

Prof. L. Thiele

Technische Informatik 1 - HS 2011

Übung 1

Datum: 13.10.2011

1 Aufgaben

Diese Übung soll Ihnen einen praktischen Einstieg in die MIPS-Programmierung geben. In der ersten Aufgabe erstellen Sie ein einfaches MIPS-Assembler Programm, welches das Skalarprodukt implementiert. Im zweiten Teil der Übung analysieren Sie das Resultat der Compilation einer Matrixmultiplikations-Funktion.

Bevor Sie mit der Übung beginnen, laden Sie von der TI1-Homepage das Archiv mit den benötigten Rahmenprogrammen herunter und entpacken Sie es in Ihrem Homeverzeichnis.

1.1 Skalarprodukt in MIPS Assembler

In dieser Aufgabe soll das Skalarprodukt zweier Vektoren ausgerechnet werden. Die Formel

$$s = \sum_i A_i \cdot B_i$$

soll als Funktion `skalar(int* A, int* B, int n)` in Assembler implementiert werden. Die Vektoren sind Arrays von 32-bit Integer Werten.

Im Verzeichnis "skalar" befinden sich entsprechende Vorlagen. Das C-Programm `skalar-main.c` ist das Rahmenprogramm zum Test der Skalarprodukt-Funktion. Das Skalarprodukt selbst wird im File `skalar.S` implementiert. Für Ihre Skalarprodukt-Implementierung gelten folgenden Pre- und Postconditions:

- In Register `a0` steht die Adresse des Vektors A. In Register `a1` steht die Adresse des Vektors B. Register `a2` enthält die Länge der Vektoren. Auf dem MIPS-Prozessor ist es eine Konvention, die `a`-Register (`a` steht für Argument) für Funktionsparameter zu verwenden.
- Ist das Skalarprodukt berechnet, soll das Resultat in Register `v0` abgespeichert werden. Dies ist wiederum eine Konvention, die auch vom C-Compiler eingehalten wird. D.h. wenn in einer aufrufenden Funktion `x = skalar(A,B, 100);` steht, erzeugt der Compiler Code, um Register `v0` nach dem Sprung zu `skalar` der Variablen `x` zuzuweisen (`x` ist ein Register oder steht im Speicher, je nach Typ der Variablen).
- In der Assemblerfunktion `skalar` im File `skalar.S` steht im vorgegebenen Code die fixe Zuweisung `li v0, 999`, damit die Funktion einen definierten Wert zurückgibt. Ersetzen Sie diese konstante Zuweisung nach der Implementierung mit dem tatsächlichen Skalarprodukt.

1.1.1 Arbeiten mit dem Debugger

Untersuchen Sie zunächst das gegebene Programmskelett mit Hilfe des MIPS-Simulators. In dieser einführenden Übung betrachten wir Compiler, Assembler und MIPS-Simulator als Blackbox. In einer nächsten Übung werden Sie Gelegenheit haben, die Funktion dieser Tools im Detail zu untersuchen.

Folgen Sie den folgenden Anweisungen, um das Programm zu compilieren und zu simulieren:

- Verwenden Sie das vorgegebene Makefile um `skalar.S` und das Rahmenprogramm `skalar-main.c` zu compilieren. Benutzen Sie dazu das Kommando `make`. Dabei wird das auf dem MIPS-Simulator ausführbare Programm `skalar` erstellt.
- Das kompilierte Programm lässt sich nun mit einem MIPS-CPU-Simulator ausführen. Wir verwenden dazu den im GNU Debugger (GDB) integrierten Simulator sowie Insight, ein grafisches Frontend zu GDB. Rufen sie Insight zusammen mit dem auszuführenden Binary auf:
`mips-elf-insight skalar &`
- Führen Sie das Programm mittels `Run` aus. Beim ersten Starten werden Sie aufgefordert, das Target einzurichten. Wählen sie unter Target `Simulator` und deselektieren Sie die Optionen `Set breakpoint at 'main'` und `Set breakpoint at 'exit'`. Kontrollieren Sie ausserdem, dass unter `More Options` die Optionen `Attach to Target`, `Download Program` und `Run Program` selektiert sind.

Erkunden Sie nun die Funktionen des Debuggers:

- Lassen Sie den Sourcecode `skalar-main.c` in verschiedenen Ansichten anzeigen (als C-Sourcecode, Assembler oder im gemischten C/Assembler-Modus)
- Setzen Sie nun einen Breakpoint an den Anfang der `main`-Funktion indem sie auf das Minus am Anfang der entsprechende Source-Code-Zeilen klicken.
- Führen Sie das Programm noch einmal aus. Bewegen Sie sich dann mit Hilfe der Funktionen `Step`, `Next`, `Finish` und `Continue` bis zum Anfang der Funktion `skalar`.
- Öffnen Sie nun die Registeransicht (`View` → `Register`), sowie die Speicheransicht (`View` → `Memory`). Finden Sie die Eingabevektoren `A` und `B` im Speicher. Was enthalten Sie?

1.1.2 Implementierung

Vervollständigen Sie die Implementierung der Skalarprodukt Funktion in `skalar.S`. Sie können das Programm anschliessend mit dem MIPS-Prozessor-Simulator testen.

Für die Implementierung benötigen Sie die Instruktion `mult`. Diese erwartet zwei Register als Eingabe und legt das Ergebnis in den Spezialregistern `hi` und `lo`¹ ab . Diese können Sie mit Hilfe der Befehle `mfhi` und `mflo` auslesen:

```
mult $Rsrc1, $Rsrc2      {$hi, $lo} = $Rsrc1 * $Rsrc2
mfhi $Rdst              $Rdst := $hi
mflo $Rdst              $Rdst := $lo
```

Neben den in der Vorlesung eingeführten MIPS-Instruktionen versteht der Compiler auch noch folgende Pseudo-Instruktionen, die sie verwenden können:

```
move $Rdest, $Rsrc      $Rdest := $Rsrc
li $Rdest, immediate   $Rdest := immediate
bgt $Rsrc1, $Rsrc2, $Label if ($Rsrc1 > $Rsrc2) then goto $Label
bge $Rsrc1, $Rsrc2, $Label if ($Rsrc1 >= $Rsrc2) then goto $Label
blt $Rsrc1, $Rsrc2, $Label if ($Rsrc1 < $Rsrc2) then goto $Label
ble $Rsrc1, $Rsrc2, $Label if ($Rsrc1 <= $Rsrc2) then goto $Label
blez $Rsrc, $Label     if ($Rsrc <= 0) then goto $Label
```

¹Zwei Ergebnisregister werden benötigt, da das grösstmögliche Ergebnis der Multiplikation zweier 32-Bit-Zahlen eine 64-Bit-Zahl ergibt. Für diese Übung reicht es, wenn sie davon ausgehen, dass das Ergebnis maximal 32 Bit gross ist und entsprechend das `hi`-Register ignorieren.

1.2 Matrixmultiplikation in C (Zusatzaufgabe)

Am Beispiel der Matrixmultiplikation $C = A \times B$ soll analysiert werden, wie der C-Compiler Code für die MIPS-Architektur erzeugt. Der Einfachheit halber wird angenommen, dass die beiden Matrizen gleich gross sind: $A: (n \times m)$, $B: (m \times n)$, $C: (n \times n)$. Das Produkt ist dann definiert als

$$c_{ij} = \sum_{k=1}^m a_{ik} \cdot b_{kj}$$

und kann in C zum Beispiel folgendermassen implementiert werden:

```
static void matrix(int *A, int *B, int *C, int n, int m) {
    /* multipiziere die Matrizen A(n x m), B(m x n), Resultat in C */
    int i,j,k,sum;
    for (i = 0; i < n; i++) { /* alle Zeilen */
        for (j = 0; j < n; j++) { /* alle Spalten */
            sum = 0;
            for (k = 0; k < m; k++) { /* Skalarprodukt */
                sum = sum + (A[i * m + k] * B[k * n + j]);
            }
            C[i * n + j] = sum;
        }
    }
}
```

Genau dieser C-Code (verwenden Sie das vorgegebene File `matrix.c`) soll nun übersetzt werden mit dem Befehl `mips-elf-gcc -fno-delayed-branch -O -c matrix.c`.

Mit dem Befehl `mips-elf-objdump --disassemble matrix.o` lässt sich der vom Compiler generierte MIPS-Code anschauen ('disassemblieren'). Die Aufgabe besteht nun darin, die Matrixmultiplikation zu analysieren und zu dokumentieren. Schreiben Sie zu jeder Assembler-Zeile einen Kommentar, was gerade gemacht wird! Verwenden Sie dazu auch Referenzen auf den ursprünglichen C-Code.

Hinweis zur Parameterübergabe: Da die Matrixmultiplikations-Funktion 5 Parameter benötigt, die MIPS-Konvention aber nur 4 Register für Funktionsparameter reserviert, wird das letzte Argument auf dem Stack übergeben werden. In der Funktion `matrix` wird der Parameter `m` mit dem Assembler-Befehl `lw` von der Adresse `16(sp)` (vom Stack) in ein temporäres Register geladen. `A`, `B`, `C` und `n` sind in `a0` bis `a3` abgelegt.