Klausur 2001

Aufgabe 1

a) Adress: Programmrelative Adressierung mit Adressdistanz.

```
MOVE.W d(PC), D1.
```

b) Diese sind Schift-Befehle

ASL
ASR

c) Parameter, Rücksprungadressen, globale Register, lokale Variablen.

d) Makros sind Textersetzungen beim Kompilieren. U-Programme sind eine in sich abgeschlossene Befehlsfolge und die mehrfach gesprungen werden kann.

Name: *Matrikelnummer:*

Aufgabe 1: Assembler **(16 Punkte)**

a) **(2 Punkte)** Nennen Sie eine Adressierungsart des MC68000-Assembler, die besonders für im Speicher zusammen mit dem Programmcode verschiebbare Daten geeignet ist. Geben Sie außerdem einen beliebigen konkreten Befehl in MC68000-Assemblersyntax an, der diese Adressierungsart verwendet!

Folie 22.

MOVE.L d(PC),D0

MOVE.W d(PC),D1

b) **(2 Punkte)** Mit welchen Assembler-Befehlen kann eine Verdoppelung bzw. Halbierung eines ganzzahligen Wertes besonders effizient durchgeführt werden? Nennen Sie einen Beispielbefehl des MC68000-Assemblercodes!

schnell.

Verdopplung ASL, ASR ← halbierung

$1.0 \times 2^{1} \; 10.$ $\quad$ $100. \times 2^{-1}$ $\quad$ ASL *2

ASR /2

c) **(2 Punkte)** Nennen Sie mindestens drei verschiedene Arten von Daten (***nicht*** Typen wie z.B. Integer), die im Keller abgelegt werden!

lokale Daten
parameter
Registerinhalt.
Rücksprungadresse

d) **(1 Punkt)** Worin unterscheiden sich Makros und Unterprogramme?

Bei Makros wird beim compilieren der Text anstelle des Makro namen (im Programm) geschrieben. Nachteilig Programcode wird länger.

(Unterprogramme verhalten sich wie Programme)

Bei Unterprogrammen geschieht dies nicht es gibt lediglich ein hin und Rücksprung.

Punkte	

Name: *Matrikelnummer:*

Unterprogramme

Ein automatisches Übersetzungsprogramm soll ein als Zeichenkette übergebenes Wort aus einer Sprache in eine andere übersetzen.

Entwickeln Sie hierzu eine Assembler-Funktion, die ein vorhandenes Wörterbuch durchsucht und die gefundene Übersetzung zurückliefert. Zeichenketten seien byteweise nacheinander im Speicher abgelegt und mit einem Null-Byte als Endekennzeichen versehen (wie in C üblich). Falls eine Übersetzung gefunden wurde, muß sie an die durch `WortUebersetzt` vorgegebene Adresse kopiert werden. Es wird vorausgesetzt, dass dort ausreichend Speicherplatz für das Ergebnis vorhanden ist. Der `int`-Rückgabewert der Funktion soll 1 sein, falls eine Übersetzung gefunden wurde; 0 falls nicht. Er wird im Register D0 zum Zeitpunkt der Rückkehr aus der Funktion übergeben.

Die Assembler-Funktion soll folgende Signatur in C-Notation besitzen:

```
int Uebersetze(char *WortOriginal, char *WortUebersetzt);
```

Das vorgegebene Wörterbuch ist wie folgt aufgebaut: jedes Wortpaar wird durch zwei Zeiger auf die Zeichenketten der Wörter mit gleicher Bedeutung beschrieben. Die Konstanten `Woerterbuch` und `AnzahlWoerter` sollen im Programm verwendet werden. Im Wörterbuch vorhandene Zeichenketten enthalten mindestens einen Buchstaben.

```
struct Wortpaar
    {
        char *Original;                 // Wort im Original
        char *Uebersetzt;               // Wort uebersetzt
    };

#define AnzahlWoerter 1000              // Anzahl der Wortpaare
                                        // im Wörterbuch

Wortpaar Woerterbuch[AnzahlWoerter];  // Das Wörterbuch
```

Hinweis: Die Parameter liegen nach C-Konvention von rechts nach links im Keller und sind **dort** alle 4 Byte groß. Die Parameter `WortOriginal` und `WortUebersetzt` sind ein Zeiger auf das erste Element der Zeichenketten. Ein `int` ist ebenfalls 4 Bytes groß.

Punkte	

Name: *Matrikelnummer:*

e) **(2 Punkte)** Geben Sie den Kellerinhalt und den Kellerzeiger nach dem Aufruf der Funktion Uebersetze unmittelbar nach Sichern der Register an. Sie können die vorgegebenen Kästchen als Zeichenhilfe verwenden. Ein Kästchen sei dabei 4 Byte groß.

Distanz zum SP	Inhalt
	Wortübersetzer
	Wortargnal
	RSA
	A0
	A1
	A0
	D1
	D2 A'6

(32) ~~D2~~ 6 A
(28) 6 A
RSA (24) A
A3 (20)
A2 (16)
A1 (12)
A0 (8)
~~D2~~ (4) ESP (4)
D1 (0)

Punkte	

Name: Matrikelnummer:

Hinweis nur für Wiederholer: Alle Assemblerprogramme können sinngemäß auch in Modula-2-Inline-Assembler implementiert werden. Geben Sie dann immer den entsprechenden Prozedurkopf an!

f) **(7 Punkte)** Schreiben Sie die Funktion `Uebersetze` in Motorola 68000-Assembler. Berücksichtigen Sie die genannten Anforderungen.

Prozeduren ohne Dokumentation werden mit Null Punkten bewertet!

```
.GLOBAL. ÜBERSETZE,

_ÜBERSETZE:  MOVE_M.L  A0-A3/D1-D2, -(SP)
             MOVE.L    _wörterbuch, A1.
             MOVE.2.   ANzahlworter, D1.
             LLR       D.          // Rückgabewert:

STARTW:      MOVEA.L   28(SP), A0.   // originalwort:
             MOVEA.L   (A1), A3.     // Ad suchwort.

NEXT 2:      MOVE.B    (A0)+, D2.
             CMP.B     (A3)+, D2.
             BNE       NEXTW.
             TST.B     (A3)          wortende?
             BNE       Next2.
             TST.B     (A0).         wortende original.
             BNE       NEXTW.
             MOVEA.L   4(A1), A0.    Adresse übersetzen.
             MOVEA.L   32(SP), A1.   Adr. erg.

Zweiter:     MOVE.B    (A0)+, (A1)+.   ; Byte kopieren.
             TST.B     (A0)            fertig.
             BNE       zweiter
             MOVE.B    #0, (A1)        endzeichen kopie.
             MOVEQ.L   #1, D0          gefunden
                       fertig.

NEXTW        ADDA.L    A8, A1.
```

Punkte	

Anil 01

Aufg 1

```
.Global  -Übersicht-
.EVEN
MOVEM.L  D0-D7/A0-A6,(-SP)
MOVEA.L  68(SP),A0
MOVEA.L  64(SP),A1
MOVE.L   #0,D1

MOVEA    -Wörterbuch,A2.   -Anfangsadr. d.
                            Wörterbuch
MOVE.L   Anzwörter,D2

LOOP     CMP    -Anzwörter,D1
         BEQ    Wort gefunden.
```

Name: *Matrikelnummer:*

Aufgabe 2: Semaphore **(13 Punkte)**

Modellierung einer Auto-Fähre

Die betrachtete Fähre ist ein Schiff zum Transport von Autos über einen Fluss.

Die Fähre sowie die Autos werden durch Prozesse modelliert (für jedes Auto sowie die Fähre jeweils einen) und mit Hilfe von Semaphoren synchronisiert. Dabei sollen folgende Bedingungen gelten:

1. Autos fahren nur von links nach rechts über den Fluß.
2. Jedes Auto hat ein spezielles Gewicht (lokale Konstante).
3. Die maximale Ladung `MaxLadung` der Fähre (Summe der Gewichte der Autos auf der Fähre) darf nicht überschritten werden.
4. Die Fähre soll losfahren, wenn sie voll beladen ist oder wenn ein ankommendes Auto die maximale Ladung überschreiten würde.
5. Auf der rechten Seite des Flusses fahren die Autos von der Fähre, die danach zur linken Seite zurückkehrt.

Im folgenden sehen Sie, wie die oben geschilderten Synchronisationsbedingungen mit Semaphoren implementiert sind. Leider sind dabei ein paar **Fehler** unterlaufen.

Punkte	

Informatik 4 (April 2001)

Name: *Matrikelnummer:*

Globale Deklarationen:

```
100 FähreRechts     :    Semaphore(0)
110 FähreLinks      :    Semaphore(0)
120 FähreVoll       :    Semaphore(0)
130 FähreLeer       :    Semaphore(0)
140 Ladung_Schutz   :    Semaphore(5)  (1)
150 Ladung          :    INTEGER = 0
160 MaxLadung       :    CONST INTEGER = 10
```

Prozeß Fähre:

```
200   REPEAT
210         -- nach links fahren, anlegen
220         FähreLinks.V
230         FähreVoll.P
240         -- nach rechts fahren, anlegen
250         FähreRechts.V
260         FähreLeer.P
270   END REPEAT
```

Prozeß Auto:

```
500   -- Auto kommt an
510   FähreLinks.P
520   WHILE Ladung + Gewicht > MaxLadung DO
530         FähreVoll.V
540         FähreLinks.P
550   END WHILE
560   Ladung_Schutz.P
570   Ladung := Ladung + Gewicht
580   Ladung_Schutz.V
590   -- auf die Fähre fahren
600   FähreLinks.V
610   FähreRechts.P
620   Ladung_Schutz.P
630   Ladung := Ladung - Gewicht
640   -- von der Fähre herunterfahren
650   IF Ladung = 0 THEN FähreLeer.V
660   FähreRechts.V
670   -- Auto fährt weg
```

charger

décharger

Punkte	

Name: *Matrikelnummer:*

a) **(5 Punkte)** Simulieren Sie von Hand das Verhalten der vorgegebenen (**nicht korrekten**) Prozesse. Schreiben Sie in die erste Zeile zunächst den Anfangszustand; geben Sie für Semaphore den Wert des Semaphorzählers an. Gehen Sie im folgenden vom Zustand in einer Zeile aus und simulieren Sie das Verhalten der Prozesse nach Eintreffen des darunter in der linken Spalte genannten Ereignisses jeweils bis zu dem Punkt, an dem **alle** gestarteten Prozesse entweder blockiert oder komplett abgearbeitet sind. Achten Sie daher auch auf Prozesse, die weiter laufen können, nachdem ihre Blockierung endet! Geben Sie die Programmzeile an, an dem der letzte laufende Prozess anhält. Schreiben Sie dann den Zustand der Variablen und Sempahorzähler in die entsprechenden Felder der zum Ereignis gehörenden Zeile. Fahren Sie so Zeile für Zeile fort. Werte, die mit der Vorgängerzeile übereinstimmen, brauchen nicht noch einmal aufgeschrieben zu werden.

Ereignis	**letzte blockierende Programmzeile**	**FähreRechts**	**FähreLinks**	**FähreVoll**	**FähreLeer**	**Ladung_Schutz**	**Ladung**
Anfangszustand	X	0	6	0	0	5	0
Auto 1 kommt an (Gewicht=6)	510 610	- -1	-1 0			4 5	 6
Fährenprozess wird gestartet	230		0	-1			
Auto 2 kommt an (Gewicht=4)	510		-1				
Auto 3 kommt an (Gewicht=3)	510		-2				
Auto 4 kommt an (Gewicht=6)	510		-3				

Punkte	

Name: *Matrikelnummer:*

b) **(6 Punkte)** Die gegebene Implementierung hat logische Fehler (die jedoch nicht alle in Teil a aufgetreten zu sein brauchen) bzw. verhält sich nicht gemäß den Bedingungen. Geben Sie in der folgenden Tabelle die von Ihnen gefundenen Fehler und ihre Auswirkungen an. Schlagen Sie Korrekturmaßnahmen für das Programm vor, damit das Verhalten den Anforderungen entspricht.

Programmfehler und Auswirkungen	**Korrekturmaßnahme**

Punkte	

Name: _____ *Matrikelnummer:* _____

c) **(2 Punkte)** Ist die Verwendung der Semaphore `Ladung_Schutz` (zum Schutz des Zugriffs auf die Variable `Ladung`) notwendig? Begründen Sie Ihre Antwort!

Punkte	