

10. Übungsblatt - zu bearbeiten bis 25.01.2021

Aufgabe 1 Assemblersprache

- | | | |
|--|---------------------|---------------------|
| a) Gegeben ist das nebenstehende Assembler-Programm in Hack-Assemblersprache, welches die Summe der Zahlen <code>first = 5</code> bis <code>last = 7</code> (inklusive) bildet und das Ergebnis an der Speicherstelle <code>res</code> ablegt. Es haben sich jedoch fünf Fehler eingeschlichen. Finden und korrigieren Sie diese Fehler! | <code>@7</code> | <code>@last</code> |
| | <code>D=A</code> | <code>D=D+M</code> |
| | <code>M=D</code> | <code>@END</code> |
| | <code>@first</code> | <code>D; JLT</code> |
| | <code>@5</code> | <code>@first</code> |
| | <code>D=A</code> | <code>D=M</code> |
| | <code>@last</code> | <code>@res</code> |
| | <code>M=D</code> | <code>M=D+M</code> |
| b) Geben Sie die durch einen Assembler erzeugte Symboltabelle für das in a) betrachtete Assembler-Programm an! | <code>@first</code> | <code>@first</code> |
| | <code>D=M</code> | <code>M=1</code> |
| | <code>@res</code> | <code>@LOOP</code> |
| | <code>M=0</code> | <code>0; JMP</code> |
| c) Nennen Sie zwei Anwendungsgebiete, in welchen Assemblersprache heutzutage noch verwendet wird! Warum wird sie in diesen Gebieten verwendet? | <code>(LOOP)</code> | <code>(END)</code> |
| | <code>@first</code> | <code>@END</code> |
| | <code>D=M</code> | <code>0; JMP</code> |

Zusatzaufgabe Assemblersprache: Multiplikation

Die Zusatzaufgabe auf Aufgabenblatt 8 beschäftigte sich mit der Implementierung der Multiplikation. Dort sollte der Standard-Multiplikationsalgorithmus so angepaßt werden, daß er nur die Funktionen der Hack-ALU nutzt. Als Lösung wurde dort das folgende Java-Programm vorgestellt (für eine Multiplikation 8 Bit \times 8 Bit \rightarrow 16 Bit):

```
static int multiply (int a, int b) {
    int i;                // Laufvariable
    int r = 0;            // Ergebnis
    int t = 1;            // als nächstes zu testendes Bit
    for (i = 8; --i >= 0; ) { // durchlaufe die Stellen des Faktors a
        if ((a & t) != 0)    // falls aktuelles Bit gesetzt (= 1),
            r = r + b;      // addiere (verschobenen) Faktor b zum Ergebnis
        t = t + t;          // verschiebe Testbit um eine Stelle nach links
        b = b + b;          // verschiebe Faktor b um eine Stelle nach links
    }
    return r;              // gib das berechnete Ergebnis zurück
}
```

Übersetzen Sie nun dieses Java-Programm in die Hack-Assemblersprache, so daß es auch auf dem Hack-Prozessor ausgeführt werden kann! Nehmen Sie an, daß das Argument `a` im virtuellen Register `R0` und das Argument `b` im virtuellen Register `R1` steht. Legen Sie das Ergebnis `r` im virtuellen Register `R2` ab! Prüfen Sie Ihre Implementierung mit Hilfe des Hack-CPU-Simulators!

Aufgabe 2 Virtuelle Maschine

Gegeben sei die nebenstehende Funktion `f` in der Sprache der virtuellen Maschine, die in der Vorlesung vorgestellt wurde. Skizzieren Sie den Ablauf des Programmes für den Fall, daß das einzige Argument der Funktion den Wert 5 hat.

- Wie verändert sich der Stapel mit jeder Operation?
- Was berechnet das Programm allgemein, wenn als Argument der Wert n gegeben ist?

Hinweise:

Um sich den Ablauf zu verdeutlichen, können Sie den Emulator der virtuellen Maschine benutzen, der unter

<http://www.nand2tetris.org/software.php>

zur Verfügung steht. Schreiben Sie dazu die rechts gezeigte Funktion in eine Textdatei `Main.vm`. Die zum Ausführen in dieser Datei zusätzlich benötigte Funktion `main` ist unten links gezeigt, den Inhalt der ebenfalls benötigten Datei `Sys.vm` finden Sie unten rechts.

```
function Main.main 1      function Sys.init 0
  push constant 5         call Main.main 0
  call Main.f 1           label WHILE
  pop local 0             goto WHILE
  return
```

```
function Main.f 2
  push constant 1
  pop local 1
  label LOOP
  push local 0
  push argument 0
  push local 1
  lt
  if-goto END
  push local 1
  push local 1
  push constant 1
  and
  if-goto ADD
  neg
  label ADD
  add
  pop local 0
  push local 1
  push constant 1
  add
  pop local 1
  goto LOOP
  label END
  return
```

Legen Sie beide Dateien in einem Verzeichnis ab und wählen Sie zum Ausführen des Programms dieses Verzeichnis im VM-Emulator aus!

Aufgabe 3 Virtuelle Maschine: Fibonacci-Folge

- Nennen Sie jeweils zwei Vor- und zwei Nachteile von virtuellen Maschinen!
- Die Fibonacci-Folge ist eine unendliche Folge von Zahlen, bei der sich die jeweils nächste Zahl durch Addition der beiden vorangehenden Zahlen ergibt:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... (siehe auch <https://oeis.org/A000045>).

Gegeben sei die folgende, in Java implementierte Methode `fib(int n)` zur rekursiven Berechnung der n -ten Fibonacci-Zahl (wobei die 0-te Fibonacci-Zahl, also die 0 am Anfang der Folge, vernachlässigt wird):

```
public static int fib (int n) {
    if (n <= 2)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

Implementieren Sie eine Funktion `function Main.fib` zur rekursiven Berechnung der n -ten Fibonacci Zahl unter Verwendung der Sprache der virtuellen Maschine des Hack-Systems. Die Funktion `Main.fib` erwartet dabei den Parameter `n`, der nach Aufruf der Funktion `Main.fib` in `argument[0]` vorliegt.

Hinweis: Vervollständigen Sie die Funktion `Main.fib` im folgenden Programmtext (Inhalt einer Textdatei `Main.vm`; wofür steht eigentlich das k ?):

<pre>function Main.main 1 push constant n call Main.fib 1 pop local 0 return</pre>	<pre>function Main.fib k ... Berechnung hier einfügen ... return</pre>
--	--

Zum Testen benötigen Sie außerdem die Datei `Sys.vm` aus Aufgabe 2.

Legen Sie beide Dateien (`Main.vm` und `Sys.vm`) in einem Verzeichnis ab und wählen Sie zum Ausführen des Programms dieses Verzeichnis im VM-Emulator aus!

Aufgabe 4 Virtuelle Maschine: Multiplikation

Schreiben Sie eine Funktion in der Sprache der virtuellen Maschine des Hack-Systems, die zwei Zahlen als Parameter hat und deren Produkt ausgibt! Fügen Sie ein Hauptprogramm hinzu, das diese Funktion aufruft! Sie können Ihr Programm mit Hilfe des VM-Emulators testen (siehe auch Aufgabe 2).

Syntaxhilfe zur virtuellen Maschine des Hack-Systems

Anweisung	Rückgabewert	Kommentar
<code>add</code>	$x + y$	Ganzzahladdition (Zweierkomplement)
<code>sub</code>	$x - y$	Ganzzahlsubtraktion (Zweierkomplement)
<code>neg</code>	$-y$	arithmetische Negation (Zweierkomplement)
<code>eq</code>	-1 falls $x = y$, sonst 0	Test auf Gleichheit
<code>gt</code>	-1 falls $x > y$, sonst 0	Test auf größer
<code>lt</code>	-1 falls $x < y$, sonst 0	Test auf kleiner
<code>and</code>	$x \& y$	bitweises Und
<code>or</code>	$x y$	bitweises Oder
<code>not</code>	$\sim y$	bitweise Negation

