Hinweise zur Bearbeitung und Abgabe

- Bitte nutzen Sie MARS zum Simulieren Ihrer Lösung. Stellen Sie sicher, dass Ihre Abgabe in MARS ausgeführt werden kann.
- Sie erhalten für jede Aufgabe eine separate Datei, die aus einem Vorgabe- und Lösungsabschnitt besteht. Ergänzen Sie bitte Ihren Namen und Ihre Matrikelnummer an der vorgegebenen Stelle. Bearbeiten Sie zur Lösung der Aufgabe nur den Lösungsteil unterhalb der Markierung:

```
#+ Loesungsabschnitt
#+ -----------------
```

- Ihre Lösung muss auch mit anderen Eingabewerten als den vorgegebenen funktionieren. Um Ihren Code mit anderen Eingaben zu testen, können Sie die Beispieldaten im Lösungsteil verändern.
- Bitte nehmen Sie keine Modifikationen am Vorgabeabschnitt vor und lassen Sie die vorgegebenen Markierungen (Zeilen beginnend mit #+) unverändert.
- Eine korrekte Lösung muss die bekannten **Registerkonventionen** einhalten. Häufig können trotz nicht eingehaltener Registerkonventionen korrekte Ergebnisse geliefert werden. In diesem Fall werden trotzdem Punkte abgezogen.
- Falls Sie in Ihrer Lösung zusätzliche Speicherbereiche für Daten nutzen möchten, verwenden Sie dafür bitte ausschließlich den **Stack** und keine statischen Daten in den Datensektionen (`.data`). Die Nutzung des Stacks ist gegebenenfalls notwendig, um die Registerkonventionen einzuhalten.
- Die zu implementierenden Funktionen müssen als Eingaben die Werte in den **Argument-Registern** (`$a0`–`$a3`) nutzen. Daten in den Datensektionen der Assemblerdatei dürfen nicht direkt mit deren Labels referenziert werden.
- Bitte gestalten Sie Ihren Assemblercode nachvollziehbar und verwenden Sie detaillierte Kommentare, um die Funktionsweise Ihres Assemblercodes darzulegen.
- Wir untersuchen alle Abgaben auf Plagiate. **Plagiate sind Täuschungsversuche und führen zur Bewertung „nicht bestanden“ für die gesamte Modulprüfung. Die aktuell bestehende Freiversuchsregelung an der TU Berlin greift in diesem Fall nicht.**
- Die Abgabe erfolgt über ISIS. Laden Sie die zwei Abgabedateien separat hoch.

Aufgabe 1: Nachbarfelder im Irrgarten (10 Punkte)

In dieser Hausaufgabe verwenden wir ein Array von $8 \times 8 = 64$ Feldern, um Irrgärten darzustellen. Die Elemente des Arrays sind vorzeichenlose Bytes. Aus den Feld-Koordinaten x und y lässt sich der Array-Index als

$$\text{Index} = 8 \cdot y + x$$

berechnen, wobei die x-Achse von links nach rechts und die y-Achse von oben nach unten verläuft. Die Feld-Koordinaten x und y können Werte zwischen 0 und 7 annehmen. Abbildung 1a illustriert dieses Schema.

Jedes Feld im Irrgarten hat bis zu vier *Nachbarfelder*. Diagonal angrenzende Felder werden nicht als Nachbarfeld betrachtet. Aufgepasst: Felder am Rand des Irrgarten haben nur 3 oder 2 Nachbarfelder.

Implementieren Sie die Funktion `neighbor`, welche ausgehend von einem Irrgarten-Index `pos` den Index des angrenzenden Felds in Richtung `direction` zurückgibt. Die Funktion `neighbor` soll den Wert -1 zurückgeben, falls kein Nachbarfeld in die angefragte Richtung existiert. Die folgende Tabelle zeigt die Kodierung des Arguments `direction` sowie beispielhaft einige erwartete Rückgabewerte:

Ausgangsfeld	**Nachbar oben** `direction` = 0	**Nachbar links** `direction` = 1	**Nachbar unten** `direction` = 2	**Nachbar rechts** `direction` = 3
`pos` = 0	-1	-1	8	1
`pos` = 1	-1	0	9	2
`pos` = 2	-1	1	10	3
...				
`pos` = 8	0	-1	16	9
`pos` = 9	1	8	17	10
...				
`pos` = 63	55	62	-1	-1

Signatur der zu implementierenden Funktion:

```
int  neighbor(  int pos,  int direction);
$v0               $a0          $a1
```

Sie dürfen annehmen, dass `neighbor` nur mit `pos`-Werten im Bereich 0–63 und `direction`-Werten im Bereich 0–3 aufgerufen wird. Die Implementierung der `neighbor`-Funktion mittels Lookup-Tabelle im Code ist nicht zulässig, siehe Aufgabe 2.

In der Assembler-Vorgabedatei wird die Funktion `neighbor` für alle Möglichkeiten von `pos` und `direction` einmal aufgerufen. Die Rückgabewerte werden als Tabelle ausgegeben.

Tipps:

- Die niederwertigsten drei Bits von `pos` kodieren die x-Koordinate, die darauf folgenden drei Bits die y-Koordinate.
- Assemblerbefehle für bitweise Logikoperationen (`and`, `or`, `xor`, ...) sowie für Schiebeoperationen (`sll`, `srl`, `sra`) sind hilfreich.

Aufgabe 2: Wegsuche im Irrgarten (10 Punkte)

Eine häufig zu findende Problemstellung in der Informatik ist die Suche nach einem kürzestmöglichsten Weg in einer zweidimensionalen Struktur. Beispiele sind Bewegungen von Spielfiguren in Computerspielen oder das automatische Verdrahten von elektronischen Komponenten auf Leiterplatten oder Chips. Der *Lee-Algorithmus* ist ein Lösungsansatz zu dieser Problemstellung. Um einen Weg zwischen den Feldern S und D im Irrgarten zu finden, geht dieser in zwei Schritten vor:

1. *Wellenausbreitung*: Das Startfeld wird mit der Entfernung 1 markiert. Ausgehend vom Startfeld S werden alle erreichbaren freien Felder mit der Entfernung zum Startfeld markiert.
2. *Rückverfolgung*: Ausgend vom Zielfeld D wird eine Folge von Feldern mit kontinuierlich sinkender Entfernung zum Startfeld gesucht und als Pfad markiert.

Ihre Aufgabe ist es, den zweiten Schritt der Rückverfolgung zu implementieren. **Die Wellenausbreitung wurde also bereits ausgeführt und ist nicht Teil der Aufgabe.**

Der Irrgarten wird, wie aus Aufgabe 1 bekannt, als Array von vorzeichenlosen Bytes (`unsigned char` in C) dargestellt. Der Wert 255 zeigt ein Hindernis an. Die Werte 1–99 werden verwendet, um freie Felder mit einer Distanz 1–99 zum Startfeld S zu kodieren. Das Startfeld hat dabei den Wert 1. Unerreichbare freie Felder haben den Wert 0. Um den Weg durch den Irrgarten zu markieren, sollen die Felder des Weges einschließlich Start- und Zielfeld mit dem Wert 254 markiert werden. Die Kodierung der Irrgarten-Felder ist in Tabelle 1 zusammengefasst.

Implementieren Sie die Funktion `trace_back`, welche im Irrgarten `maze` ausgehend vom Zielfeld `dest` den Weg zum Startfeld findet und markiert. Gehen Sie **iterativ** vor, also ohne rekursiven Aufruf. Verwenden Sie die Hilfsfunktion `neighbor`, welche in der Vorgabe definiert ist und sich wie die in Aufgabe 1 geforderte Funktion verhält[1].

Die Funktion soll die Länge des gefundenen Wegs zurückgeben, wobei Start- und Zielfeld mitgezählt werden sollen. (Erklärung: Falls Start- und Zielfeld identisch sind, soll 1 zurückgegeben werden. Falls Start- und Zielfeld direkt benachbart sind, soll 2 zurückgegeben werden.) Die erwarteten Rückgabewerte für die in Abbildung 1 dargestellten Beispiel-Irrgärten mit den dort markierten Start- und Zielfeldern sind 16 (`maze1`), 19 (`maze2`) und 14 (`maze3`).

Signatur der zu implementierenden Funktion:

`int trace_back( unsigned char *maze, int dest);`		
`$v0`	`$a0`	`$a1`

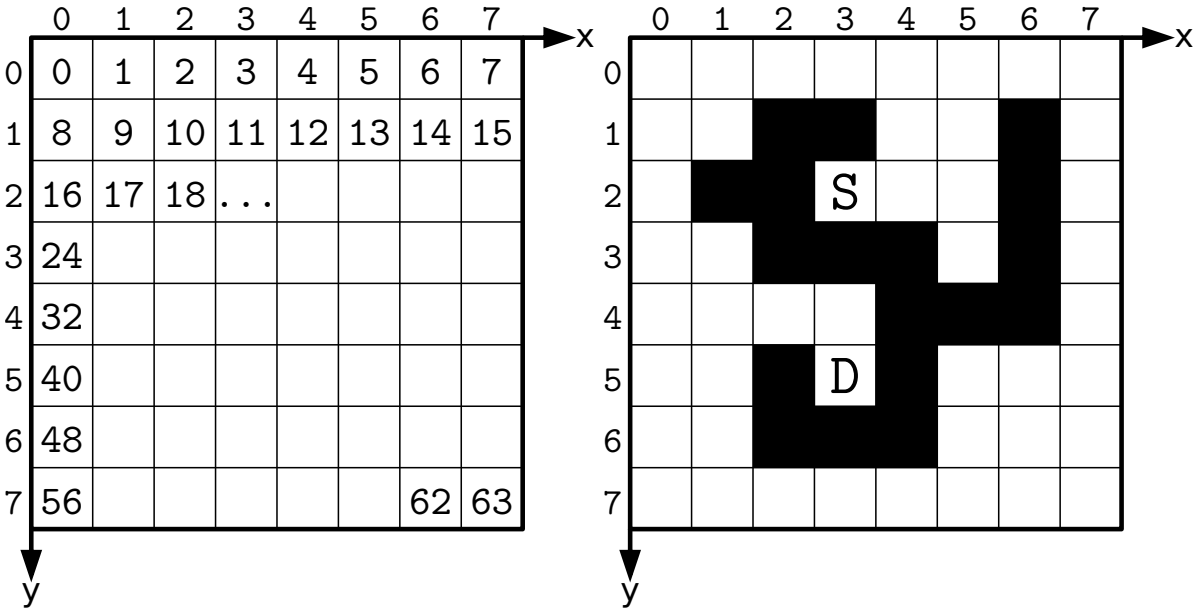
In der Assembler-Vorgabedatei wird die Funktion `trace_back` aufgerufen und danach der Irrgarten als Text ausgegeben. Hindernisse werden dabei mit einer Reihe von X-Zeichen dargestellt. Felder, die als Weg markiert wurden, werden mit einer Reihe von Plus-Zeichen dargestellt.

Sie dürfen annehmen, dass `trace_back` nur mit Irrgärten aufgerufen wird, welche wie oben beschrieben mit Entfernungswerten vormarkiert wurden, und dass es sich bei dem Feld `dest` um ein erreichbares freies Feld handelt.

Tipps:

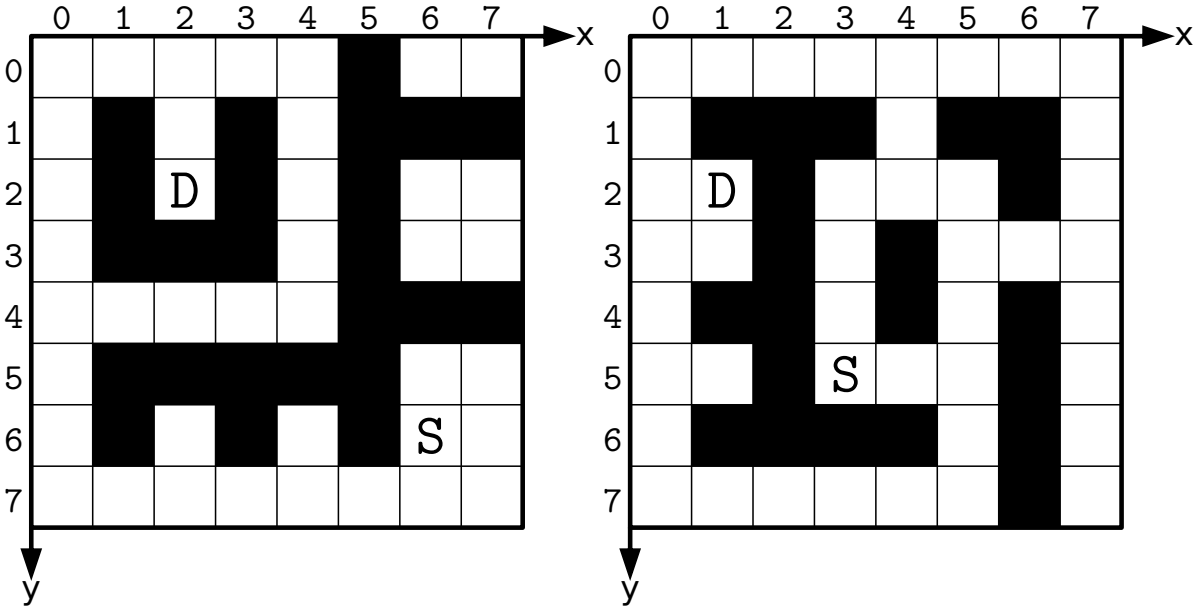
- Nutzen Sie den Befehl `lbu`, um Werte aus dem Array in Prozessorregister zu laden.
- Die Länge des zu findenden Weges (Rückgabewert) können Sie direkt im Zielfeld ablesen.
- Die Anzahl der zu markierenden Felder ist gleich der Länge des zu findenden Wegs (Rückgabewert).
- Wenn Sie das Programm mit einem anderen der Beispielirrgärten testen möchten, passen Sie die Werte von `test_maze_select` und `test_maze_destination` an.
- Das Funktionsargument `maze` ist ein Pointer auf das erste Element des Irrgarten-Arrays. Bei `dest` handelt es sich um den Array-Index.
- MARS gibt beim Assemblieren der Vorgabedatei die Warnung „0xFF out of range“ aus. Diese Warnung kann ignoriert werden.

[1] Die Funktion `neighbor` aus der Vorgabedatei ermittelt den Rückgabewert aus einer Lookup-Tabelle und ist daher keine zulässige Lösung für Aufgabe 1.



(a) Array-Indizes der Irrgarten-Felder

(b) Irrgarten 1 (`maze1`)



(c) Irrgarten 2 (`maze2`)

(d) Irrgarten 3 (`maze3`)

Abbildung 1: Beispiel-Irrgärten sowie Veranschaulichung der Array-Offsets. Die leeren Felder sind frei, die ausgefüllten Felder repräsentieren Hindernisse. *S* und *D* markieren die Start- und Zielfelder.

Wert	**Hindernis**	**Weg-Markierung**	**Abstands-Markierung**	**Darstellung**
0	nein	nein	nein	leer
1–99	nein	nein	ja, Wert = Abstand	als Zahl
254	nein	ja	nein	++
255	ja	nein	nein	XX

Tabelle 1: Kodierung der Felder im Irrgarten durch vorzeichenlose Bytes