Verstehen und Anwenden des IJVM-Befehlssatzes auf einer MAL Mikroarchitektur anhand ausgewählter Algorithmen mithilfe des Mic-1 MMV Simulators

Niklas Lampe – 5017616
Thade Struckhoff – 5016730

22. Juli 2021

Inhaltsverzeichnis

Einleitung

Dieser Laborversuch behandelt die Mikroarchitektur und die IJVM-Entwicklung mit dem Mic-1 MMV Simulator. Dieser steht Ihnen unter Aulis[1] zur Verfügung. Der Simulator ist komplett in Java geschrieben und verfügt über ein Mikroprogramm, dass die IJVM-Befehle aufgeführt in Tabelle 1 zur Verfügung stellt. Er ist interaktiv und visualisiert umfangreich die Mikroarchitektur-Simulation. Darüber hinaus wird die Programmentwicklung durch Einbindung von 'Compile and Load'-Assemblern für die Microcode Assembly Language (MAL) und die IJVM Assembly Language (JAS) unterstützt.

Hex	**Mnemonik**	**Bedeutung**
0x10	`BIPUSH` byte	Lege Byte auf den Keller
0x59	`DUP`	Kopiere oberstes Kellerwort und lege es auf den Keller
0xA7	`GOTO` offset	Unbedingter Sprung
0x60	`IADD`	Hole zwei Wörter vom Keller und lege ihre Summe auf den Keller
0x7E	`IAND`	Hole zwei Wörter, AND-verknüpfe sie, lege das Ergebnis auf den Keller
0x99	`IFEQ` label	Hole Wort vom Keller und verzweige, falls es null ist
0x9B	`IFLT` label	Hole Wort vom Keller und verzweige, falls es kleiner null ist
0x9F	`IF_ICMPEQ` label	Hole zwei Wörter vom Keller und verzweige, falls sie gleich sind
0x84	`IINC` varnum const	Addiere eine Konstante zu einer lokalen Variablen
0x15	`ILOAD` varnum	Lege lokale Variable auf den Keller
0xB6	`INVOKEVIRTUAL` offset	Rufe eine Methode auf
0xB0	`IOR`	Hole zwei Wörter, OR-verknüpfe sie, lege das Ergebnis auf den Keller
0xAC	`IRETURN`	Kehre aus Methode zurüuck
0x36	`ISTORE` varnum	Hole Wort vom Keller und speichere es in lokaler Variablen
0x64	`ISUB`	Hole zwei Wörter vom Keller und lege ihre Differenz auf den Keller
0x13	`LDC_W` index	Lege Konstante aus dem Konstantenpool auf den Keller
0x00	`NOP`	Nichts tun
0x57	`POP`	Lösche oberstes Kellerwort
0x5F	`SWAP`	Vertausche die beiden obersten Kellerwörter
0xC4	`WIDE`	Präfixbefehl; nächster Befehl hat einen 16-Bit-Index
0xFF	`HALT`	Den Simulator anhalten
0xFE	`ERR`	Text 'ERROR' drucken und Simulator anhalten
0xFD	`OUT`	Hole Wort vom Keller und gib die niederwertigen 8-Bits als ASCI-Zeichen zur Anzeige auf dem Bildschirm aus
0xFC	`IN`	Liest ein Zeichen von der Standardeingabe und setzt es in die niederwertigen 8-Bits eines Wortes, das auf den Stack gelegt wird

Tabelle 1: Standard IJVM-Befehlssatz des Mic-1 Simulators

[1] im Stammverzeichnis Fk4 TI COMARCH → Programme/Simulatoren → Simulator der Mic1-Mikroarchitektur

Falls noch nicht durch die Bearbeitung der Vorlesung erfolgt, machen Sie sich zur Vorbereitung auf den Laborversuch vertraut mit dem bereitgestellten IJVM-Befehlssatz. Dieser ist weitestgehend mit dem Ihnen aus dem Skript bekannten Befehlssatz gleich. Er verfügt lediglich über einige zusätzliche Befehle (`HALT`, `ERR`, `OUT` und `IN`) zur Steuerung des Simulators. Die Implementierung des zugehörigen Mikroprogramms ist in der Datei 'mic1ijvm.mal' enthalten und kann zur Beseitigung etwaiger Unklarheiten ebenfalls herangezogen werden.

Ziel des Versuchs ist es, den Simulator und seine Funktionalitäten kennen zu lernen sowie ein tieferes Verständnis über die Mikroarchitektur und den IJVM-Befehlssatz der Beispiel-ISA zu erlangen. Dazu werden Sie schrittweise an einzelne Themen herangeführt, die sukzessive die Basisfunktionalität und andere wichtige Aspekte aufzeigen. Daher empfiehlt es sich, die Aufgaben in der angegebenen Reihenfolge zu bearbeiten.

Aufgabe 1: Einführung

Für die erste Aufgabe soll sich zunächst mit der Umgebung des Mic-1 MMV-Simulators vertraut gemacht werden. Als Vorbereitung muss der Mic-1 Simulator entpackt und die JAR-Datei 'Mic1MMV.jar' im Ordner 'bin' gestartet werden.

1.1) Starten Sie den Simulator. Über das Menü 'File → Load IJVM program' wählen Sie nun das IJVM-Programm 'echo.ijvm' aus. Um das Programm auszuführen, wählen Sie in der Kommando-Konsole die Geschwindigkeit 'Prog' aus und starten es mithilfe des blauen Pfeiles.

Über die Eingabekonsole können Sie nun Eingaben tätigen, die von dem Programm direkt in die Ausgabe geschrieben werden. Nun wählen Sie die Geschwindigkeit 'IJVM' aus und aktivieren das Delay. Nach einem Reset können Sie nun mithilfe der blauen Pfeiltaste jeden IJVM-Befehl durchgehen. Dies lässt sich im Methodenbereich (Method Area) verfolgen. Der Delay zeigt in der Grafik den Fluss im Datenpfad an. Über den roten Pfeil kann analog eine Ebene wieder zurück gesprungen werden.

Als nächstes wird der Aufbau eines Mikroprogrammes bekannt gemacht.

1.2) Dieses Mikroprogramm führt die folgende Rechenoperation durch:

$$a + b + c - d - e$$

Stellen Sie eine Möglichkeit dder Entwicklung des Stacks dar, wenn die fünf Variablen bereits initialisiert sind.

1.3) Öffnen Sie nun die Datei 'rechenOperation.jas' in einem Editor Ihrer Wahl. Beschreiben Sie die Funktionen dieses Programmes in Stichpunkten. An welcher Stelle wird die Rechenoperation durchgeführt und welche Schritte sind noch in diesem Programm enthalten? Dabei müssen Sie auf die print-Methode nicht weiter eingehen, da diese nur ein Mittel zur Zweck ist, um ein Byte in der Ausgabekonsole als ASCII-Zeichen darzustellen.

Neben dem Methodenbereich gibt es in dem Simulator auch noch zwei weitere Speicheransichten, den Konstantenpool (Constant Pool) und den Kellerbereich (Stack Area). Alle verfügen über eine visuelle Verfolgung durch Hervorhebung der aktuellen Stelle, sofern dies in den Einstellungen (Preferences) nicht deaktiviert wurde.

1.4) Nun können Sie mithilfe des Simulators im Kellerbereich die Entwicklung des Stacks nachvollziehen. Öffnen Sie dazu über das Menü 'File → Assemble / Load JAS file' die Datei 'rechenoperation.jas'. Stellen Sie die Geschwindigkeit auf 'IJVM'. Wenn Sie nun durch die einzelnen Befehle durchgehen, können Sie die Entwicklung des Stacks verfolgen. Vergleichen Sie dies mit Ihrem Ergebnis aus der vorherigen Aufgabe. Stimmen diese überein? Wenn nicht, warum?

Aufgabe 2: Fehleranalyse

Einige Möglichkeiten zur Ablaufsteuerung haben Sie bereits in der vorherigen Aufgabe kennen gelernt. Die Verfolgung der Speicheransichten sowie die zielgerichtete Steuerung des Programmablaufs über die Kommando-Konsole sind insbesondere bei der Fehleranalyse und zu Debugging-Zwecken hilfreich. Der Simulator verfügt darüber hinaus noch über eine weitere Möglichkeit, nämlich das Setzen von Breakpoints. Damit ist es möglich, den Simulator an einer konkreten Stelle automatisch anzuhalten. Um solche Haltestellen festzulegen, muss jeweils die Speicherstelle des IJVM-Programms, zu sehen im Methodenbereich, identifiziert werden. Dann können diese Speicherstellen in das unterste Textfeld bei den Einstellungen[2] durch eine leertasten-separierte Auflistung angegeben werden. Nach einem Klick auf 'OK' sind die Breakpoints festgelegt, was auch an der jeweiligen blau hervorgehobenen Stelle im Methodenbereich visualisiert wird.

Auf der linken Fensterseite des Simulators befindet sich die Architekturansicht. Sie zeigt alle Mic-1 MMV Register und seine Datenpfade. Bei niedriger Ausführungsgeschwindigkeit (SubClock, Clock und IJVM) wird der Datenfluss des jeweiligen Datenpfads hervorgehoben und die Werte in den Registern können ggf. abgelesen und nachvollzogen werden. Des Weiteren bietet abschließend die Anweisungsansicht einen Blick auf die Mi-

[2]Menü: Preferences → Edit Prefernces

krosteuereinheit des Mic-1 MMV. Das Speicheradressregister (MPC) zeigt die aktuelle Mikrobefehlsadresse und stellt den Mikrobefehl symbolisch dar. Darunter zeigt das Speicherdatenregister (MIR) den aktuellen Mikrobefehl, aufgeteilt in das ausgefüllte Mikrobefehlsformat, an. Wenn eine IJVM-Anweisung interpretiert wird, wird dies ebenfalls angezeigt.

Nun haben Sie den Simulator grundlegend kennengelernt und sollen ihn zur praktischen Vertiefung anwenden. Bearbeiten sie dazu nachfolgende Aufgabenstellung:

2.1) Gegeben ist das IJVM-Programm 'ggT.jas'. Wie der Dateiname bereits vermuten lässt, handelt es sich um ein Programm zur Berechnung des größten gemeinsamen Teilers (ggT), in diesem Beispiel für zwei Eingangswerte. Zum besseren Verständnis wird der grobe Programmablauf erläutert. Der Einstiegspunkt des Programms ist die Methode 'main'. Sie ist eine umliegende Dauerschleife. Darin werden zunächst zwei hexadezimale Zahlen (jeweils als WORT) aus der Eingabekonsole eingelesen. Dies geschieht jeweils durch einen Druck der Enter-Taste oder sobald genau die Länge eines Wortes erreicht ist. Danach wird von diesen beiden Zahlen der ggT mit dem euklidischen Algorithmus[3] berechnet und in der Ausgabekonsole ausgegeben. Die Ausgabe erfolgt dabei im Format 'a: <Eingabe1>, b: <Eingabe2> - ggt: 0000<ggT>', wobei die spitzen Klammern als Platzhalter für einzugebende Werte bzw. des errechneten Ergebnisses stehen.

Sichten Sie zunächst das gegebene IJVM-Programm in einem Editor Ihrer Wahl und versuchen Sie es nachzuvollziehen. In dem Programm sind insgesamt fünf Fehler eingebaut. Finden und beheben Sie alle Fehler. Nutzen Sie dazu die durch den Simulator zur Verfügung stehenden Mittel und denken Sie über den implementierten Algorithmus nach. Der zuvor beschriebene Ablauf stellt in gewissem Maße den Soll-Zustand des Programms dar, denken Sie bei der Fehleranalyse auch da dran.

Abzugeben sind die fehlerfreie Datei 'ggT.jas' sowie das durch den Simulator assemblierte Programm 'ggT.ijvm'.

Aufgabe 3: Implementierung

In dieser Aufgabe sollen Sie Ihr erlerntes Wissen auf eine vorgegebene Problemstellung übertragen. Wenden Sie die kennen gelernten Möglichkeiten des Simulators an, um Ihr Programm ggf. einer Fehleranalsyse zu unterziehen. Die zu lösende Problemstellung ist ein gängiger Sortieralgorithmus, der durch ein IJVM-Programm umgesetzt werden soll. Das Listing 1 beschreibt eine Java-Implementierung des umzusetzenden Programms.

[3] Sollte Ihnen noch aus diversen Mathe-Vorlesungen bekannt sein.

3.1) Sichten Sie den Programmcode des vorgegebenen Algorithmus in Listing 1. Um welchen Sortieralgorithmus handelt es sich dabei?

Für die Aufgabe wird Ihnen die Datei 'sortieralgorithmus.jas' bereitgestellt. Diese soll als Vorlage für Ihre Implementierung dienen. Sie enthält ein vorgegebenes Grundgerüst, in dem zu Beginn fünf Werte über die Eingabe-Konsole eingelesen und am Ende wieder ausgegeben werden sollen. Das Eingabe-Array ist durch fünf lokale Variablen 'a0' bis 'a4' abgebildet. Die Ausgabe ist so aufgebaut, dass der kleinste Wert, repräsentiert durch 'a0', und aufsteigend bis 'a4', der größte Wert, ausgegeben wird. Dabei werden die Ihnen bereits bekannten Methoden 'getnum' und 'print' verwendet. Nachdem der Programmablauf durchlaufen ist, beginnt er wieder von vorne.

3.2) Implementieren Sie den identifizierten Sortieralgorithmus in die vorgegebene Schablone. Nutzen Sie dazu wieder den IJVM-Befehlssatz aus Tabelle 1 und wenden Sie ggf. bei einer Fehleranalyse die kennengelernten Debug-Möglichkeiten des Simulators an.

3.3) Führen Sie eine Qualitätssicherung durch, in der Sie Ihre Implementierung durch geeignete Testfolgen verifizieren.

```
1  public void Sort(int[] data) {
2      for (int i = 0; i < data.length-1; i++){
3          int currMinIndex = i;
4          int tmp;
5          for(int j = i + 1; j < data.length; j++){
6              if (data[currMinIndex] > data[j]){
7                  currMinIndex = j;
8              }
9          }
10         tmp = data[i];
11         data[i] = data[currMinIndex];
12         data[currMinIndex] = tmp;
13     }
14 }
```

Listing 1: Sortieralgorithmus