2048 (12+1 Punkte)

In diesem Projekt werden Sie das beliebte Spiel 2048 [1, 2] in MIPS implementieren.

1 Das Spiel

2048 ist ein Denkspiel, bei dem man Steine auf dem Spielfeld verschieben muss. Ziel ist es, Steine mit immer höheren Werten bis hin zur 2048 zu erzeugen. Das Spiel beginnt mit zwei Steinen, die entweder den Wert 2 oder 4 haben. In jeder Runde darf der Spieler eine Richtung wählen, in die alle Steine – falls möglich – geschoben werden. Ein Stein kann nur verschoben werden, falls das Nachbarfeld in der gewählten Richtung frei ist. Falls auf dem nächsten belegten Feld in dieser Richtung ein Stein des gleichen Wertes ist, werden beide entfernt und ein neuer Stein entsteht auf diesem Feld mit dem summierten Wert der beiden alten. Nach jeder Runde wird ein neuer Stein vom Wert 2 oder 4 zufällig auf dem Spielfeld plaziert. Das Spiel ist verloren, sobald es keine Richtung mehr gibt in die man die Steine auf dem Spielfeld verschieben kann. Dies ist der Fall, wenn alle Felder belegt sind und es keine zwei Steine des gleichen Wertes gibt, die nebeneinander liegen. In allen anderen Fällen kann man entweder Steine in die noch freien Felder schieben oder benachbarte Steine des gleichen Wertes zu einem neuen Stein verschmelzen und damit ein freies Feld gewinnen.

Abbildung 1: Grafische Benutzerschnittstelle

2 Vorbereitung

Git

Falls Sie nicht am Vorkurs teilgenommen haben, müssen Sie Ihr Git noch konfigurieren. Führen Sie dazu die folgenden Befehle aus:

- `git config --global user.name "Vorname Nachname"`
- `git config --global user.email "kennung@stud.uni-saarland.de"`

Dabei ersetzen Sie Vorname und Nachname durch Ihren Vor- bzw. Nachnamen und kennung durch Ihre studentische Kennung.

Klonen Sie das Projekt-Depot in einen Ordner mit dem Namen `project1`:

- `git clone https://prog2scm.cdl.uni-saarland.de/git/project1/$username project1`

Ersetzten Sie dabei `$username` durch Ihren Benutzernamen. Wir möchten Sie drauf hinweisen, dass unser Server nur aus dem Universitätsnetz erreichbar ist.

MARS

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.asm` zu übersetzen, und den Programmeinstiegspunkt auf die Marke `main` zu setzen.

3 Implementierung

Ein Teil der Implementierung ist bereits gegeben. Dazu gehören zum Beispiel das Einlesen der Tastatureingabe und die Koordinierung des Spielablaufes. Diesen vorgegeben Teil müssen Sie durch die Implementierung einzelner Unterfunktionen vervollständigen. In den folgenden Abschnitten wird die Darstellung des Spielfeldes im Speicher und die Übergabe des Spielfeldes an Ihre Unterprogramm beschrieben.

3.1 Spielfeld

Das Spielfeld wird durch *vorzeichenlose Halbwörter* an aufeinander folgenden Adressen im Speicher dargestellt. Das bedeutet, dass zum Beispiel das Feld mit den Koordinaten $(2, 2)$ an der Adresse Basisadresse $+20$ im Speicher liegt. Die Basisadresse ist die Adresse des ersten Feldes. Abbildung 2 veranschaulicht die Darstellung im Speicher grafisch. Die Darstellung des Spielfeldes aus Abbildung 1 ist in Abbildung 3 zu sehen.

$(0, 0)$	$(1, 0)$	$(2, 0)$	$(3, 0)$
$(0, 1)$	$(1, 1)$	$(2, 1)$	$(3, 1)$
$(0, 2)$	$(1, 2)$	$(2, 2)$	$(3, 2)$
$(0, 3)$	$(1, 3)$	$(2, 3)$	$(3, 3)$

Basisadresse

$(0, 0)$	$(1, 0)$	$(2, 0)$	$(3, 0)$	$(0, 1)$	$(1, 1)$	$(2, 1)$	$(3, 1)$	$(0, 2)$	$(1, 2)$	$(2, 2)$	$(3, 2)$	$(0, 3)$	$(1, 3)$	$(2, 3)$	$(3, 3)$
$+0$	$+2$	$+4$	$+6$	$+8$	$+10$	$+12$	$+14$	$+16$	$+18$	$+20$	$+22$	$+24$	$+26$	$+28$	$+30$

Abbildung 2: Darstellung des Spielfeldes im Speicher

Basisadresse

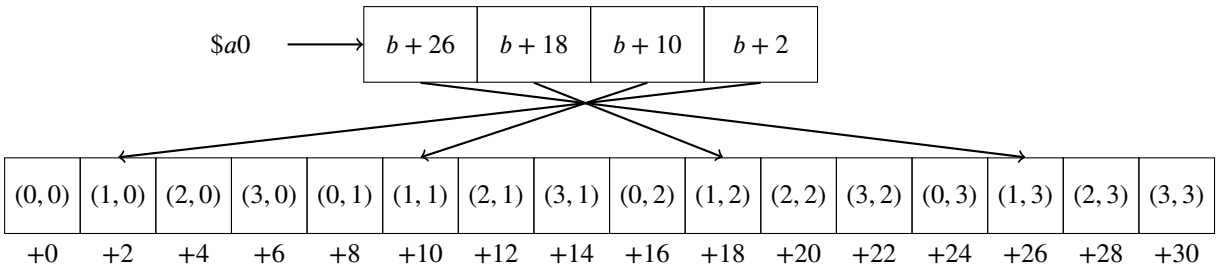
$+0$	$+2$	$+4$	$+6$
0000000000000010	0000000000000100	0000000000000000	0000000000000000
$+8$	$+10$	$+12$	$+14$
0000000000000100	0000000000001000	0000000000000000	0000000000000000
$+16$	$+18$	$+20$	$+22$
0000000000010000	0000000000100000	0000000000000100	0000000000000010
$+24$	$+26$	$+28$	$+30$
0000000000000010	0000100000000000	0000001000000000	0000000000001000

Abbildung 3: Darstellung des Spielfeldes aus Abbildung 1 im Speicher

3.1.1 Übergabeparameter

Wird eine Zeile/Spalte des Spielfeldes einer Funktion als Parameter übergeben, wird immer eine Adresse auf eine Reihung der Adressen der benötigten Felder übergeben. Dies ermöglicht durch die Anordnung der Spielfelder in der übergebenen Reihung die Richtung des Spielzuges zu bestimmen. Deshalb müssen Sie in den folgenden Aufgaben nur den Fall eines Zuges nach links implementieren. Das Aufrufen der Unterprogramme mit der richtigen Anordnung der Spielfelder ist bereits vorgegeben.

Beispiel Wird zum Beispiel die zweite Spalte für einen Spielzug nach unten übergeben, erhält das Unterprogramm die Adresse der folgenden Reihung:



4 Aufgaben

Sie müssen nur die im Folgenden beschriebenen Unterprogramme implementieren. Jedes Unterprogramm wird in einer eigenen Datei implementiert. Dort finden Sie auch eine weitere Beschreibung der Übergabeparameter und des Rückgabewertes. Alle anderen Dateien (`main.asm, buffer.asm, utils.asm`) dürfen Sie **nicht** ändern.

4.1 Überprüfung Sieg (1 Punkt)

Das Spiel ist gewonnen, wenn man einen Stein mit dem Wert 2048 erreicht. In dieser Aufgabe sollen Sie ein Unterprogramm in der Datei `check.asm` implementieren, das überprüft ob im Spielfeld ein Stein mit dem Wert 2048 enthalten ist.

Parameter

1. `$a0` Adresse der Reihung aller Felder des Spielfeldes
2. `$a1` Länge der Reihung

Das Unterprogramm gibt 1 zurück, wenn das Spielfeld einen Stein mit dem Wert 2048 enthält. Sonst gibt es 0 zurück.

Hinweis: Das Unterprogramm muss für Spielfelder der Größe $n \in \mathbb{N}$ mit $n \geq 2$ implementiert werden. Deshalb wird die Länge der Reihung als Parameter übergeben und ist damit nicht auf 4 beschränkt.

4.2 Stein setzen (1 Punkt)

Nach jedem Spielzug wird ein neuer Stein mit dem Wert 2 oder 4 auf einem beliebigen freien Feld plaziert. In dieser Aufgabe sollen Sie ein Unterprogramm in der Datei `place.asm` implementieren, welches einen Wert an einer Position einfügt, wenn das Feld nicht schon belegt ist.

Parameter

1. `$a0` Adresse der Reihung aller Felder des Spielfeldes
2. `$a1` Länge der Reihung
3. `$a2` Position des Feldes

4. `$a3` Wert der gesetzt werden soll

Der Wert soll nur gesetzt werden, wenn das Feld an der gegebenen Position leer ist (den Wert 0 enthält). Konnte der Wert gesetzt werden, soll das Unterprogramm 0 zurück geben. Konnte der Wert auf Grund einer belegten Position nicht gesetzt werden, soll das Unterprogramm 1 zurück geben.

4.3 Spielzug möglich (2 Punkte)

Das Spiel ist verloren, wenn in keine der vier Richtungen ein Zug mehr möglich ist. In dieser Aufgaben sollen Sie ein Unterprogramm in der Datei `move_check.asm` implementieren, welches überprüft, ob in der übergebenen Reihung ein Zug nach links möglich ist.

Parameter

1. `$a0` Adresse einer Reihung mit den Adressen aller Felder der Zeile/Spalte
2. `$a1` Länge der Reihung

Ein Zug ist möglich, wenn es ein leeres Feld (Feld mit dem Wert 0), das links von einem belegten Feld (Wert > 0) liegt gibt, oder zwei nebeneinander liegende Felder den gleichen Wert ($\neq 0$) haben.
Wenn ein Zug möglich ist, gibt das Unterprogramm 1 zurück, sonst 0.

Hinweis: Das Unterprogramm muss für Spielfelder der Größe $n \in \mathbb{N}$ mit $n \geq 2$ implementiert werden. Deshalb wird die Länge der Reihung als Parameter übergeben und ist damit nicht auf 4 beschränkt.

4.4 Spielzug ausführen (5 Punkte)

Ein Spielzug ist unterteilt in zwei 2 Komponenten, das Verschieben und das Verschmelzen von Spielsteinen. Im Folgenden werden Sie mehrere Unterprogramme implementieren, die verwendet werden einen Spielzug auszuführen.

Hinweis: Für alle Teilaufgaben gilt: Das Unterprogramm muss für Spielfelder der Größe $n \in \mathbb{N}$ mit $n \geq 2$ implementiert werden. Deshalb wird die Länge der Reihung als Parameter übergeben und ist damit nicht auf 4 beschränkt.

4.4.1 Steine um eins verschieben

In dieser Aufgabe sollen Sie ein Unterprogramm in der Datei `move_one.asm` implementieren, das alle Spielsteine in der übergebenen Reihung, wenn möglich, eins nach links verschiebt.

Parameter

1. `$a0` Adresse einer Reihung mit den Adressen aller Felder der Zeile/Spalte
2. `$a1` Länge der Reihung

Das Unterprogramm gibt 1 zurück wenn mindestens ein Spielstein verschoben wurde. Sonst gibt es 0 zurück

Beispiel Im folgenden Beispiel werden die Werte der Spielsteine anstelle ihrer Adressen verwendet um die Darstellung zu vereinfachen.

Eingabe

0	0	2	4

Ausgabe

0	2	4	0

4.4.2 Steine soweit wie möglich verschieben

In dieser Aufgabe sollen Sie ein Unterprogramm in der Datei `move_left.asm` implementieren, das alle Spielsteine in der übergebenen Reihung soweit wie möglich nach links verschiebt. Hinweis: Verwenden Sie dazu das Unterprogramm aus Aufgabenteil 4.4.1.

Parameter

1. `$a0` Adresse einer Reihung mit den Adressen aller Felder der Zeile/Spalte
2. `$a1` Länge der Reihung

Das Unterprogramm gibt nichts zurück.

Beispiel Im folgenden Beispiel werden die Werte der Spielsteine anstelle ihrer Adressen verwendet um die Darstellung zu vereinfachen.

Eingabe

0	2	0	4

Zwischenergebnis

2	0	4	0

Ausgabe

2	4	0	0

4.4.3 Steine verschmelzen

Die zweite Komponente eines Spielzuges ist das Verschmelzen zweier Spielsteine. In dieser Aufgabe implementieren Sie ein Unterprogramm in der Datei `merge.asm`, welches zwei nebeneinander liegende Spielsteine des gleichen Wertes verschmelzt. Zwei Spielsteine werden verschmolzen indem der linke Stein durch die Summe der beiden Steine und der rechte durch 0 ersetzt wird. Verschmelzen Sie die Steine von links nach rechts.

Parameter

1. `$a0` Adresse einer Reihung mit den Adressen aller Felder der Zeile/Spalte
2. `$a1` Länge der Reihung

Das Unterprogramm gibt nichts zurück.

Beispiel Im folgenden Beispiel werden die Werte der Spielsteine anstelle ihrer Adressen verwendet um die Darstellung zu vereinfachen.

Eingabe

2	2	4	4

Ausgabe

4	0	8	0

4.4.4 Kompletten Spielzug ausführen

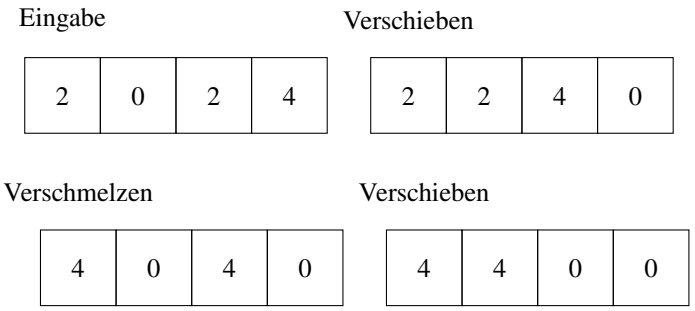
In dieser Aufgabe kombinieren Sie die beiden Unterprogramme zum Verschieben (Aufgabenteil 4.4.2) und Verschmelzen (Aufgabenteil 4.4.3) der Spielsteine um alle Steine so weit wie möglich nach links zu verschieben und zu verschmelzen.
Schieben Sie dazu erst alle Steine so weit wie möglich nach links. Verschmelzen Sie dann einmal alle möglichen Steine. Schieben Sie dann wieder alle Steine so weit wie möglich nach links.
Implementieren Sie das Unterprogramm in der Datei `complete_move.asm`.

Parameter

1. `$a0` Adresse einer Reihung mit den Adressen aller Felder der Zeile/Spalte
2. `$a1` Länge der Reihung

Das Unterprogramm gibt nichts zurück.

Beispiel Im folgenden Beispiel werden die Werte der Spielsteine anstelle ihrer Adressen verwendet um die Darstellung zu vereinfachen.
Beachten Sie, dass Steine pro Spielzug nur einmal verschmolzen werden. Deshalb werden im gezeigten Beispiel die beiden Spielsteine mit dem Wert 4 nicht verschmolzen.



4.5 Ausgabe Spielfeld (3 Punkte)

Um das Spiel 2048 nun spielen zu können, fehlt noch eine graphische Darstellung des Spielfeldes. In dieser Aufgabe implementieren Sie ein Unterprogramm, das ein Spielfeld der Größe 4×4 und maximal vierstelligen Werten auf der Kommandozeile ausgibt.

Parameter

1. `$a0` Adresse einer Reihung aller Felder des Spielfeldes

Das Unterprogramm gibt nichts zurück. Implementieren Sie das Unterprogramm in der Datei `printboard.asm`. Das folgende Beispiel veranschaulicht, wie die Ausgabe aussehen soll.

```
-----------------------------
|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|
|␣␣␣␣2␣|␣␣␣␣4␣|␣␣␣␣0␣|␣␣␣␣0␣|
|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|
-----------------------------
|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|
|␣␣␣␣4␣|␣␣␣␣8␣|␣␣␣␣0␣|␣␣␣␣0␣|
|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|
-----------------------------
|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|
|␣␣␣16␣|␣␣␣32␣|␣␣␣␣4␣|␣␣␣␣2␣|
|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|
-----------------------------
|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|
|␣␣␣␣2␣|␣2048␣|␣␣512␣|␣␣␣␣8␣|
|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|␣␣␣␣␣␣␣|
-----------------------------
```

Das Zeichen ␣ seht für ein Leerzeichen und wird nur verwendet um die Anzahl der verwendeten Leerzeichen zu verdeutlichen. Fügen Sie keine weiteren Leerraum oder Zeilenumbrüche ein. Für jeden Wert sind 4 Stellen vorgesehen. Hat die Zahl weniger als 4 stellen, werden die fehlenden Stellen *vor* der Zahl mit Leerzeichen aufgefüllt. Beachten Sie, dass nach der letzten gestrichelten Linie eine leere Zeile (entspricht 2 `newline`) ausgegeben werden soll.

4.6 Bonus (1 Punkt)

Um den Bonuspunkt zu erhalten, müssen Sie ein Unterprogramm implementieren in der Datei `points.asm`, das die Punkte eines Spielzuges wie folgt berechnet:
Pro Verschmelzen zweier Steine gibt es $x \times 2^{v-1}$ Punkte, wobei x der Wert des neuen Steines und v die Anzahl der gleichzeitig verschmolzenen Steine ist. Zum Beispiel würde es beim Verschmelzen vier 2er Steine zu zwei 4er Steinen 16 Punkten geben. Falls gleichzeitig noch zwei weitere 2er Steine zu einem 4er Stein verschmolzen werden, also insgesamt sechs 2er Steine zu drei 4er Steinen, gäbe es insgesamt $3 \times 4 \times 2^2 = 48$ Punkte.

Parameter

1. `$a0` Adresse einer Reihung aller Felder des Spielfeldes

Das Unterprogramm gibt die berechnete Punktzahl zurück. Sie können davon ausgehen, dass das Spielfeld immer eine Größe von 4×4 hat. Beachten Sie, dass dem Unterprogramm das komplette Feld und nicht nur eine Zeile bzw. Spalte übergeben wird. Sie dürfen das übergebene Spielfeld nicht verändern.

5 Bewertung

Für jede der Teilaufgaben gibt es *public*, *daily* und *eval* Tests. Die *public* Tests stehen Ihnen lokal zur Verfügung. Im nächsten Abschnitt wird beschrieben, wie Sie diese ausführen können. Sie müssen für jeden Aufgabenteil alle *public* Tests bestehen um Punkte für diesen Aufgabenteil zu erhalten. Ein Aufgabenteil kann dabei aus mehreren Unteraufgaben bestehen, wie zum Beispiel bei Aufgabe 4.4.
Die *daily* Tests stehen Ihnen nicht lokal zur Verfügung. Sie werden jedesmal, wenn Sie Ihr Projekt auf unseren Server hochladen (`git push`) ausgeführt. Danach erhalten Sie per e-Mail und über das CMS Rückmeldung. Aufgrund einer begrenzten Serverkapazität, kann die Zeit bis Sie Rückmeldung erhalten variieren. Wir versuchen aber mindestens eine Rückmeldung pro Tag zu gewährleisten. Die *eval* Tests werden nach Beendigung des Projektes zur Bewertung des Projektes verwendet.

6 Tests selbst ausführen und debuggen

Verwenden Sie den Befehl `./run_tests.py` im Hauptverzeichnis des Projektes, um Ihre Implementierung zu testen. Mit dem Befehl `./run_test.py -t tests/pub/test_X` können Sie nur den test `test_X` ausführen.
Wenn ein Test, z.B. `tests/pub/test_X.asm`, fehlschlägt können Sie ihn in MARS debuggen.
Der Befehl `./build_testbox.py tests/pub/test_X.asm` 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.

Eigene Tests

Wir empfehlen Ihnen zusätzlich zu den mitgelieferten *public* Tests noch weitere eigene Tests zu schreiben. Diese können Sie im Ordner `tests/student` ablegen. Mit dem Befehl `./run_tests.py --student` können Sie Ihre eigenen Tests ausführen. Mit dem Befehl `./run_test.py -t tests/student/test_X` können Sie nur den test `test_X` ausführen.
Ein Test besteht aus zwei Dateien, `test_name.asm` enthält die Eingabe und `test_name.ref` die gewünschte Rückgabe/Ausgabe. Sie können die public Tests als Vorlage verwenden.

7 GUI

Zusätzlich zum Kommandozeileninterface, für welches Sie die `printboard` Funktion implementieren müssen, stellen wir Ihnen ein graphisches Interface zur Verfügung. Dieses können Sie mit dem Befehl `./run_gui.py` aufrufen. Steuern können Sie das Spiel dann über die Pfeiltasten.

8 Abgabe

Letzter Commit auf dem Master-Branch bis zum **07.05.2019, 23:59**.

Literatur

[1] `http://gabrielecirulli.github.io/2048/`

[2] `http://de.wikipedia.org/wiki/2048_%28Computerspiel%29`