Hochschule Konstanz Technik, Wirtschaft und Gestaltung (HTWG)
Fakultät Informatik
Rechner- und Kommunikationsnetze
Prof. Dr. Michael Blaich

Übung zur Vorlesung Rechnerarchitektur AIN

Programmieraufgabe: Rekursive Aufrufe und lokale Arrays in Assembler

Die Abgabe erfolgt durch Hochladen der Lösung in Moodle. Zusätzlich wird die Lösung in der Übung nach dem Abgabetermin stichprobenartig kontrolliert.

Bearbeitung in Zweier-Teams

Team-Mitglied 1: `Tobias Latt`
Team-Mitglied 2: `Jannis Liebscher`

Aufgabe 1: Implementierung einer rekursiven Funktion

Implementieren Sie auf Papier die folgende rekursive Funktion als Prozedur in Assembler:

$$f(n,k) = \begin{cases} n+k+5 \text{ für } k-n>7 \\ f\big(n-1, max(8, g(k))\big) \text{ sonst} \end{cases}$$

Die Funktion $g(k)$ befindet sich an der Speicheradresse mit Label G: und Sie können davon ausgehen, dass die Funktion entsprechend der MIPS-Konventionen implementiert ist.

- Halten Sie sich bei den Implementierungen an die MIPS-Konventionen.
- Verwenden Sie keine Pseudo-Instruktionen außer `move`
- Verwenden Sie für bedingte Sprünge nur die Instruktionen `beq` und `bne`
- Unten finden Sie die Funktion in C. Sie müssen den C Code nicht eins-zu-eins nach Assembler übersetzen, sondern könne auch eine eigene Implementierung finden.
- Anbei finden Sie ein Prozedur G, mit der Sie ihre Implementierung testen können. Die Testprozedur liefert `G(k)=200+k` und überschreibt dabei alle Register außer den s-Registern.

```
1   int f(int n, int k) {
2   if (n-k>7) {
3       return n+k+5;
4   }else {
5       return f(n-1,max(8,g(k)));
6   }
```

Aufgabe 2: Lokale Variablen und Arrays

Implementieren Sie eine Prozedur `getPrime(n)`, die zu dem Übergabeparameter `n` die n. Primzahl bestimmt. Die Prozedur nutzt eine Prozedur `isDivisor(a,b)`, die bestimmt, ob a ein Teiler von b ist.

Um die Implementierung zu erleichtern können Sie die Pseudo-Instruktionen `li`, `move`, `bgt` und `blt` verwenden:

```
li $t0, c       # lädt Konstante c in Register $t0
move $t0, $t1   # kopiert Inhalt von Register $t1 in
                                    # Register $t0
bgt $t0, $t1, L # springt zu Label L, wenn $t0>$t1
bgt $t0, c, L   # springt zu Label L, wenn $t0>c
blt $t0, $t1, L # springt zu Label L, wenn $t0<$t1
blt $t0, c, L   # springt zu Label L, wenn $t0<c
```

Auf Moodle finden Sie ein Grundgerüst für das Hauptprogramm mit Ein- und Ausgabe sowie Makros für den einfacheren Zugriff auf Integer-Arrays.

Den C-Code der beiden Prozeduren finden Sie hier:

```
1   int isDivisor(int a, int b) {
2       // Prüft, ob a eine Teiler von b ist
3       // Prinzip: subtrahiere a solange von b, bis b<=0 ist
4       while (b>=0) {
5           if (b==0) {
6               return 1;   // Teiler
7           } else {
8               b-=a;
9           }
10      }
11      return 0;   // kein Teiler
12  }
```

```
13  int getPrime(int n)
14  {
15      // lokale Variablen
16      int p=2; // Primzahlkandidaten
17      int k=0; // Anzahl gefundene Primzahlen
18      int j=0; // Laufvariable für gefundene Primzahlen
19      int A[200]; // Array zum Speichern von Primzahlen
20
21      // Eingabeparameter prüfen
22      if (n>200) {
23          return 0;
24      }
25      // Suche n Primzahlen
26      while (k<n) {
27          // Prüfe, ob eine gefundene Primzahl Teiler von p ist
28          j=0;
29          while (j<k) {
30              if (isDivisor(A[j],p)) {
31                  // Teiler gefunden, keine Primzahl
32                  break;
33              }
34              j++;
35          }
36          if (j==k) {
37              // nächste Primzahl gefunden
38              A[k]=p; // in Array speichern
39              k++; // Anzahl gefundener Primzahlen erhöhen
40          }
41          // Nächste Zahl zum Testen
42          p++;
43      }
44      // Rückgabewert
45      return p-1;
46  }
```

Aufgabe 3: Ausführen eines Programms in MARS

Das Produkt zweier natürlicher Zahlen n*m lässt sich rekursiv wie folgt berechnen:
n*m=m*(n-1)+m=(m-1)*(n-1)+n+m-1

Das Programm rekmul.asm berechnet das Produkt zweier natürlicher Zahlen rekursiv. Es steht in Moodle zum Download zur Verfügung und lässt sich leicht mittels des MIPS-Simulators Mars ausführen.

Beantworten Sie die folgenden Fragen zu diesem Programm:

a) Was wird im Allgemeinen im Register $ra gespeichert? Erläutern Sie in diesem Zusammenhang die Zeilen 26 und 50 (jal rekmul), sowie Zeile 58 (jr $ra).
b) An welcher Zeile wird das Programm nach Ausführung von Zeile 58 fortgeführt?

a: Es wird die Rücksprungadresse der nächsten Instruktion gespeichert bei jal rekmul wird also zum label rekmul gesprungen und zusätzlich $pc + 4 in $ra gespeichert. mit jr $ra wird pc auf den wert gesetzt der in $ra gespeichert ist

b: Das programm wird in Zeile 28 bzw Zeile 51 fortgesetzt, je nachdem von welchem jal befehl gerade die adresse in $ra gespeichert wurde