Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik VI, Technische Informatik

Prof. Dr. Joachim K. Anlauf

12. Übung für die Vorlesung

Technische Informatik

Wintersemester 2022/2023

Abgabe: spätestens Dienstag, 31.1.2023, 8:15 Uhr

Hinweis: Die Bearbeitung der Aufgaben 4 und 5 ist freiwillig und nicht mehr relevant für die Zulassung zur Prüfung. Alle Inhalte dieses Zettels werden aber besprochen und sind klausurrelevant!

Aufgabe 1. *Assembler* 7 P.

Gegeben ist die in der Vorlesung besprochene Rechnerarchitektur und der dazugehörige Assembler-Befehlssatz.

1. Schreiben Sie ein möglichst kurzes Assemblerprogramm, das die Funktion $min(a, b)$ für zwei positive Ganzzahlen a und b berechnet. Gehen Sie davon aus, dass sich die Eingaben an den Speicheradressen 14 und 15 befinden. Das Ergebnis der Berechnung soll nach der Beendigung des Programms an der Speicheradresse 13 vorliegen. Verwenden Sie Labels, um sich die Notation der Sprünge zu vereinfachen.

2. Geben Sie den vollständigen Speicherzustand des Rechners nach der Berechnung von $min(5, 7)$ an. Übertragen Sie hierzu das von Ihnen geschriebene Assemblerprogramm in das Binärformat und tragen Sie es zusammen mit dem vollständigen Speicherzustand in eine Tabelle ein. Sie können nicht verwendete Speicherstellen mit einem Strich ("-") markierern.

Aufgabe 2. *Assembler* 5 P.

Betrachten Sie das folgende Programmgerüst:

```
0000 LDA 0011
0001 STA 1111
0010 JMP func:
0011 ...
.... ...
0110 LDA 1001
0111 STA 1111
1000 JMP func:
1001 ...
.... ...
1100 func: ...
.... ...
1111 exit: JMP 0000
```

1. Welches bekannte Programmierkonzept wird in dem obigen Programm umgesetzt?

2. Wie könnte der Modellrechner erweitert werden, um solche Programme besser zu unterstützen?

Aufgabe 3. *Assembler* 12 P.

Gegeben sei die aus der Vorlesung bekannte CPU-Architektur, dessen Registerbreite auf 8-Bit erweitert wurde.

1. Mit der Erweiterung ist unser Prozessor nun in der Lage mit einem größeren RAM umzugehen. Wieviele Speicherstellen kann die CPU jetzt maximal adressieren?

2. Schreiben Sie ein Assember-Programm, das die Summe der Elemente eines Vektors berechnet.

   - **Eingabe:** Vektor $V = (v_n, ..., v_1)$ mit $1 \leq n \leq 32$. Die Elemente v_i sind in aufeinanderfolgenden Speicherstellen 32 bis 63 in aufsteigender Reichenfolge, nach i sortiert, gespeichert. Die Größe des Vektors n befindet sich in Speicherstelle 31.
   - **Ausgabe:** Das Ergebnis $SUM_V = \sum_1^n v_i$ soll in Speicherstelle 32 gespeichert werden.

   (Hinweis: Vielleicht hilft Ihnen bei der Lösung dieser Aufgabe die Idee aus der vorherigen Aufgabe.)

3. Führen Sie ihr Programm mit der Eingabe $V = (13, 7, 11, 31, 3)$ aus und geben Sie nach jedem `STA`-Befehl den Zustand der veränderten Speicherstelle an.
   (Hinweis: Vergessen Sie dabei nicht, dass eine Speicherstelle eine Instruktion und einen Operand enthält.)

Aufgabe 4. *Cache* 0 P.

Betrachten Sie einen Direct Mapped Cache mit 64 Cache-Zeilen für einen Prozessor, der 2 Byte breite Adressen, 1 Byte breite Datenworte und eine Blockgröße von einem Byte verwendet. Dabei soll die Write-Through Strategie verwendet werden, sodass ein Dirty-Bit nicht notwendig ist.

1. Welche Bits der Adresse werden für die Adressierung der Cache-Zeile bzw. die Tag-Bits verwendet?

2. Zeichnen Sie das komplette Blockschaltbild des Cache, sodass die Leseoperation realisiert wird. Schreiben Sie an jede markierte Leitung die Anzahl der verwendeten Bits.

3. Wie viele Bits Speicherplatz werden insgesamt für den Cache benötigt?

Aufgabe 5. *Cache* 0 P.

Ein 8-Bit Computer (einer mit 8 Bit Adressen) greift nacheinander auf folgende Adressen zu (dezimal):
0, 1, 2, 0, 9, 17, 18, 13, 0, 17, 2

1. Jeder Zugriff auf den Hauptspeicher benötige 5 Takte. Wie viele Takte werden für obige Speicherzugriffe benötigt?

2. Nun soll der Computer einen kleinen Direct-Mapped Cache mit 8 Cachezeilen bekommen. Bei jedem Hit benötigt der Speicherzugriff nur noch einen Takt, bei einem Miss sind es immer noch 5 Takte.

   Tragen Sie den Zustand des Caches nach Abarbeitung der Speicherzugriffe ein. Vergessen Sie nicht das Valid-Bit und die Tagbits. Tragen Sie ebenfalls die Hauptspeicheradressen ein (in einem richtigen Cache ständen hier die Daten). Streichen Sie bei Bedarf überschriebene Tags und Adressen durch.

   Tragen Sie gleichzeitig die Adressen in der Reihenfolge ihres Auftretens in die zweite Tabelle ein und vermerken Sie bei jedem Zugriff, ob es sich um einen Hit oder einen Miss handelt.

Zeile	Valid	Tag	Hauptspeicheradresse
0			
1			
2			
3			
4			
5			
6			
7			

Adresse	Hit

3. Wie viele Takte benötigt der Computer nun bei derselben Abfolge von Adressen?