



11. Aufgabenblatt

10.7.2023

Mikroarchitekturen, Pipeline-Prozessor, Hazards, Dateien

Aufgabe 1: Theoriefragen

- a) In welche Phasen wird die Ausführung einer Instruktion bei dem vorgestellten Pipelined-Prozessor unterteilt?
- b) Was versteht man unter einem Hazard?
- c) Bei der Entwicklung von modernen Prozessoren werden viele Ressourcen in die Verbesserung der Sprungvorhersage gesteckt. Warum wirkt sich ein falsch vorhergesagter Sprung negativ auf die Performance aus?
- d) Bei Prozessoren unterscheidet man zwischen in-order oder out-of-order Ausführung von Instruktionen. Welcher Ansatz ist schneller? Können Sie sich auch Einsatzszenarien für den anderen Ansatz vorstellen?

Aufgabe 2: Auflösen von Hazards

Zeichnen Sie die Hazards der angegebenen ARM-Programme in das Pipeline-Diagramm ein. Modifizieren Sie anschließend das Programm, sodass es korrekt ausgeführt wird. Sie dürfen nur **nops** einfügen oder die Zeilenreihenfolge ändern, nicht jedoch die Instruktionen verändern. Versuchen Sie die minimale Lösung zu finden. Gehen Sie davon aus, dass der eingesetzte Prozessor keine Hazard Unit besitzt. Register **r11** ist initial mit 0 belegt.

a) Programm 1

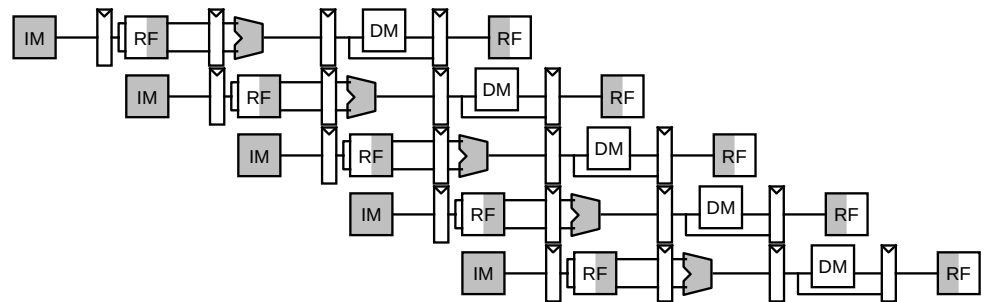
add r2, r11, #3

add r0, r11, #5

lsl r0, r0, #2

ldr r1, [r11, #40]

and r1, r1, #1

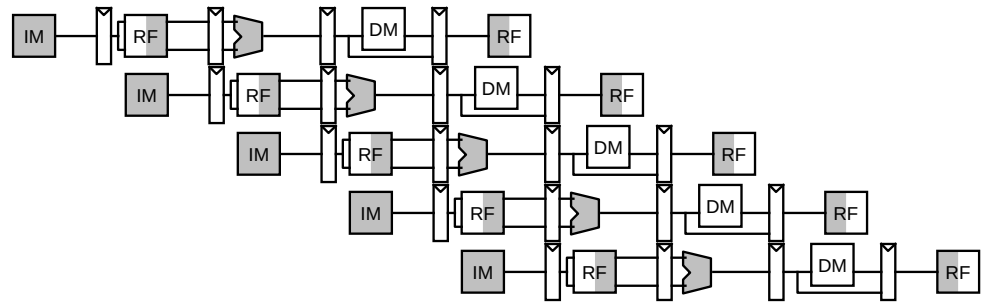


b) Programm 2

```

sub r3, r2, r1
add r1, r3, r2
ldr r0, [r5, #4]
orr r0, r3, r4
add r3, r1, #5

```



Aufgabe 3: Dateien

Diese Aufgabe wurde von Florian Piana im Rahmen eines Praktikums in der Lehre entwickelt.

Dateien sind essentielle Bestandteile der Nutzer-Maschinen-Interaktion und ein wesentliches Konzept in der Datenmanipulation des alltäglichen Gebrauchs. Diese Aufgabe soll Ihnen die Verwendung von Dateien mit Hilfe des ARM-Befehlssatzes nahebringen und verständlich machen.

Die Grundlage für einen Dateizugriff bietet ein sogenannter „File-Descriptor“ (Oftmals abgekürzt als „FD“). Jedes Mal, wenn Sie einen Dateizugriff anfordern, wird ein solcher File-Descriptor vom Betriebssystem festgelegt und dem Programmierer (in diesem Falle Sie) übergeben. Der File-Descriptor dient nun als eine Art Identifikationsnummer, mit der Sie verschiedene Funktionalitäten an der Datei aufrufen können.

In Niedersprachen sind Software-Interrupts besonders dafür geeignet. Wir werden uns zunächst mit Software-Interrupts auseinandersetzen, um diese später mit bereitgestellten Funktionen zu vergleichen.

Die

```
SWI{cond} <immediate_24_bit>
```

Instruktion wird genutzt, um einen Software-Interrupt auszuführen. Dies sorgt für eine Statusänderung der Prozessor-Einheit in einen besonderen Modus (die Details werden hier nicht weiter behandelt), sodass das Betriebssystem die Kontrolle übernehmen und eine durch den Immediate-Wert bestimmte Aktion ausführen kann. Jedoch wird aus Performancegründen inzwischen immer ein Interrupt mit Immediate-Wert 0 (Null) ausgeführt und die Aktion als Zahl im Register 7 (R7) abgelegt.

Man nennt solch eine Interaktion mit dem Betriebssystem manchmal auch „syscall“.

Die Werte, die in R7 abgelegt werden müssen, sind systemspezifisch. Wir gehen hier davon aus, dass Sie Dateizugriff auf ein ARM-Linux mit EABI-Standard (also zum Beispiel dem Client-SSH) nutzen wollen. Hier eine Zusammenfassung einiger Funktionalitäten, die durch Software-Interrupts genutzt werden können:

R7-Wert	Beschreibung	R0-Wert	R1-Wert	R2-Wert	Rückgabe
3	Lesen	File-Descriptor	Zeiger auf Puffer, in dem die gelesenen Bytes gespeichert werden sollen	Anzahl der zu lesenden Bytes	Anzahl der tatsächlich gelesenen Bytes
4	Schreiben	File-Descriptor	Zeiger auf Puffer, in dem die zu schreibenden Bytes liegen	Anzahl der zu schreibenden Bytes	Anzahl der tatsächlich geschriebenen Bytes
5	Öffnen	Zeiger auf Datei/Pfad (als String)	Flags (gibt an, welche Zugriffsrechte angefordert werden sollen)	Modus (dies kann für unsere Zwecke zunächst 0666 ₈ bleiben)	Negativ (-1), falls Datei nicht geöffnet werden konnte, sonst File-Descriptor
6	Schließen	File-Descriptor	-	-	Bei Fehler -1, sonst 0
19	Lese-/Schreib-Index verschieben	File-Descriptor	Relative Verschiebung	Bezugspunkt	Der geänderte Lese-/Schreib-Index

a) Die Konsole als „Datei“

Viele Betriebssysteme – einschließlich Linux-Systeme – nutzen das Prinzip der File-Descriptors sogar um die Konsole zu repräsentieren. Dadurch kann Text auf der Konsole auch ausgegeben werden, indem man sie als (bereits geöffnete) „Datei“ behandelt. Diese Konsolen-Schnittstelle nennt man häufig „stdout“ (gesprochen: Standard-Out). STDOUT hat standardmäßig den File-Descriptor 1.

Hello World

Das Vorgehen ist beim Schreiben in die Konsole dann wie folgt:

1. Vorbereiten der Daten und Argumente

- In R0 den Wert 1 ablegen (da 1 der File-Descriptor für die Ausgabe auf der Konsole ist)
- In R1 einen Zeiger auf Zeichen, die in die Ausgabe geschrieben werden sollen, ablegen.
- In R2 die Anzahl der Zeichen, die in STDOUT geschrieben werden sollen.
- In R7 den Wert 4 ablegen (da wir in die Konsole schreiben möchten)

2. SWI 0 aufrufen

Aufgabe: Schreiben Sie mit Hilfe des ARM-Instruktionssatzes ein Programm, in dem der String „Hello World“ ausgegeben wird.

Eingabe

Aufgabe: Versuchen Sie nun, eine Eingabe des Nutzers in einem Puffer zu speichern, indem Sie die Konsole als „Datei“ betrachten. Geben Sie die Nutzer*inneneingabe anschließend aus und hängen Sie an das Ende der Nutzer-Eingabe ein „!“ an. Was fällt Ihnen auf, wenn der Nutzer mehr Zeichen eingibt, als Sie in den Argumenten des „Lesen“-Syscall angegeben haben? Hinweis: Der ASCII-Code für „!“ ist 33_{10} (als dezimal) bzw. 21_{16} (als hexadezimal).