Karlsruher Institut für Technologie
Institut für Technische Informatik (ITEC)

Rechnerorganisation im WS 2020/21

Musterlösungen zum 3. Übungsblatt

Prof. Dr. Jörg Henkel
Dr.-Ing. Lars Bauer
Roman Lehmann, M. Sc.
Haid-und-Neu-Str. 7,
Geb. 07.21 (Technologiefabrik)
Email: roman.lehmann@kit.edu

Lösung 1 (10 Punkte)

Siehe `caesar_musterloesung.c` im Ilias.

Lösung 2 (8 Punkte)

1. Die erwünschte Ausgabe lautet: `17541.2330000000` 3 P.
   Die wirkliche Ausgabe lautet: `17541.2324218750`
   (x64 Prozessor, gcc, Ubuntu Linux 18.04).
   Das Problem bei der Ausgabe stellt die Zahl selbst dar, die nur angenähert und nicht exakt in einer Fließkommazahl gespeichert werden kann.

2. Das Programm erzeugt nicht die Ausgabe `x = 5`, sondern je nach System `x = 25` oder es produziert einen Absturz. Wie der Compiler schon beim übersetzen warnt wird mit `return &i` die Adresse einer Variablen zurückgegeben, die nur innerhalb der Funktion definiert ist. Dies führt dazu, dass der Inhalt der Speicherstelle auf die `&i` verweist nach dem Funktionsaufruf nicht mehr gültig ist. 2 P.
   Variablen wie die Interger-Zahl `i` werden auf dem Stack angelegt und verschwinden nach dem Aufruf der Funktion wieder. Nach einem Funktionsaufruf muss allerdings der Bereich auf dem Stack nicht sofort freigegeben werden.

3. Off-by-One-Fehler: Die Schleife wird auch für `i=0` durchlaufen. Beim letzten Durchlaufen der Schleife führt dies zu einem Unterlauf von `i` und es erfolgt ein Zugriff auf das Element `0xFFFFFFFF`. Dies führt zu einer Speicherschutzverletzung (Segmentation Fault), da diese Adresse nicht im adressierbaren Speicherbereich des Programms liegt. Der Fehler kann ganz einfach korrigiert werden, indem der Vergleich `i >= 0` durch den Vergleich `i > 0` ersetzt wird. 3 P.

Lösung 3 (3 Punkte)

```
1. - 6. Hol- und Ausfuehrungsphase wie bekannt
7. Akku -> X, Akku -> Y
8. ALU: Addition (c2 = 0, c1 = 0, c0 = 1)
9. Z -> Y
10. ALU: Addition (c2 = 0, c1 = 0, c0 = 1)
11. Z -> Akku
```

Lösung 4 (8 Punkte)

1. Konstanten werden geladen, indem der Inhalt des Instruktionsregister in das Akku-Register kopiert wird. Beide Register haben eine Länge von 24 Bit (wie auch der interne Datenbus). Bei einem anderen Opcode als $0x0$ könnten allerdings Konstanten wie 0, 1, 2, 3 nicht mehr geladen werden, da diese mit im Binärformat mit `0000` beginnen. 2 P.

2. Das Befehlsformat der MIMA sieht die Verwendung der vier höchstwertigen Bit als Opcode vor. Sind die vier höchstwertigen Bit alle gesetzt ($0xF$), so können über die vier nachfolgenden Bit weitere 16 Maschinenbefehle unterschieden werden. Insgesamt sind mit diesem Befehlsformat $15 + 16 = 31$ Maschinenbefehle realisierbar, von denen allerdings nur 13 Befehle implementiert sind. Nicht belegte Opcodes sind ungültig und versetzen die MIMA in einen Ausnahmezustand. 2 P.

3. Registernamen und Bedeutung: 2 P.

   - `IAR` (InstruktionsAdressRegister):
     Speichert die Adresse des aktuell auszuführenden Befehlsworts zu Beginn der Lese-Phase *(fetch phase)*. Nach der Lese-Phase enthält es die Adresse des im nächsten Zyklus auszuführenden Maschinenbefehls.
   - `IR` (InstruktionsRegister):
     Speichert das aktuell auszuführende Befehlswort.
   - `SAR` (SpeicherAdressRegister):
     Enthält die Speicheradresse auf die bei der nächsten Lese- ($R = 1$)/Schreibanfrage ($W = 1$) zugegriffen wird.
   - `SDR` (SpeicherDatenRegister):
     Enthält bei einer Schreiboperation ($W = 1$) das zu schreibenden Datenwort.
     Nach einer Leseoperation ($R = 1$) enthält es das gelesene Datenwort.

4. Das Speicheradress- (`SAR`) und das Speicherdatenregister (`SDR`) werden nur für Befehle benötigt, die auf den Speicher zugreifen. Daher werden sie zum Beispiel für den Befehl `LDC` oder `JMN` nicht benötigt. Das InstruktionsAdressRegister (`IAR`) wird beispielsweise für den `LDC`-Befehl nicht benötigt. 2 P.