RA FS 12 Serie 2

Samuel Bucheli, Gian Calgeer, Judith Fuog

Die zweite Serie ist bis Donnerstag, den 29. März 2012 um 17:00 Uhr zu lösen und in schriftlicher Form in der Übungsstunde abzugeben. Für Fragen steht im ILIAS jederzeit ein Forum zur Verfügung. Allfällige unlösbare Probleme sind uns so früh wie möglich mitzuteilen, wir werden gerne helfen.
Viel Spass!

Theorieteil

Gesamtpunktzahl: 11 Punkte

1 Function Pointer (1 Punkt)

Was gibt folgendes Programmstück aus:

```
1  void callA(int a) {
2    printf("A: %i\n", a);
3  }
4
5  void callB(int a) {
6    printf("B: %i\n", a);
7  }
8
9  void callC(int a) {
10   printf("C: %i\n", a);
11 }
12
13 void (*functionPointer[3])(int) = { &callA, &callB, &callC };
14
15 functionPointer[0](10);
16 functionPointer[1](11);
17 functionPointer[2](12);
```

2 Const (2 Punkte)

Beschreiben Sie die unterschiedlichen Eigenschaften der folgenden vier Deklarationen.

```
1  int * a;
2  int const * b;
3  int * const c;
4  int const * const d;
```

3 Arrays (1 Punkt)

Was ist das Problem bei folgendem Programmstück:

```
1  int array[10] = {10, 0, 10, 0, 10, 0, 10, 0, 10, 0};
2  int i;
3  for (i=0; i<=10; ++i) {
4    printf("%i ", array[i]);
5  }
```

4 MIPS Branch Instructions (2 Punkte)

Nehmen Sie an, Register `$s2` enthalte den Wert 11, Register `$s1` enthalte den Wert 1. Beschreiben Sie mit äquivalentem C-Code was folgendes Beispiel bewirkt:

```
L1: beq $s2, $zero, L2
    #do something
    sub $s2, $s2, $s1
    j L1
L2:
```

5 MIPS More Branch Instructions (1 Punkt)

Wie wird der folgende Pseudo-Befehl vom Assembler erweitert?

```
bge $s2, $s1, Label
```

6 Endianness (1 Punkt)

Angenommen, ein 32-Bit `integer` werde als `word` an der Adresse 10004 abgespeichert:

Adresse	Binärwert
10004	0110 1101
10005	1000 1100
10006	0010 0100
10007	0000 0000

Welchen Wert hat die Zahl (Dezimal) und an welcher Adresse ist das *least significant byte* abgespeichert, wenn

(a) Big Endian

(b) Little Endian

als Byte-Reihenfolge verwendet wird?

7 Array-Zugriff (2 Punkte)

Schreiben sie folgendes C-Programmstück in Assembler um:

```
1 void mul(short x[], int index, int mul) {
2   x[index] *= mul;
3 }
```

Nehmen Sie dabei an, dass die Adresse des ersten Arrayelements im Register `$t2`, `index` in `$t1` und `mul` in `$t0` gespeichert sind.

8 MIPS Register/Hauptspeicher (1 Punkt)

Angenommen, A sei ein Array mit zehn Daten-Wörtern (im Hauptspeicher), die Basisadresse des Arrays befinde sich in `$s1`. Laden Sie das letzte Wort von A mit genau einem Befehl in das Register `$s2`.

Optionale Fragen

Die folgenden Fragen beziehen sich auf die Dateien aus dem Programmierteil, sie müssen nicht beantwortet werden, helfen aber beim Verständnis des Programmierteils.

Byte-Reihenfolge

Welche Endianness (Byte-Reihenfolge) verwendet unsere MIPS-Simulation (Big-Endian oder Little-Endian)? Begründen Sie Ihre Antwort.

Deklarationen

Was wird hier deklariert? Wozu wird dies später verwendet? (in `mips.h`)

```
210 extern Operation operations[OPERATION_COUNT];
211 extern Function functions[FUNCTION_COUNT];
```

Sign Extension

Beschreiben Sie, was die Funktion `word signExtend(halfword value)` bezweckt (in `mips.c`).

Tests

Beschreiben Sie die Tests, die die bereits implementiert sind (z.B. `void test_addi()` in `test.c`).

Programmierteil

Ihre Aufgabe ist es, das gegebene Programmgerüst wie folgt zu vervollständigen:

(a) Laden Sie die zu Beginn erwähnten Dateien von Ilias herunter und studieren diese aufmerksam. Versuchen Sie zu verstehen, was die bereits vorhandenen Teile bedeuten, die optionalen Fragen aus dem vorherigen Abschnitt können Ihnen dabei behilflich sein.

(b) Tragen Sie Ihren Namen sowie den Namen einer allfälligen Übungspartnerin oder eines allfälligen Übungspartners an den vorgesehenen Stelle in den Dateien `mips.c` und `test.c` ein.

(c) Implementieren Sie die Funktion `storeWord` (in `mips.c`).

(d) Schreiben Sie sinnvolle und ausführliche Tests für folgende MIPS-Operationen (in `test.c`).

`lw`, `ori` und `sub`

(e) Implementieren Sie die folgenden MIPS-Operation gemäss den Spezifikationen im Buch "Rechnerorganisation und -entwurf" von D.A. Patterson und J.L. Hennessy (in `mips.c`).

`add`, `addi`, `jal`, `lui` und `sw`

(f) Stellen Sie sicher, dass Ihre Implementation ohne Fehler und Warnungen kompilierbar ist, überprüfen Sie dies mit `make`

Dies ist eine notwendige Voraussetzung, damit der Programmierteil als erfüllt gilt.

(g) Stellen Sie sicher, dass Ihre Implementation die gegebenen und Ihre eigenen Tests ohne Fehler und Warnungen absolviert, überprüfen Sie dies mit `make test`

Dies ist eine notwendige Voraussetzung, damit der Programmierteil als erfüllt gilt.

(h) Erstellen Sie aus Ihrer Lösung eine Zip-Datei namens `<nachname>.zip` (wobei `<nachname>` natürlich durch Ihren Nachnamen zu ersetzen ist).

(i) Geben Sie die Datei elektronisch durch Hochladen in Ilias ab.

(j) Drucken Sie *zusätzlich* die Dateien `mips.c` und `test.c` aus und geben Sie diese in der Übungsstunde zusammen mit der restlichen Serie 2 ab.