Hochschule Konstanz
Fakultät Informatik

Prof. Dr.-Ing. Michael Blaich
Robotik und Künstliche Intelligenz

Übung Rechnerarchitekturen AIN 2 SoSe2025

2. Assemblerprogrammierung und MIPS Konventionen

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: Herbert Haase

Team-Mitglied 2: Tom Bonsiep

Hochschule Konstanz
Fakultät Informatik

Prof. Dr.-Ing. Michael Blaich
Robotik und Künstliche Intelligenz

Aufgabe 2.1 Daten und Arrays

Implementieren Sie in MARS ein Assemblerprogramm, das

1. im Speicher eine Zahl `n` und ein Array `A` mit `n` Integer-Werten anlegt und
2. die Summe der `n` Werte im Array bestimmt und in das Register `$v0` schreibt

Wählen Sie als Beispiel `n=6`.

Verwenden Sie zum Anlegen der Werte im Speicher die Assembler Direktive `.word`. Laden Sie die Variable `n` sowie die Adresse des Arrays mit den Pseudo-Instruktionen `lw Register, Label` bzw. `la Register, Label`. (Kapitel 2.6 bzw. Hilfe in MARS)

Aufgabe 2.2 Erste Prozedur

Implementieren Sie eine Prozedur (Label: `ISODD`) mit einem Integer-Wert `x` als Argument. Die Prozedur soll den Wert `1` zurückliefern, falls `x` ungerade ist und den Wert `0` zurückliefern, falls `x` gerade ist.
Tipp: Verwenden Sie die Instruktion `andi`.

Implementieren Sie eine zweite Prozedur (Label: `ISEVEN`), die komplementär zur ersten Prozedur arbeitet. Die Prozedur soll den Wert `1` zurückliefern, falls `x` gerade ist und den Wert `0` zurückliefern, falls `x` ungerade ist. Implementieren Sie diese Funktion, indem Sie die Funktion `ISODD` aufrufen und das Ergebnis invertieren.

Testen Sie die beiden Funktionen in MARS, in dem Sie sie nacheinander aufrufen und das Ergebnis in die Register `$s1` und `$s2` speichern.

Achten Sie auf die MIPS-Konventionen zur Implementierung von Prozeduren.

Hochschule Konstanz
Fakultät Informatik

Prof. Dr.-Ing. Michael Blaich
Robotik und Künstliche Intelligenz

Aufgabe 2.3 Prozeduren und Arrays

1. Erweitern Sie den Code aus Aufgabe 2.2 und legen Sie ein zweites Array `B` an, das zu Beginn mit `n` Nullen gefüllt ist.
2. Implementieren Sie eine Prozedur, die alle geraden Elemente eines Arrays `A` in ein Array `B` schreibt und die Anzahl der geraden Elemente zurückliefert. Argumente der Prozedur sind die Adressen der Arrays `A` und `B` sowie die Anzahl `n` der Elemente im Array. Der C-Code der Funktion ist unten gegeben. Halten Sie sich bei der Implementierung des Assembler-Code strikt an die Vorgaben der C-Funktion sowie die MIPS Konventionen.

```
int evenElem(int A[], int B[], int n) {
    int i=0;
    int j=0;
    while (i<n) {
        if (isEven(A[i])) {
            B[j]=A[i];
            j++;
        }
        i++;
    }
    return j;
}
```

3. Testen Sie ihre Prozedur mit dem Array `A=[3, 4, 6, 8, 11, 13]`.

Aufgabe 2.4 Rekursive Funktion

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?

Aufgabe 2.5 Implementierung einer rekursiven Funktion

Implementieren Sie 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.

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