Rechnerarchitektur

Praktikum 5: ACPS und bedingte Befehlsausführung (Game of Life)

Wintersemester 2024/25 (Dr. Mathias Schnee)

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Allgemeine Hinweise zum Praktikum:

- Bereiten Sie die Aufgaben unbedingt zu Hause oder in einem freien Labor vor. Das beinhaltet:
  - Entwurf der Lösung
  - Codieren der Lösung im SourceNavigator (virtuelle Maschine für RA)
- Die Zeit während des Praktikums dient dazu, die Lösung testieren zu lassen sowie eventuelle Korrekturen vorzunehmen.
- Das Praktikum dient auch zur Vorbereitung der Klausur am Ende des Semesters. Versuchen Sie also in Ihrem eigenen Interesse, die Aufgaben selbständig nur mit Verwendung Ihrer Unterlagen und ohne Codefragmente aus dem Netz zu lösen.
- Die Lösung wird nur dann testiert, wenn
  - sie erklärt werden kann bzw. Fragen zur Lösung beantwortet werden können.
  - das Programm ablauffähig und die Lösung nachvollziehbar ist.
  - die Hinweise oder Einschränkungen aus der Aufgabenstellung befolgt wurden.
- Zur Erinnerung hier noch einmal die Regeln des Praktikums, die schon in der Vorlesung besprochen wurden:
  - Sie arbeiten in 2er Gruppen.
  - Ein Testat gibt es nur zum jeweiligen Termin.
  - Abschreiben und Kopieren ist verboten.
  - Es gibt keine Noten. Die Bewertung ist lediglich erfolgreich / nicht erfolgreich.
  - Das Praktikum ist Zulassungsvoraussetzung für die Klausur. Hierfür müssen alle fünf Praktikumsübungen testiert sein.

Lernziele:

- ARM Procedure Calling Standard (APCS) verstehen und anwenden können
- In Assembler und einer Hochsprache geschriebene Routinen wechselseitig verwenden können
- Verstehen, wie Hochsprachen sich der Maschinensprache bedienen

Der britisch-amerikanische Mathematiker John Horton Conway (1937 – 2020) verstarb am 11. April 2020 an den Folgen einer COVID-19-Erkrankung. Conway ist insbesondere für sein „Game of Life" bekannt, einem Zellulären Automat der das Leben und Sterben von Organismen simuliert. Er hat zahlreiche Beiträge zur „Unterhaltungsmathematik" geliefert und ist Informatikern durch seine Arbeit bekannt.

Aufgabe 1

„Game of Life" ist ein zellulärer Automat, der das Leben und Sterben von Organismen auf sehr einfache Weise simuliert. Die Simulation läuft auf einem beliebig großen zweidimensionalen Array. Jedes Feld, hier Zelle genannt, hat 8 direkte Nachbarn. Eine Zelle ist entweder besetzt oder nicht besetzt oder anders ausgedrückt, an dieser Stelle lebt etwas oder eben nicht. Die Folgegeneration wird für alle Zellen gleichzeitig berechnet und ersetzt die aktuelle Generation.

Auf Wikipedia finden Sie eine gute Übersicht. Wir beschränken uns zunächst auf die 'Urfassung' (die 23/3-Welt) und betrachten die weiteren Varianten nicht. In der ursprünglichen Fassung von 1970 gelten folgende Regeln fürs Leben und Sterben (für besetzte Zellen):

- Eine Zelle mit weniger als 2 Nachbarn stirbt an Vereinsamung.
- Eine Zelle mit 2 oder 3 Nachbarn bleibt am Leben.
- Eine Zelle mit mehr als 3 Nachbarn stirbt an Überbevölkerung.

Es kann aber auch neues Leben entstehen (auf einer unbesetzten Stelle):

- Eine neue Zelle wird geboren, wenn sie genau 3 Nachbarn hat.

Die scheinbar einfachen Regeln ergeben vielfältige Formen und manche Konstellationen von Zellen haben interessante Eigenschaften: Sie ändern sich nicht, sie oszillieren, oder sie bewegen sich zyklisch verändernd fort. Es gibt Konstellationen, die immer wieder zyklisch 'Gleiter' abfeuern, man kann sogar logische Gatter (mit relativ komplizierten Gebilden) nachbilden.

Für Sie wurde eine **minimale Implementierung von Game of Life in C** bereits erstellt.

Es berechnet Nachfolgegenerationen auf einem endlichen Feld der Größe 50x20. Zellen am Rand haben also entsprechend weniger Nachbarzellen, und bei der Berechnung der Nachbarn muss man darauf achten, nicht außerhalb der Feldgrenzen zuzugreifen.

Wir nutzen das Programm, um uns mit dem APCS und der bedingten Befehlsausführung vertraut zu machen. Sie ersetzen zunächst die Berechnung der Nachbarn durch eine ARM-Assembler-Routine (Aufgabe 2). Anschließend ersetzen Sie die in C geschriebene Funktion zur Berechnung einer neuen Generation durch eine entsprechende Routine in ARM-Assembler (Aufgabe 3).

Zunächst machen Sie sich aber erst mal in Aufgabe 1 mit dem C-Code vertraut:

Wie sehen die Datenstrukturen aus, mit der eine neue Generation berechnet wird? Wie wird eine tote und eine lebende Zelle modelliert? Wieviel Speicher wird benutzt? Was macht `index()`? Warum ist die Berechnung der Nachbarn so kompliziert mit den vielen if-Bedingungen? Warum passiert die Berechnung der Nachfolgegeneration in 2 Phasen?

Aufgabe 2

Ersetzen Sie die in C geschriebene Funktion `neighborC()` durch eine Funktion `neighbor()` in ARM-Assembler.

Der Funktionsrahmen wurde für Sie wie bei den anderen Praktika bereits angelegt. (Sowohl `neighbor()` als auch `newGen()` sind in der Datei `newGen.S`.)

Beachten Sie unbedingt die Regeln des APCS: Wenn Sie ein Nicht-Scratch-Register wie etwa `R4` nutzen wollen, müssen Sie es zu Beginn der Funktion mit PUSH auf den Stack retten und vor Verlassen wieder mit POP vom Stack nehmen. Beachten Sie auch wegen des `LR`, ob sie eine Blatt-Funktion vorliegen haben oder nicht. Schauen Sie nochmal in die Vorlesungsfolien, wenn Sie das nicht parat haben.

Wenn Sie mehrere Vergleiche zu implementieren haben, nutzen Sie statt Sprüngen besser die bedingte Befehlsausführung. Überlegen Sie sich dazu bevor Sie mit dem Code schreiben beginnen, was folgende Assemblersequenz macht:

```
cmp   R0,#42
cmpne R1,#17
movne R2,R3
```

Prüfen Sie ob ihre Funktion korrekt implementiert ist, indem Sie in der C-Funktion `newGenC` ihre selbstgeschriebene Routine aufrufen. Passiert das gleiche in der Ausgabe?

Aufgabe 3

Ersetzen Sie als nächstes die Funktion `newGenC()` durch `newGen()`.

Um die erste Phase des Algorithmus zu prüfen, können Sie sich von C aus das Feld `n` analog zu `show()` ausgeben lassen (zur Not auch direkt von `show()`, dann sind eben alle Felder mit 0 Nachbarn als '.' dargestellt und alle Felder mit ≥ 1 Nachbarn als '*').

Das Feld `f` ist mit einigen einfachen Figuren vorbelegt: Oben links ist ein Gleiter – diese Figur sollte sich nach rechts unten bewegen, bis der Block darunter den Gleiter stoppt. Blöcke, 4 quadratisch angeordnete Zellen, bleiben stabil. Rechts neben dem Gleiter ist ein weiterer Block, und daneben ein Blinker. Blinker sind 3 in einer Linie angeordnete Zellen die von Generation zu Generation abwechselnd horizontal und vertikal angeordnet sind. Schließlich ist unten rechts ein R-pentomino, ein nur 5 Zellen großes Gebilde, dass auf einem unbeschränkten Feld 1103 Generationen leben würde. Es breitet sich bei uns fast über das ganze Feld aus.

Die Objekte wurden eingefügt damit Sie einfach durch Beobachtung der Ausgabe in C sehen können, ob die Regeln korrekt arbeiten. Sie sind nahe dem oberen Rand, damit sie beim Debuggen nicht so lange durchsteppen müssen. Natürlich können Sie `f` nach Bedarf frei ändern.

Aufgabe 4

Wenn Sie alles richtig gemacht haben und ACPS korrekt umgesetzt haben, dann vertragen sich Ihre Assemblerfunktionen einträchtig mit dem C-Code.

Prüfen Sie, ob das auch umgekehrt gilt, ob Sie aus dem Assembler heraus C-Funktionen, also `neighborC` statt `neighbor` aufrufen können. Das sollte eigentlich durch das Zufügen von nur einem Buchstaben 'C' in newGen.S gelingen...

Aufgabe 5

Gegeben ist ein direkt abbildender Cache.

In einer Cache-Line liegen 2 Wörter (mit je 4 Byte). Wie viele Bits werden für die Offsets verwendet?

ByteOffset _____ Bits, WordOffset _____ Bits, Insgesamt _____ Bits.

Mit welcher Bitfolge wird Byte 2 in Wort 1 adressiert? _____

Der Cache hat 4 Cachelines. Die Speicheradressen haben 8 Bit. Überlegen Sie sich, welche Bits für Tag und Index verwendet werden, vergessen Sie dabei die oben bestimmten Offset-Bits nicht!

Tag: _____

Index: _____

Tragen Sie in der folgenden Tabelle in der Spalte Treffer jeweils HIT oder MISS für den Lesezugriff ein. Die Reihenfolge der Lesezugriffe ist in der ersten Spalte angegeben. Geben Sie bei einem MISS den in diesem Schritt neu abgespeicherten Tag in der richtigen Cacheline an. Unveränderte Cachelines müssen Sie nicht erneut eintragen (außer für einen Treffer, siehe unten). Bei einem HIT markieren Sie in der entsprechenden Zeile die getroffene Cacheline zusätzlich, indem Sie den bereits gespeicherten Tag-Wert erneut eintragen und unterstreichen (vgl. vorausgefüllte Zeilen in den Tabellen).

		Tag			
Adresse	Treffer	Index 00	Index 01	Index 10	Index 11
0100 1010	MISS		010		
0100 1011	HIT		<u>010</u>		
1100 1101					
0101 1011					
0100 1010					
0101 1110					
1011 0001					

Warum entsteht ein Hit beim zweiten Zugriff trotz unterschiedlicher Adressen?

_____

_____