HEINRICH-HEINE-UNIVERSITÄT DÜSSELDORF
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR RECHNERNETZE
PROFESSOR DR. M. MAUVE

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Grundlagen der Technischen Informatik

Übungsblatt 7

Abgabefrist: 05.06.2013 8:30 Uhr

Ansprechpartner: Der Tutor ihrer Übungsgruppe

Geben Sie zu jeder Aufgabe Ihren Lösungsweg in eigenen Worten an!

Aufgabe 7.1 *Kompressionsprogramm* (2 Punkte)

Sie möchten ein Kompressionsprogramm mit grafischer Benutzeroberfläche (GUI) entwickeln. Begründen Sie jeweils kurz, ob es sinnvoll wäre, folgende Komponenten Ihres Packers in Assembler statt in einer Hochsprache wie C/C++ zu schreiben:

- GUI / Benutzerinteraktion
- Datei-Ein- und Ausgabe
- Kompressionsalgorithmus.

Lösungsvorschlag: (2 Punkte)

Man programmiert in Assembler (wenn überhaupt) nur die besonders performancekritischen Teile eines Programms, um eine möglichst optimale Ausführungsgeschwindigkeit zu erzielen. Bei einem Packprogramm wäre nur der Kompressionsalgorithmus als performancekritisch einzustufen. Es macht keinen Sinn, den relativ umfangreichen Code für die GUI / Benutzerinteraktion in Assembler zu programmieren, da dies ein ziemlich fehlerträchtiges und sehr mühsames Unterfangen mit kaum messbarem Nutzen wäre. Während der Packer seine Hauptarbeit erledigt, also eine Datei komprimiert, passiert bei der GUI bis auf die Aktualisierung des Fortschrittsbalkens praktisch nichts. Die Steuerung der Datei-Ein- und Ausgabe erfolgt üblicherweise mittels einer überschaubaren Anzahl von I/O-Systemfunktionen in Verbindung mit einem Puffer und verbraucht vergleichsweise wenige CPU-Zyklen. Auch hier macht eine Assemblerimplementierung kaum einen Sinn.

Aufgabe 7.2 *Assembler-Analyse* (5 Punkte)
Gegeben ist das folgende, leider unkommentierte Assembler-Programm.

1. Ergänzen Sie das Programm um sinnvolle Kommentare ab Zeile 9, damit der Programmablauf verständlich wird.
2. Was ist die Ausgabe des Programms?
3. Was macht das Programm?

```
1  %include "asm_io.inc"
2
3  segment .data
4  codebase   dd    0deadbeefh
5  erg        db    "Ergebnis : ",0
6  segment .bss
7  segment .text
8    global asm_main
9  asm_main:
10   enter 0,0
11   pusha
12
13   mov eax, [codebase]
14   mov ecx, 32
15   mov ebx, 0
16 start:
17   shl eax,1
18   jnc continue
19   inc ebx
20 continue:
21   loop start
22
23   mov eax, erg
24   call print_string
25   mov eax, ebx
26   call print_int
27   call print_nl
28
29   popa
30   mov eax, 0
31   leave
32   ret
```

Lösungsvorschlag: (2 + 1 + 2 = 5 Punkte)

1.
```
1  %include "asm_io.inc"
2
3  segment .data
4  codebase  dd  0deadbeefh
5  erg   db  "Ergebnis: ",0
6  segment .bss
7  segment .text
8    global asm_main
9  asm_main:
10   enter 0,0          ; Setup, d.h. Stack Frame erzeugen
```

```
                        ; Parameter 1: 0 Bytes für lokale Variablen
                        ; Parameter 2: 0 nach C Calling-Conventions
  pusha                 ; Register auf dem Stack sichern

  mov eax, [codebase] ; initialisiert EAX mit der Hexzahl deadbeefh
  mov ecx, 32           ; initialisiert ECX als Schleifenzähler auf 32
  mov ebx, 0            ; initialisiert EBX mit 0
start:                  ; Label
  shl eax,1             ; Shift von EAX um eine Stelle nach links
  jnc continue          ; Jump zu Label continue, falls Carry Flag == 0
  inc ebx               ; 'else'-Teil: ebx = ebx + 1
continue:               ; Label
  loop start            ; (Jump zu Label start und ecx = ecx - 1),
                        ;   falls ecx > 0

  mov eax, erg          ; Adresse von erg nach EAX kopieren
  call print_string     ; String ausgeben, auf den Adresse in EAX zeigt
  mov eax, ebx          ; Inhalt von EBX nach EAX kopieren
  call print_int        ; Ergebnis ausgeben
  call print_nl         ; Neue Zeile ausgeben

  popa                  ; Register wiederherstellen
  mov eax, 0            ; Rückgabewert 0 nach EAX schreiben
  leave                 ; stack frame zerstören
  ret
```

2. `Ergebnis: 24`
3. Das Programm berechnet die Anzahl der Einsen in der Binärdarstellung der Hexadezimalzahl `deadbeef` und gibt sie auf der Kommandozeile aus.

Aufgabe 7.3 *Euklidscher Algorithmus (iterative Version)* (6 Punkte)
Der Euklidische Algorithmus berechnet den größten gemeinsamen Teiler (ggT) zweier natürlicher Zahlen. Die iterative Variante des Algorithmus lautet in Pseudocode:

```
Euclid(a, b)
   while (b != 0)
   {
      r = a mod b
      a = b
      b = r
   }
   return a
```

Implementieren Sie die Funktion *Euclid(a, b)* in Assembler. Übergeben Sie dabei die beiden Parameter `a` und `b` gemäß der C-Aufrufkonvention auf dem Stack. Die Rückgabe des Ergebnisses soll über das Register `EAX` erfolgen. Schreiben Sie anschließend das Hauptprogramm, welches von der Konsole die Zahlen `a` und `b` einliest, dann die Funktion *Euclid(a, b)* aufruft und schließlich den ggT auf der Konsole ausgibt. Kommentieren Sie Ihren Quelltext stichwortartig.

Kommentar für die Tutoren:

> Aufgabe 7.3 und 7.4 dürfen (ja eigentlich sollen sie sogar) das gleiche Assembler Rumpfprogramm zur Eingabe der Parameter und Ausgabe des Ergebnisses nutzen. Unterscheiden sollten sich die Abgaben eigentlich nur in der Implementierung der Euklid bzw. EuklidRek Funktionen. Die Punkteverteilung bei Aufgabe 7.3 ist daher 3 Punkte für den Rumpf mit ein und Ausgabe und dem Funktionsaufruf bzw. der Funktionsrückkehr und weitere 3 Punkte für die iterative Euklid Funktion. Für Aufgabe 7.4 gibt es nur Punkte für die rekursive Euklid Funktion. Der Ein-/Ausgabeteil wird nicht erneut bewertet.
> Sollte es Abgaben geben, bei denen nur Aufgabe 7.4, aber nicht 7.3 abgegeben wird, dann wertet bitte in Aufgabe 7.3 maximal 3 Punkte für den Ein-/Ausgabeteil und maximal 3 Punkte für die rekursive Euklid Implementierung in Aufgabe 7.4.

Lösungsvorschlag: (6 Punkte)

```
1  include "asm_io.inc"
2
3  ; initialized data is put in the .data segment
4  segment .data
5  ; These labels refer to strings used for output
6  prompt1 db    "a = ", 0           ; with null terminator
7  prompt2 db    "b = ", 0           ; with null terminator
8  prompt3 db    "ggT(a, b) = ", 0   ; with null terminator
9
10 ; code is put in the .text segment
11 segment .text
12         global  asm_main
13 asm_main:
```

```
14      enter   0,0         ; setup routine
15      pusha
16
17      mov  eax, prompt1
18      call print_string  ; ask the user to enter a
19      call read_int      ; read a and store it in eax
20      mov  ecx, eax      ; mov a to ecx
21      mov  eax, prompt2
22      call print_string  ; ask the user to enter b
23      call read_int      ; read b and store it in eax
24
25      push ecx           ; put a on the stack
26      push eax           ; put b on the stack
27      call euclid
28      add  esp, 8        ; clean up the parameters on the stack
29      mov  ecx, eax      ; save ggT to ecx
30
31      mov  eax, prompt3
32      call print_string  ; print out the result
33      mov  eax, ecx      ; pass ggT to print_int in eax
34      call print_int
35      call print_nl
36
37      popa
38      mov  eax, 0        ; return back to C
39      leave
40      ret
41
42  euclid:
43      mov  eax, [esp+8]  ; fetch parameter a from stack
44      mov  ebx, [esp+4]  ; fetch parameter b from stack
45  while:
46      cmp  ebx, 0        ; while (b != 0)
47      jz   end
48      mov  edx, 0        ; zero edx, 32-bit dividend in eax
49      div  ebx           ; eax = edx:eax / ebx, edx = edx:eax % ebx
50      mov  eax, ebx      ; a = b
51      mov  ebx, edx      ; b = a mod b, remainder in edx
52      jmp  while
53  end:
54      ret
```

Aufgabe 7.4 *Euklidscher Algorithmus (rekursive Version)* (3 Punkte)
Der Euklidischen Algorithmus kann auch rekursiv aufgeschrieben werden. Die rekursive Variante lautet in Pseudocode wie folgt:

```
EuclidRek(a, b)
   if (b == 0)
      return a
   else
      return EuclidRek(b, a mod b)
```

Implementieren Sie nun die Funktion *EuclidRek(a, b)* in Assembler. Übergeben Sie dabei die beiden Parameter `a` und `b` (als 32-Bit-Integer) gemäß der C-Aufrufkonvention auf dem Stack.

Die Rückgabe des Ergebnisses soll über das Register `EAX` erfolgen. Nutzen Sie anschließend das Hauptprogramm der letzten Aufgabe, um Ihre Funktion zu testen. Vergessen Sie nicht, Ihren Quelltext zu kommentieren.

Lösungsvorschlag: (3 Punkte)

```
EuclidRek: mov  eax, [esp+8]  ; fetch parameter a from stack
           mov  ebx, [esp+4]  ; fetch parameter b from stack
           cmp  ebx, 0
           jnz  recursion     ; if (ebx != 0) goto recursion
           ret                ; else return from function

recursion: mov  edx, 0        ; zero edx, 32-bit dividend in eax
           div  ebx
           push ebx           ; put ebx as parameter a on the stack
           push edx           ; put edx (a mod b) as par. b on the stack
           call EuclidRek
           add  esp, 8        ; clean up the parameters on the stack
           ret                ; return from function
```

C im Selbststudium

Lesen Sie das Kapitel 5 und 6 aus dem open book *C von A bis Z* von Jürgen Wolf (`http://openbook.galileocomputing.de/c_von_a_bis_z/`).