64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 12 Ausgabe: 15.01., Abgabe: 22.01. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 12.1 (Punkte 6·5)

x86-Adressierung: Angenommen, die folgenden Werte sind in den angegebenen Registern bzw. Speicheradressen gespeichert. Adressen sind verkürzt dargestellt und von den Werten sind jeweils auch nur die unteren 32-bit angegeben:

Register	Wert
`%rax`	`0x00000100`
`%rcx`	`0x00000002`
`%rdx`	`0x00000018`

Adresse	Wert
`0x100`	`0x0000abba`
`0x108`	`0x000000dc`
`0x110`	`0x000000ef`
`0x118`	`0x00054321`

Überlegen Sie sich, welche Speicheradressen bzw. Register als Ziel der folgenden Befehle ausgewählt werden und welche Resultatwerte sich aus den Befehlen ergeben:

Befehl	Ziel (Adresse/Register)	Wert
`addq %rcx, (%rax)`		
`subq %rdx, 8(%rax)`		
`imulq $16, %rdx`		
`incq 16(%rax)`		
`decq %rcx`		
`subq %rdx, %rax`		

Zur Erinnerung: für den gnu-Assembler gilt

- der Zieloperand steht rechts
- Registerzugriffe werden direkt ausgedrückt
- eine runde Klammer um ein Register bedeutet einen Speicherzugriff, ggf. mit Immediate-Offset und Index: $\langle imm \rangle$`(`$\langle Rb \rangle$`,`$\langle Ri \rangle$`,`$\langle s \rangle$`)` → `MEM[`$\langle Rb \rangle$`+`$\langle s \rangle$`*`$\langle Ri \rangle$`+`$\langle imm \rangle$`]`

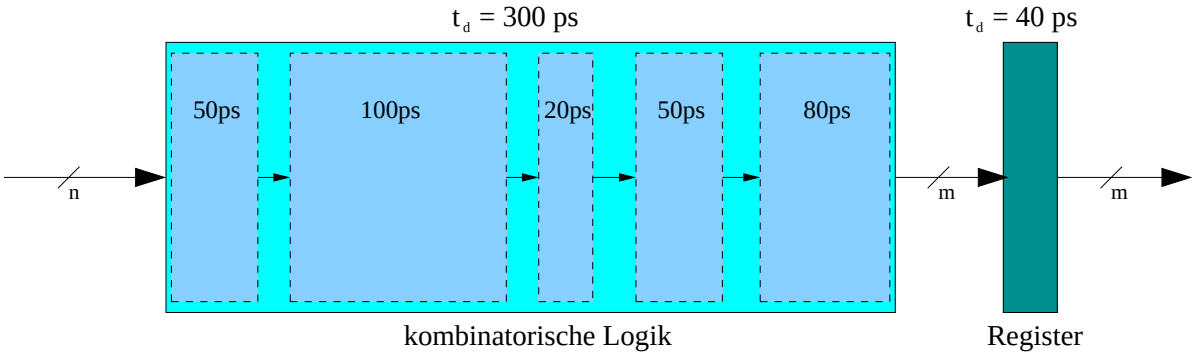
⇒ Beispiel: Befehl `addq %rcx, 24(%rax)`
Operation `MEM[0x00000118] := MEM[0x00000118]+2 = 0x00054323`

Aufgabe 12.2 (Punkte 5·5)

x86-Prozeduraufrufe und Stack: Der Stack ist für die Ausführung von Funktionen, bzw. Prozeduren von zentraler Bedeutung. Dabei werden auf dem Stack verschiedene Daten gespeichert: sowohl zur Vorbereitung des Unterprogrammaufrufs, als auch während dessen Abarbeitung.

Prinzipiell können dabei bis zu 5 „Arten“ von Daten auf dem Stack abgelegt werden. Welche Daten sind das? (Aufzählung mit kurzer Beschreibung).
Tipp: Überlegen Sie sich den Ablauf eines Unterprogrammaufrufs.

Aufgabe 12.3 (Punkte 5+5+5)



Pipelining: Gegeben sei die oben gezeigte Funktionseinheit. Die Zeitangabe $t_d = 300\,\text{ps}$ sagt aus, dass die Reaktion auf einen Signalwechsel am Eingang des kombinatorischen Logikblockes nach 300 ps am Ausgang erscheint (= Verzögerungszeit der Funktion). Für das Register soll der Einfachheit halber lediglich eine Zeitbedingung eingeführt werden: die Zeitdauer zwischen Taktvorderflanke und Ausgabe des neuen Werts am Datenausgang sei $t_d = 40\,\text{ps}$.

(a) Geben Sie die Latenzzeit und den Durchsatz der Schaltung an.

(b) Der kombinatorische Logikblock der obigen Schaltung lässt sich, an den angegebenen Stellen in kleinere Teilfunktionen aufteilen, wobei die Reihenfolge beizubehalten ist. Zeigen Sie, wie durch Einführung von Pipelineregistern (mit gleichen Zeitbedingungen wie das vorhandene Register) der Durchsatz maximiert werden kann.

Wir erlauben zunächst nur ein einziges Pipelineregister, also eine zweistufige Pipeline. An welcher Stelle muss das Pipelineregister für maximalen Durchsatz platziert werden? Geben Sie auch für diese Konfiguration Latenzzeit und Durchsatz an.

(c) Welche Werte ergeben sich, wenn an allen vier möglichen Stellen Pipelineregister eingefügt werden?

Aufgabe 12.4 (Punkte 10+20 [+10 Bonus])

Speicher-Gebirge („Memory Mountain"): Die Performanz eines Rechners wird neben Taktfrequenz und Aufbau des Prozessors (also seiner Pipeline und den vorhandenen Rechenwerken) vor allem von der Speicherhierarchie mit den Befehls- und Daten-Caches und dem Hauptspeicher geprägt.

Im Lehrbuch von Bryant und O'Hallaron wird zur Analyse das C-Programm `mountain` beschrieben (siehe Abschnitt 6.6.1). Es misst die Ausführungszeit von Unterprogrammen abhängig von der Größe der beteiligten Daten und den Zugriffsmustern auf ein eindimensionales Array. Abhängig von den beiden Parametern Größe (*Working-Set*) und Abstand benachbarter Daten (*Stride*) wird die Leserate in MByte/s gemessen.

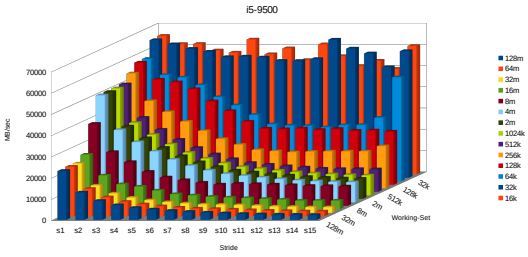
Unter `http://csappbook.blogspot.com/2017/05/a-gallery-of-memory-mountains.html` ist die Idee zu dem Experiment erklärt. Je nachdem aus welcher Quelle die Daten gelesen werden (Hauptspeicher, Cache-Speicher L1…L3) sind die Datentransferraten unterschiedlich hoch und im Idealfall sieht man deutliche Plateaus in dem 3-D Plot.

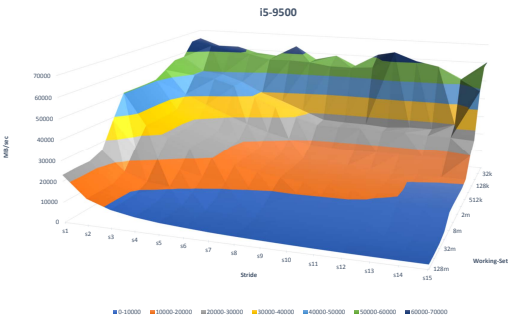
(a) Laden Sie das Programm, bzw. dessen Quelldateien herunter[1] und entpacken Sie die Datei `mountain.tar`.[2] Das Programm ist auf Linux-Rechnern (z.B. in den RZ-Poolräumen) direkt lauffähig. Sie können es aber auch selber aus den Quellen neu übersetzen.[3]

Führen Sie das Programm, möglichst auf mehreren verschiedenen Rechnern, aus und speichern Sie die Ausgabe in Datei(en) ab.[4] Was erhalten Sie als maximale Leserate in MByte/s Ihres Rechners, und welche Leserate bleibt davon für große Datenmengen noch übrig?

(b) Versuchen Sie, die Daten als Gebirge zu plotten (z.B. Import in Excel oder LibreOffice Calc und Erzeugen eines Diagramms).

Geben Sie dazu die Ausgabe von `mountain` und den zugehörigen Plot ab. Beispiele: s.u.





[c] Wenn Sie das Programm `mountain` mehrfach laufen lassen, können sich deutliche Unterschiede bei den gemessenen Werten ergeben. Wie lässt sich das erklären? Welche weiteren Faktoren spielen da zusätzlich noch eine Rolle?

[1] Lokal von `tams.informatik.uni-hamburg.de/lectures/2019ws/vorlesung/rsb/uebung` oder der Webseite zu dem Lehrbuch unter `csapp.cs.cmu.edu/3e/students.html` (Chapter 6).

[2] `tar xf mountain.tar`

[3] `make`

[4] `./mountain > my-mountain.out`