



Cedric Aebi 17-103-235

Nicolas Müller 17-122-094

RA FS 18 Serie 2

Qiyang Hu, Givi Meishvili, Adrian Wälchli

Die zweite Serie ist bis Dienstag, den 27. März 2018 um 15:00 Uhr zu lösen und (wenn möglich als .zip Datei) nur auf ILIAS hochzuladen. 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](20);
16 functionPointer[1](21);
17 functionPointer[2](22);
```

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[11] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
2 int i;
3 for (i=0; i<=11; ++i) {
4     printf("%i ", array[i]);
5 }
```

4 MIPS Branch Instructions (2 Punkte)

Nehmen Sie an, Register `$s2` enthalte den Wert 4, `$s1` enthalte den Wert 1 und `$s3` enthalte den Wert 20. Beschreiben Sie mit äquivalentem C-Code was folgendes Beispiel bewirkt:

```
L1: beq $s2, $s3, L2
    #do something
    add $s2, $s2, $s1
    j L1
L2:
```

5 MIPS More Branch Instructions (1 Punkt)

Wie wird der folgende Pseudo-Befehl vom Assembler erweitert?

```
bge $s2, $s3, Label
```

6 Endianness (1 Punkt)

$\$s2 \geq \$s3$

Angenommen, ein 32-Bit `integer` werde als `word` an der Adresse 10001 abgespeichert:

Address	Stored binary value
10001	0101 1010
10002	1011 0110
10003	0101 1110
10004	1001 1010
10005	0110 1001

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 `$t4` gespeichert sind.

8 MIPS Register/Hauptspeicher (1 Punkt)

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

Optionale Fragen

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

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 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`

Sie finden die Spezifikationen auch im PDF “MIPS Reference Data”, das Sie von ILIAS herunterladen können (Literatur.zip)

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

1) Output: " A: 20
B: 21
C: 22 "

2) `int * a`: `a` ist ein Pointer des Typen `int`
`int const * b`: Zeiger auf einen konstanten `int`
`int * const c`: Ein Zeiger, welcher auf nichts
anderes Zeigen kann. Also Konstant-
er Zeiger auf `int`.

`int const * const d`: Ein konstanter Zeiger auf
konstanten `int`.

3) Das Problem ist, dass der Output am Schluss
eine "0" zuviel ausgibt. Dies liegt daran, dass
die for-Schleife bis `i <= 11` geht. Korrekt wäre
`i < 11`, da das Array nur 11 Elemente hat, und
mit `i = 11` ein Element aufgerufen werden soll,
welches nicht im Array ist.

4)

Nehmen Sie an, Register \$s2 enthalte den Wert 4, \$s1 enthalte den Wert 1 und \$s3 enthalte den Wert 20. Beschreiben Sie mit äquivalentem C-Code was folgendes Beispiel bewirkt:

```
L1: beq $s2, $s3, L2
    #do something
    add $s2, $s2, $s1
    j L1
L2:
```

```
for (int i = 4; i != 20; i++) {
    #do something
}
```

```
5) slt $at, $s2, $s3
    beq $at, $zero, Label
```

(\$at bekommt den Wert 1, wenn s2 kleiner ist als s3, dann wird mit beq geprüft ob \$at gleich null ist, wenn ja wird zum Label gesprungen. Also erst wenn $s3 \geq s2$.)

6)

Angenommen, ein 32-Bit integer werde als word an der Adresse 10001 abgespeichert:

Address	Stored binary value
10001	0101 1010
10002	1011 0110
10003	0101 1110
10004	1001 1010
10005	0110 1001

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?

a) lsb ist an der höchsten Speicheradresse (10005).

0101101010110110010111101001101001101001
 = (389606709865)₁₀

b) lsb ist an der niedrigsten Adresse (10001)

011010011001101001011110101101100101
 1010
 = (453561464410)₁₀

7 Array-Zugriff (2 Punkte)

Schreiben sie folgendes C-Programmstück in Assembler um:

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

Nehmen Sie dabei an, dass die Adresse des ersten Arrayelements im Register \$t2, index in \$t1 und mul in \$t4 gespeichert sind.

1. Array Element : \$t2

index : \$t1

mul : \$t4

```
mul:  add $t3, $t1, $t1    // Index * 2 berechnen  
      add $t0, $t3, $t2    // Adresse von x[index] holen  
      mult $t0, $t4        // multiplizieren  
      mflo $t4             // Resultat holen  
      jr $ra               // Junpreturn
```

8 MIPS Register/Hauptspeicher (1 Punkt)

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

Annahme int:

```
lw $s2, 28($s1)
```