA  MINUTEN  ;nein, also Minuten speichern...
          BRA  TASTEN   ;...und zur Tastaturabfrage

ERHÖHE    CLR  MINUTEN ;Minutenzähler auf Null
          LDA  STUNDEN  ;Stundenzähler laden,
          INCA          ;inkrementieren,
```

```
          DAA          ;und BCD-Justage
          STA  STUNDEN ;Stunden speichern
          CMPA #24     ;24 Stunden erreicht?
          BNZ  TASTE   ;nein, also zur Tastaturabfrage
          CLR  STUNDEN ;ja, also auf Null setzen

TASTE     JSR  KEY     ;ist eine Taste gedrückt?
          CMPB #$80    ;ist die Taste "+" gedrückt?
          BEQ  ANZEIGE ;ja, sofort weiter mit dem Hochzählen
*
*         Nun Tasten <> "+" abarbeiten, also 1 Sekunde Pause
*
JSR       DLY1MS       ;bei anderen oder keiner Taste eine Sekunde
BRA       ANZEIGE      ; warten und dann zurück
*
*         Daten
*
MINUTEN   RMB  1       ;Speicher für Minutenzähler
STUNDEN   RMB  2       ;Speicher für Stundenzähler

U_END     ORG  *       ;Ende des Programmbereichs
```

a)
Assemblieren Sie das fehlerhafte Programm. Beantworten Sie folgende Fragen:

- Wie lauten die Fehlerausgaben des Assemblers?
- Was bedeuten die angezeigten Fehler?
- Wie lauten die richtigen Befehle?

Berichtigen Sie die Fehler.

b)
Assemblieren Sie nun diese berichtigte Datei. Wenn Sie alle Fehler aus Teil a) berichtigt haben, erscheint nun als Ausgabe ein Listing, in dem weitere Fehler angezeigt werden. (Falls Sie wie in Teil a) nur eine kurze Liste der fehlerhaften Zeilen ohne Listing bekommen, müssen Sie sich weiter mit diesen Fehlern auseinandersetzen, bis das Listing erscheint.) Beantworten Sie folgende Frage zum Listing:

- Welche drei Fehlerarten werden angezeigt?

Die Fehlerart *Symbol Undefined* bedeutet, daß ein Symbolname falsch angegeben wurde. Beantworten Sie folgende Frage:

- Wie lauten die jeweils korrekten Symbolnamen, bzw. warum werden die Symbolnamen als falsch gewertet?

Berichtigen Sie die Fehler! Assemblieren Sie die Datei. Falls alle Symbolnamen korrekt sind, werden Sie ein Listing ohne Fehlermeldungen erhalten. Beantworten Sie folgende Fragen:

- Warum erscheinen die beiden anderen Fehlerarten nicht mehr? (Vergleichen Sie das Listing 1A.LST mit dem Listing 1B.LST !)

- Wie kann ein *Phasing Error* vermieden werden, falls eine ähnliche Situation gewünscht ist?

c)
Laden Sie nun die in Teil b) erzeugte Hex.-Datei (1B.HEX) in den Simulator zum Praktikumsrechner! Lassen Sie das Programm ablaufen, testen Sie auch die Funktion der Taste „+“. Beantworten Sie folgende Frage:

- Welche zwei Fehlfunktionen zeigt das Programm? (Beachten Sie nicht die Funktion der anderen Tasten außer der Taste „+“; dazu kommen wir in Teil d).)

Beide Fehler werden vom Assembler nicht erkannt. Der eine ist eher ein logischer Fehler, der andere eher eine Eigenart des Assemblers. Beantworten Sie nach gründlicher Durchsicht des Listings folgende Fragen:

- Wie lauten die Fehlerkorrekturen?
- Was sagen Sie zur Zeile 47: „STUNDE RMB 2“?

Speichern Sie bitte das korrigierte Programm ab.

d)
Als Programmerweiterung kann eingebaut werden, daß bei Betätigen der Taste „S“ das Programm stoppt.

Dazu fügen Sie nach dem Abschnitt

```
TASTE     JSR  KEY      ;ist eine Taste gedrückt?
          CMPB #$80     ;ist die "+"-Taste gedrückt?
          BEQ  ANZEIGE  ;ja, sofort weiter mit dem Hochzählen
```

die folgenden Zeilen ein:

```
          CMPB #$86     ;ist die "S"-Taste gedrückt?
          BEQ  ENDE     ;ja, zum Programmende
```

Vor der Datendefinition fügen Sie ein:

```
ENDE      SWI           ;Programmende
```

Speichern Sie die geänderte Datei unter einem anderen Namen ab. Assemblieren Sie das Programm und laden Sie die Hex-Datei in den Simulator zum Praktikumsrechner. Beantworten Sie folgende Fragen:

- Welches Fehlverhalten zeigt das Programm? Falls das Programm richtig zu laufen scheint, halten Sie bitte die Taste „G“ einige Sekunden gedrückt, was eigentlich ja keine Wirkung zeigen sollte.
- Was ist die Ursache dieses Fehlers?
- Wie können Sie ihn beheben?

Aufgabe 2:

Hinweis: In dieser Aufgabe soll der Umgang mit dem Simulator geübt werden. In der Datei 2_VORL.ASM erhalten Sie wieder eine fehlerhafte Lösung zum folgenden Problem. Sie sollen anhand dieses Problems den Simulator ein wenig kennenlernen.

Problem: *Schreiben Sie ein Programm, welches eine vierstellige Hexadezimalzahl einliest und in S5..S2 anzeigt. Durch Drücken der Tasten „+“ bzw. „-“ ist dann wahlweise rechts bzw. links von der Zahl die Anzahl der eingeschalteten Segmente dieser Zahl dezimal bzw. hexadezimal anzuzeigen. (Taste „+“ zeigt also in S1,..,S0 die Anzahl dezimal an, Taste „-“ in S6,..,S7 hexadezimal.)*

Fehlerhafte Lösungsdatei 2_VORL.ASM:

```
************************************************************************
*         fehlerhafte Lösung
************************************************************************
          ORG  $0400    ;Beginn des Programmbereichs

EINGABE   JSR  CLRDISP  ;Anzeige löschen
          CLR  ANZHEX   ;Anzahl im Hex.-Format löschen
          CLR  ANZDEZ   ;Anzahl im Dez.-Format löschen
          LDY  #TABELLE;Y mit Zeiger auf Segmentanzahlen laden
          LDX  #5       ;X mit Ausgabeposition laden
          JSR  ZIFFER   ;erste Ziffer einlesen
          JSR  ZIFFER   ;zweite Ziffer einlesen
          JSR  ZIFFER   ;dritte Ziffer einlesen
          JSR  ZIFFER   ;vierte Ziffer einlesen
*
*         Nun werden die beiden Zähler direkt in den Siebensegmentcode
*         gewandelt, um gleich einfacher ausgegeben werden zu können.
*
          LDB  ANZHEX   ;hexadezimale Anzahl laden...
          JSR  B7SG     ;...codieren...
          STD  ANZHEX   ;...und wieder speichern
          LDB  ANZDEZ   ;dezimale Anzahl laden...
          JSR  B7SG     ;...codieren...
          STD  ANZDEZ   ;...und wieder speichern
*
*         Nun die Tastaturabfrage und Auswertung der Tasten
*
TASTE     JSR  HALTKEY  ;Auf Tastendruck warten
          CMPB #$87     ;Taste "L" gedrückt?
          BEQ  EINGABE  ;ja, zur Eingabe zurück
          CMPB #$80     ;Taste "+" gedrückt?
          BEQ  PLUS     ;ja, zur Auswertung
          CMPB #$81     ;Taste "-" gedrückt?
          BNE  TASTE    ;nein, zurück zur Tasteneingabe
*
*         Tasten "+" und "-" auswerten
*
          LDD  ANZHEX   ;D mit codierter Anzahl in hexadezimal laden
          LDY  #0       ;Y mit "codierter Leere" laden
```

```
          BRA  AUSGABE ;zur Ausgabe
PLUS      LDD  #0      ;D mit "codierter Leere" laden
          LDY  ANZDEZ  ;Y mit codierter Anzahl in dezimal laden
AUSGABE   LDX  #6      ;an Stelle S6...
          JSR  SHOWD   ;...wird D ausgegeben (leer oder hex.)
          TFR  Y,D     ;Y wird nach D kopiert...
          LDX  #0      ;...und an Stelle S0...
          JSR  SHOWD   ;...ausgegeben (leer oder dezimal)
          BRA  TASTE   ;Zurück zur Tasteneingabe
*
*         Unterprogramm ZIFFER
*         Eingabe: Ausgabeposition in X; Tabellenzeiger in Y
*         Ausgabe: -
*         Ändert: X dekrementieren; ANZDEZ, ANZHEX entsprechend
*         hochzählen; A und B zerstört
*         Die Routine wartet auf eine Hex.-Ziffer und gibt diese an
*         Position X aus. Die Anzahl der Segmente dieser Ziffer
*         werden den zwei Speicherstellen dazuaddiert.
*
ZIFFER    JSR  HALTKEY ;Auf Tastendruck warten
          BMI  ZIFFER  ;Bit 7 gesetzt -> Funktionstaste -> ignorieren
          JSR  SHOWT7SG;Taste in Siebensegmentcode wandeln und ausgeben
          LDA  ANZHEX  ;Hex.-Zähler laden...
          ADDA B,Y     ;...Segmentanzahl addieren...
          STA  ANZHEX  ;...und speichern
          LDA  ANZDEZ  ;Dezimal-Zähler laden...
          ADDA B,Y     ;...Segmentzahl addieren...
          DAA          ;...BCD-Korrektur...
          STA  ANZDEZ  ;...und speichern
          TFR  X,D     ;Ausgabeposition nach D kopieren...
          DECB         ;...dekrementieren...
          TFR  D,X     ;...und wieder nach X kopieren
          RTS          ;fertig
*
*         Datenbereich
*
ANZHEX    RMB  1       ;hexadezimaler Segmentzähler
ANZDEZ    RMB  1       ;dezimaler Segmentzähler

*              0 1 2 3 4 5 6 7 8 9 A B C D E F
TABELLE   FCB  6,2,5,5,4,5,6,3,7,5,6,5,3,5,5,4

U_END     ORG  *       ;Ende des Programmbereichs
```

a)
Assemblieren Sie das Programm, laden Sie es in den Simulator und starten Sie es. Worin zeigt sich das Fehlverhalten des Programms?

b)
Schalten Sie das Registerfenster zur Beobachtung der Registerwerte ein. Lassen Sie das Programm nochmals ablaufen. Was beobachten Sie im Registerfenster?

c)
Schalten Sie nun ein Speicherfenster zur Anzeige der beiden Zähler ein und lassen Sie dann das Programm laufen. Beantworten Sie folgende Fragen:

- Welchen Speicherbereich wählen Sie zur Anzeige? Auf welche Speicherstellen achten Sie besonders?
- Vergleichen Sie den Speicherauszug mit dem Listing. Was fällt Ihnen auf?
- Wie verhalten sich die wichtigen Speicherstellen bei der Eingabe der Ziffern „A“ und „8“ als erste und zweite Ziffer?
- Was passiert, wenn Sie als dritte Ziffer „0“ eingeben?
    - Was passiert nach Eingabe der vierten Ziffer?

d)
Laden Sie das Programm erneut in den Simulator und wiederholen Sie Teil c). Beantworten Sie folgende Fragen:

- Was ändert sich im Verhalten des Programms?
- Worin liegt also der Programmfehler und wie berichtigen Sie ihn?

e)
Nun sollen Sie kurz den integrierten Disassembler kennenlernen. Versuchen Sie, die Monitorroutinen zu disassemblieren. Öffnen Sie dazu ein Disassemblerfenster und lassen Sie sich den Speicherbereich ab $F100 anzeigen. Beantworten Sie folgende Fragen:

- Was wird angezeigt? Wie sind die Monitorroutinen realisiert?
- Was haben die seltsamen Befehle bei Adresse $F10C, $F10F und $F112 zu bedeuten?

Disassemblieren Sie den Bereich ab $F200.

- Was für eine Routine wird angezeigt?
- Beschreiben Sie kurz die Funktionsweise dieser Routine!

Aufgabe 3:

a) (Vorlagedatei: 3A_VORL.ASM)
Schreiben Sie ein Programm, welches eine Codewandlung zwischen einigen ASCII-Zeichen und Hexadezimal-Ziffern ermöglicht. Das Programm soll ein Byte einlesen und dieses nach Betätigen der Taste „+“ von hexadezimal nach ASCII und nach Betätigen der Taste „-“ von ASCII nach hexadezimal wandeln. Danach soll das Programm von vorne beginnen.

Im einzelnen:
1) Verwenden Sie zur Dateneingabe die Routine in der Vorlagedatei enthaltene Routine).

2) Dann wird die Tastatur abgefragt:
   - Taste „+“: Wandlung von Hex nach ASCII, weiter mit 3)
   - Taste „-“: Wandlung von ASCII nach Hex, weiter mit 3)
   - andere Tasten: weiter mit 1)
3) In 1) eingegebenes Datum in Qülltabelle suchen. Falls gefunden, weiter mit 4). Falls nicht gefunden, weiter mit 1).
4) Gefundenes Datum der Zieltabelle im Operationsfeld ausgeben, weiter mit 1) (ohne Löschen der Anzeige).

Datei 3A_VORL.ASM:

```
          ORG  $0400    ;Beginn des Programmbereichs
*
ANFANG    JSR  CLRDISP  ;Anzeige löschen
EINGABE   JSR  HALTKEY  ;Zeichen von Tastatur lesen
          ASLB          ;viermal shift links, um das eingegebene Zeichen
          ASLB          ;in das obere Halbbyte von B zu übertragen.
          ASLB          ; Falls Funktionstaste gedrückt wurde, bleibt
          ASLB          ; nur der "Ziffernteil" im unteren Halbbyte über.
          STB  DATA     ;Eingabe sichern
          JSR  HALTKEY  ;zweites Zeichen von Tastatur lesen
          ANDB #$0F     ;oberes Halbbyte löschen (falls Funktionstaste)
          ORB  DATA     ;mit erstem Zeichen verknüpfen
          LDX  #0       ;Anzeigeposition
          JSR  SHOWB7SG;B im Siebensegmentcode ausgeben
*
*         nun beginnt das neu Programm

*
*
*         Datenbereich
*
DATA      RMB  1        ;Datenbereich zur Zwischenspeicherung der Eingabe

HEX       FCB  $00      ;hier beginnt die Hex.-Tabelle. Alle Werte werden
          FCB  $01      ;einfach einzeln als Byte-Datum definiert.
          FCB  $02      ;
          FCB  $03      ;
          FCB  $04      ;
          FCB  $05      ;
          FCB  $06      ;
          FCB  $07      ;
          FCB  $08      ;
          FCB  $09      ;
          FCB  $0A      ;
          FCB  $0B      ;
          FCB  $0C      ;
          FCB  $0D      ;
          FCB  $0E      ;
HEXEND    FCB  $0F      ;Ein Hilfs-Label zur Längenberechnung

ASCII     FDB  $3031    ;die zweite Tabelle wird ähnlich definiert,
          FDB  $3233    ;jedoch mit dem Befehl FDB statt FCB.
          FDB  $3435    ;
```

```
          FDB  $3637    ;
          FDB  $3839    ;
          FDB  $4142    ;
          FDB  $4344    ;
ASCIIE    FDB  $4546    ;ASCIIEND als Label ist leider zu lang

TABLEN    EQU  15       ;15 ist maximaler Offset auf Tabellenanfang

U_END     ORG  *        ;Ende des Programmbereichs
```

b)
Die Zeile „TABLEN EQU 15“ definiert TABLEN als Konstante. Bei jeder Änderung der Tabellenlänge müßten Sie daran denken, diese Konstante entsprechend zu ändern. Um das zu umgehen, kann ein Assembler Konstanten berechnen, z.B. aus vorgegebenen Marken. Geben Sie eine Berechnung für TABLEN aus den Marken an, die in der Vorgabedatei angegeben sind (HEX, HEXEND, ASCII, ASCIIE). Tragen Sie Ihre Lösung bitte statt der Zeile „TABLEN EQU 15“ in die Lösungsdatei zu Teilaufgabe a) ein.

c)
In der Vorlagedatei 3A_VORL.ASM ist die Tabelle der ASCII-Zeichen (ab Marke ASCII) mit dem Pseudobefehl FDB statt FCB definiert. Wieso funktioniert das und was ist dabei zu beachten, wenn Sie mehrere Tabelleneinträge in einem Befehl definieren möchten? Wie könnten Sie die Tabelle auf andere Weise definieren?

Aufgabe 4:

Schreiben Sie ein Programm, welches einen Speicherbereich anzeigt. Zuerst soll eine Anfangsadresse eingegeben werden können. Danach soll durch Betätigung den Tasten „+“ und „-“ durch den Speicher gegangen werden können, wobei in den Anzeigestellen S5,..,S2 die aktuelle Adresse und in S1,..,S0 das dort stehende Datum angezeigt werden soll.

Im einzelnen

- Anzeige löschen
- Kennung „Ad“ in Stelle S7..S6 anzeigen und eine Adresse in Stelle S5..S2 einlesen. Ende der Eingabe mit einer beliebigen Funktionstaste, die nicht ausgewertet wird.
- Kennung „dA“ an Stelle S7..S6 anzeigen, aktuelle Adresse an Position Stelle S5..S2 anzeigen, unter dieser Adresse stehendes Datum in Stelle S1..S0 anzeigen.
- Tastatur zyklisch abfragen:

- Taste „+“: aktuelle Adresse inkrementieren
- Taste „-“: aktuelle Adresse dekrementieren
- anderen Tasten: keine Funktion

- weiter mit 3).