ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG
INSTITUT FÜR INFORMATIK

Dr. Jürgen Ruf
Dipl.-Inf. Ilia Polian

Freiburg, 26. Juni 2002

5. Übungsblatt zur Vorlesung

Rechnerarchitektur

Aufgabe 1

Für diese Aufgabe wird im Folgenden eine vereinfachte RISC-Maschine definiert. Sie verfüge über die Register R0 – R31, wobei das Register R0 mit der Konstanten 0 belegt ist. Die Maschine besitze außerdem ein Statusregister SR und den Befehlszähler PC. Die einzelnen Befehle werden in einer fünfstufigen Befehls-Pipeline verarbeitet, wie sie in der Vorlesung vorgestellt wurde. (Befehl-Holen, Befehl-Dekodieren, Operation-Ausführen, Speicherzugriff und Resultat-Speichern). Sprungbefehle werten die Bedingung in der Ausführphase aus und ändern gegebenenfalls den Befehlszähler in der Speicherzugriffsphase. In den folgenden Tabellen sind die Befehle des Prozessors angegeben. Die Semantik wird durch Pseudo-C-Code dargestellt. Dabei bezeichnet Ri ein Register, Addr eine Adresse, Val eine Konstante und Cond eine Bedingung.

Arithmetische Befehle		Vergleichsbefehl	
`ADD Ri, Rj, Rk`	Rk = Ri + Rj	`CMP Ri, Rj`	SR = Ri - Rj
`SUB Ri, Rj, Rk`	Rk = Ri - Rj	Sprungbefehl	
`MUL Ri, Rj, Rk`	Rk = Ri * Rj	`JMP Cond, Addr`	if(Cond) PC = Addr
`OR Ri, Rj, Rk`	Rk = Ri — Rj	Bedingungen für JMP	
`AND Ri, Rj, Rk`	Rk = Ri & Rj	T	true
`NOT Ri, Rj`	Rj = Ri	F	false
Ladebefehle		LT	SR < 0
`LDC Val, Ri`	Ri = Val	LE	SR <= 0
`LOAD Ri, Rj`	Rj = *Ri	GT	SR > 0
`LOAD Addr, Ri`	Ri = *Addr	GE	SR >= 0
`STORE Ri, Rj`	*Rj = Ri	Z	SR == 0
`STORE Ri, Addr`	*Addr = Ri	NZ	SR ! = 0

Auf dem oben angegeben Prozessor soll ein Programm ausgeführt werden. Die folgende Codesequenz enthält noch Pipelinekonflikte, welche die in den Kommentaren angedeutete Arbeitsweise der einzelnen Befehle beeinflussen. Die Adressen a, b und c zeigen jeweils auf einen Speicherbereich mit 10 Vier-Byte-Werten.

```
func:    LDC 1, R1         ; R1 = 1
         LDC 4, R2         ; R2 = 4
         LDC 10, R3        ; R3 = 10
         LDC a, R4         ; R4 = a
         LDC b, R5         ; R5 = b
         LDC c, R6         ; R6 = c
label1:  LOAD R5, R7       ; R7 = *R5
         LOAD R6, R8       ; R8 = *R6
         SUB R7, R8, R9    ; R9 = R7 - R8
         CMP R9, R0        ; SR = R9 - 0
         JMP LE, label2    ; if (SR <= 0) goto label2
         STORE R7, R4      ; *R4 = R7
         JMP T, label3     ; goto label3
label2:  STORE R8, R4      ; *R4 = R8
label3:  SUB R3, R1, R3    ; R3 = R3 - 1
         CMP R3, R0        ; SR = R3 -0
         JMP LE, end       ; if (SR <= 0) goto end
         ADD R4, R2, R4    ; R4 = R4 + 4
         ADD R5, R2, R5    ; R5 = R5 + 4
         ADD R6, R2, R6    ; R6 = R6 + 4
         JMP T, label1     ; goto label1
end:     ...
```

a) Erläutern Sie die Funktion des Programms (gemäss den Kommentaren).

b) Nennen Sie alle möglichen Abhängigkeiten zwischen Befehlen. Überprüfen Sie, welche bei der obigen Pipeline zu Konflikten führen können. Zwischen welchen Stufen der Pipeline können diese Konflikte auftreten?

c) Finden Sie eine Assembler-Anweisung, mit der ein NOP-Befehl realisiert werden kann, und lösen Sie alle Pipelinekonflikte durch das Einfügen einer minimalen Anzahl von NOP-Befehlen auf.

d) Optimieren Sie dieses mit NOPs ergänzte Programm weiter, indem Sie durch Verschiebung von Anweisungen NOP-Befehle einsparen.

e) Die Pipeline soll nun Forwarding unterstützen. An welchen Stellen kann die Ausführungszeit damit verkürzt werden? Beziehen Sie sich auf die Lösung aus Aufgabenteil (c).

Aufgabe 2

In dieser Aufgabe geht es um den Vergleich zweier in der Vorlesung vorgestellten 2-Bit-Prädiktoren. Betrachten Sie den 2-Bit sättigenden Zähler und den 2-Bit trägen Automaten auf der Folie 49 des Kapitels 3.4. Geben Sie eine Sequenz von Sprungentscheidungen (Taken / Not Taken), so dass diese zwei Prädiktoren zwei unterschiedliche Vorhersagen produzieren. **Hinweis:** Der Startzustand der Automaten ist jeweils unbekannt.

Aufgabe 3

Gegeben sei untenstehendes Programmfragment. Die Semantik wird durch Pseudo-C-Code dargestellt.

Hinweis: In `R0` ist die Konstante 0 fest gespeichert.

```
        LDC 1,  R1      ; const1 = 1
        LDC 2,  R4      ; counter1 = 2
loop1:
        LDC 3,  R3      ; counter2 = 3
        LDC 2,  R2      ; var1 = 2
loop2:
        CMP R2, R0      ; SR = var1 - 0
        JMP Z,  label   ; if (SR == 0) goto label (Sprung 1)
        SUB R2, R1, R2  ; R2 = R2 - 1
label:
        SUB R3, R1, R3  ; R3 = R3 - 1
        CMP R3, R0      ; SR = R3 - 0
        JMP GT, loop2   ; if (SR > 0) goto loop2 (Sprung 2)
        SUB R4, R1, R4  ; R4 = R4 - 1
        CMP R4, R0      ; SR = R4 - 0
        JMP GT, loop1   ; if (SR > 0) goto loop1 (Sprung 3)
```

a) Ermitteln Sie den Verlauf der Sprünge 1 bis 3.

b) Verwenden Sie jeweils einen 1-Bit-Prädiktor (0,1) zur Vorhersage jedes Sprunges. Diese seien wie folgt initialisiert: Sprung 1 (NT); Sprung 2 (T), Sprung 3 (T). Übernehmen Sie zunächst den Verlauf der Sprünge aus Aufgabenteil 1. Tragen Sie weiterhin den Verlauf der Zustände der Prädiktoren ein und unterstreichen Sie den jeweils ausgewählten Prädiktor. Heben Sie weiterhin die falsch vorhergesagten Sprünge durch Unterstreichen hervor. Wie viele Fehlannahmen werden insgesamt gemacht? Worauf sind diese zurückzuführen?

c) Warum sind Ihrer Meinung nach die Prädiktoren im Teil 2 so initialisiert (Hinweis: Statische Sprungvorhersage).

d) Verwenden Sie nun 2-Bit sättigende Zähler zur Sprungvorhersage. Der Zustand „oben links“ heisse „Strongly Taken“ (ST); der Zustand „oben rechts“ heisse „Weakly Taken“ (WT); der Zustand „unten links“ heisse „Strongly Not Taken“ (SNT); der Zustand „unten rechts“ heisse „Weakly Not Taken“ (WNT). Die Prädiktoren seien wie folgt initialisiert: Sprung 1 (SNT), Sprung 2 (ST), Sprung 3 (ST). Wieso werden nun weniger Fehlannahmen gemacht?

Aufgabe 4

Gegeben seien drei kleine Cachespeicher DM, A2 und AV, die jeweils 8 Cacheblöcke enthalten, wobei jeder Cacheblock vier Bytes umfasst. Cache DM ist als direct mapped Cache organisiert, Cache A2 als 2-fach assoziativer Cache und Cache AV ist voll assoziativ. Bei A2 und AV soll die LRU-Ersetzungsstrategie angewendet werden. Nehmen Sie an, die Caches seien zu Beginn leer und es soll eine Serie von einzelnen Bytes mit den folgenden 32-Bit-Adressen gelesen werden:

`294928070, 294928009, 294928039, 294928083, 294928066, 294928068, 294928035, 294928080, 294928093, 294928067, 294928079, 294928037, 294928084, 294928009`

`(294928000 dezimal entspricht = 0001 0001 1001 0100 0011 1110 1000 0000 bin"ar).`

a) Welcher technologische Unterschied besteht zwischen Cache und Hauptspeicher?

b) Geben Sie zunächst für alle drei Cachespeicher an, wieviel Bits zur Verwaltung eines Cacheblocks benötigt werden. Dabei sollen für den Zustand des Cacheblocks 2 Bits verwendet werden (ein Valid-Bit und ein Dirty-Bit).

c) Geben Sie nun für jeden Cache an, ob es sich beim Lesezugriff auf die jeweilige Adresse um einen Cache-Hit oder um einen Cache-Miss handelt.

d) Stellen Sie den Zustand der drei Caches nach dem letzten Speicherzugriff dar, d.h. für jeden Cache-Block den Cache-Tag und die vier Datenbytes m[x1-x4]. Dabei sollen mit der Schreibweise m[x1-x4] die aus dem Speicherbereich [x1,x4] gelesenen Datenbytes repräsentiert werden.

Besprechung am Donnerstag, den 4.7.2002.

Hinweis: Da die Musterlösungen diesmal relativ lang sind, wird am Dienstag, den 2.7., eine Musterlösung ins Netz gestelt.