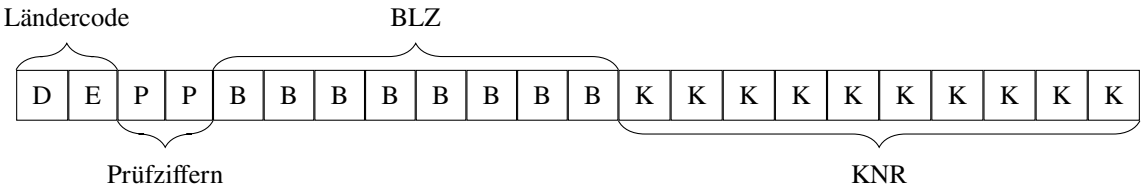
Projekt 1

Prof. Dr. Sebastian Hack
Prof. Dr. Jörg Hoffmann
M. Sc. Rebecca Eifler
M. Sc. Maximilian Fickert

Vorbereitungen

Klonen Sie das Depot `https://prog2scm.cdl.uni-saarland.de/git/project1/$NAME`, wobei `$NAME` durch Ihren Benutzernamen zu ersetzen ist. Geben Sie an, dass der Klon in ein Verzeichnis namens `project1` abgelegt wird. Falls Sie das Vorkursprojekt (*Projekt 0 – Einführung in Versionsverwaltung mit git*) nicht bearbeitet haben sollten, müssen Sie noch *Aufgabe 1 – Vorbereitung der git-Installation* aus diesem Projekt erledigen. Außerdem möchten wir Sie darauf hinweisen, dass unser Server nur aus dem Universitätsnetz erreichbar ist. Wenn Sie von zu Hause arbeiten möchten, können Sie auf folgenden Seiten nachlesen wie Sie den VPN-Client installieren `https://www.hiz-saarland.de/dienste/vpn/` und das VPN des Host-Systems in der VM nutzen `https://prog2.cs.uni-saarland.de/p2ss18/4/Software`.
Beachten Sie, dass Sie die Einstellungen *Assemble all files in directory* und *Initialize Program Counter to global 'main' if defined* im Menü *Settings* in MARS aktivieren müssen, um die Datei `src/main.s` zu übersetzen, und den Programmeinstiegspunkt auf die Marke `main` zu setzen.
Im Folgenden bezieht sich der Begriff Zeichen immer auf die ein Byte große Kodierung eines Zeichens gemäß des ASCII Standards. Die IBANs enthalten nur ASCII-Großbuchstaben und -Ziffernzeichen. Die KNRs und die BLZs bestehen ausschließlich aus ASCII-Ziffernzeichen.

IBAN-Rechner (10 Punkte)



In diesem Projekt werden Sie einen IBAN-Rechner implementieren, der eine Kontonummer (KNR) und eine Bankleitzahl (BLZ) in eine deutsche IBAN nach obigem Format übersetzt, bzw. aus einer deutschen IBAN die KNR und die BLZ extrahiert. Sie finden eine allgemeine Einführung zu IBANs in der Wikipedia (`http://de.wikipedia.org/wiki/IBAN`).
Sie werden das Programm in MIPS-Assembler schreiben. Zur einfacheren Bearbeitung ist das Projekt in mehrere Aufgaben aufgeteilt. Zu jeder Aufgabe gibt es eine eigene Datei, in welche Sie das jeweilige Unterprogramm schreiben werden. Beachten Sie auch unbedingt die Hinweise am Ende der Projektbeschreibung.

1 BLZ und KNR extrahieren (2 Punkte)

Schreiben Sie ein Unterprogramm `iban2knr` (Datei `src/iban2knr.s`), das die KNR und die BLZ aus einer deutschen IBAN extrahiert. Ihr Unterprogramm erhält folgende Argumente:

1. Einen Puffer mit einer deutschen IBAN mit 22 Zeichen.
2. Den Zielpuffer für die BLZ mit Platz für 8 Zeichen.
3. Den Zielpuffer für die KNR mit Platz für 10 Zeichen.

2 Rest berechnen (4 Punkte)

Schreiben Sie ein Unterprogramm, `modulo_str` (Datei `src/moduloStr.s`), das den Divisionsrest einer Zahl berechnet, die als Zeichenkette gegeben ist. Ihr Unterprogramm erhält folgende Argumente:

1. Den Puffer mit der Zahl.
2. Die Anzahl der Ziffern im Puffer.
3. Den Teiler.

Das Ergebnis entspricht dem Rest der im Puffer dargestellten Zahl bei einer Division durch den gegebenen Teiler und wird im Rückgaberegister zurückgegeben. Dabei stellt das letzte Zeichen im Puffer an Position $Länge - 1$ die niedrigwertigste Ziffer dar. Die Zahl im Puffer ist als Dezimalzahl in Ziffernzeichen dargestellt und hat insbesondere kein Vorzeichen. Der Teiler ist eine vorzeichenlose Zahl und immer größer als 0. Wir legen Ihnen nahe das Ergebnis ziffernweise nach dem Hornerschema zu berechnen. Beachten Sie, dass das Ergebnis auch korrekt sein muss, wenn die im Puffer dargestellte Zahl das Fassungsvermögen der Maschinenregister überschreitet. Dies können Sie durch geschickte Ausnutzung der folgenden Gleichungen erreichen ($x \bmod y$ ist der Rest der Division von x durch y):

$$(a + b) \bmod c = (a + (b \bmod c)) \bmod c$$

$$(a \times b) \bmod c = (a \times (b \bmod c)) \bmod c$$

Sie können also Zwischenergebnisse auf ihren Rest bei Division durch den Teiler reduzieren.

3 IBAN Prüfziffer validieren (2 Punkte)

Schreiben Sie ein Unterprogramm `validate_checksum` (Datei `src/validateChecksum.s`), das die Prüfziffer einer deutschen IBAN validiert, die als Zeichenkette gegeben ist. Ihr Unterprogramm erhält folgendes Argument:

1. Die Startadresse eines 22 Zeichen langen Puffers mit einer deutschen IBAN.

Das Unterprogramm gibt das Ergebnis der folgenden Rechnung im Rückgaberegister zurück. Folgende Schritte zeigen anschaulich, wie das Ergebnis berechnet wird.

1. Stellen Sie die vier ersten Stellen an das Ende der IBAN.
2. Ersetzen Sie alle Buchstaben durch Zahlen, wobei A = 10, B = 11, … , Z = 35.
3. Berechnen Sie den Rest, der bei ganzzahliger Division durch 97 bleibt. Dies ist der Rückgabewert.

Ein Beispiel:

IBAN:	DE68	210501700012345678		
Umstellung:		210501700012345678	DE	68
Buchstaben:		210501700012345678	1314	68
Modulo:		210501700012345678	1314	68 mod 97 = 1

Verwenden Sie das Unterprogramm `modulo_str` aus der vorherigen Aufgabe, um den Rest zu berechnen.

Ist das Ergebnis 1, ist die Prüfziffer gültig.

4 IBAN erzeugen (2 Punkte)

Schreiben Sie ein Unterprogramm `knr2iban` (Datei `src/knr2iban.s`), das aus einer KNR und einer BLZ eine deutsche IBAN mit gültigen Prüfziffern berechnet. Ihr Unterprogramm erhält folgende Argumente:

1. Der Zielpuffer für die IBAN mit Platz für 22 Zeichen.
2. Die BLZ als 8 Zeichen langer Puffer.
3. Die KNR als 10 Zeichen langer Puffer.

Um die Prüfziffer zu berechnen kann die Funktion zur Validierung einer Prüfziffer aus der letzten Aufgabe verwendet werden. Dazu werden die beiden Stellen nach dem Ländercode der IBAN auf 0 gesetzt. Die Prüfziffer ergibt sich dann als die Differenz von 98 und dem Ergebnis der Validierungsfunktion. Ist die Prüfziffer einstellig wird sie mit einer führenden 0 aufgefüllt.

5 IBAN Prüfziffer validieren ohne Zwischenspeicher (2 Bonuspunkte)

Schreiben Sie eine Variante von `validate_checksum` (Datei `src/bonus/validateChecksum.s`), welche die IBAN Prüziffer validiert ohne Zwischenergebnisse in den Speicher zu schreiben. Das Unterprogramm muss den gleichen Rückgabewert wie `validate_checksum` aus Aufgabe 3 liefern. Konkret disqualifiziert jeglicher Syscall oder Schreibzugriff auf den Speicher die Lösung.

Tests selber ausführen und debuggen

Verwenden Sie den Befehl `./run_tests` im Hauptverzeichnis des Projekts, um ihre Implementierung testen zu lassen. Wenn ein Test, z.B. `tests/pub/test_X.s`, fehlschlägt können Sie ihn in MARS debuggen. Der Befehl `./build_testbox tests/pub/test_X.s` kopiert den Test zusammen mit ihrer Implementierung in den neuen Ordner `testbox/`, wo Sie ihn mit MARS starten können. Beachten Sie jedoch, dass nur die Dateien an den in der Aufgabenstellung angegebenen Pfaden in die Bewertung und die automatischen Tests eingehen. Buchen Sie also insbesondere keine Dateien in `testbox/` in das Versionsverwaltungssystem ein.

Skripte für die Bonusaufgabe

Analog zu den regulären Testprogrammen gibt es die Skripte `run_tests_bonus` und `build_bonusbox` mit denen Sie Ihre Lösung der Bonusaufgabe debuggen können. Das Skript `build_bonusbox` kopiert außerdem das Projekt mit der Implementierung der Bonusaufgabe in den Ordner `bonusbox/`, wo Sie es mit MARS starten können.

Abgabe Ihrer Lösung

git push bis 30. April 2018, 23:59.

Hinweise

1. Die Datei `src/util.s` enthält nützliche Unterprogramme, die Sie für dieses Projekt verwenden können. Einige der dort implementierten Unterprogramme sollten Ihnen aus dem MIPS-Übungsblatt bekannt sein. Diese Datei sollten Sie nicht verändern, da wir für die automatischen Tests die Originalversion verwenden.
2. Denken Sie daran, dass Ihre Unterprogramme auch die Aufrufkonvention implementieren müssen.
3. Für Ihre Abgabe werden ausschließlich die Dateien im Ordner `src/` berücksichtigt, deren Namen in der Aufgabenstellung angegeben sind.
4. Beachten Sie, dass nur die auf der virtuellen Maschine installierte Version von MARS von uns unterstützt wird. Diese weicht von den offiziellen Versionen in Details ab.
5. Wir weisen aus rechtlichen Gründen darauf hin, dass keinerlei Garantien für die erzeugten IBANs, BLZs und KNRs gegeben werden.
6. Die Beschreibung des Verfahrens zur Berechnung der Prüfsumme ist weitestgehend von der auf Wikipedia beschriebenen vorgeschlagenen Methode (`http://de.wikipedia.org/wiki/IBAN#Validierung_der_Pr.C3.BCfsumme`) zur Verifizierung der Prüfsumme entnommen.
7. Dokumentieren Sie Ihr Programm mit Codekommentaren. Dadurch wird der Code lesbarer und Sie können das Programm leichter in MARS debuggen.