Übung 12

Rechnerarchitektur
Wintersemester 18/19
Prof. Dr.-Ing. Jochen Schiller

Ausgabe 18.01.2019
Abgabe 01.02.2019, 10:15 (s.t.)

Bitte beachten Sie die allgemeinen Hinweise auf Übungszettel 1

Hinweis: Dies ist der letzte *reguläre* Übungszettel. Am Fr, 25.01. stellen wir einen klausurrelevanten aber optional einzureichenden Zusatzzettel online. Wenn Sie die $n-2$ Grenze um ein Übungsblatt verfehlt haben, können Sie diese bei erfolgreicher Abgabe des 13. Zettels also noch einhalten.

Aufgabe 1: Sprungvorhersage

Fragen

1. Warum ist Sprungvorhersage sinnvoll?
2. Was ist ein Vor-/Nachteil von 2bit-Zustandsautomaten gegenüber 1bit-Zustandsautomaten? Welcher überwiegt?
3. Was ist der Vorteil vom *Hysteresis Scheme* gegenüber dem *Saturation Counter Scheme*?
4. Was ist der *Branch Target Buffer (BTB)* und was tut er?
5. Warum ist *Not Taken* bzw. *Strongly Not Taken* der einzig sinnvolle Startzustand für die Automaten (auch im Zusammenhang mit dem BTB)?
6. Im Anhang zu dieser Aufgabe sind Ihnen zwei NASM-artige Pseudocodes für While-Schleifen gegeben. Erklären Sie warum diese zwar semantisch äquivalent sind, aber ihre jeweilige Syntax für Sprungvorhersagen einen Unterschied bedeutet.

```
while(x > y) {
    S();
}
```

: Originalcode

```
.loopBody:
CMP x, y
JLE .endLoop
    CALL S
JMP .loopBody
.endLoop:
```

: 1. NASM-Übersetzung

```
JMP .startLoop
.loopBody:
    CALL S
.startLoop:
CMP x, y
JG .loopBody
```

: 2. NASM-Übersetzung

Predicated Instructions

Gegeben sei folgende C-artige Befehlsfolge:

:

```
a = a + 1;
a = a + 1;
a = a + 1;
if(a == 0) {
    b = b / 2;
    b = b + 2;
}
a = a + 1;
a = a + 1;
a = a + 1;
```

Gehen Sie wieder von der 5-stufigen Pipeline aus (~~wie Übung 7, Aufgabe 1a~~). Für bedingte Befehle wird in der zweiten Stufe (Instruction Decode (ID)) festgestellt werden, ob die Bedingung erfüllt ist oder nicht. Bedingte Befehle beeinflussen das PSW nicht! Ein Pipelineflush kostet 10 Takte und kann sofort beginnen, nachdem festgestellt wurde, dass eine Vorhersage inkorrekt war, d.h. der inkorrekt vorhergesagte Befehl muss nicht erst zu Ende ausgeführt werden.

1. **Ausführung ohne Predicated Instructions**
   Übersetzen Sie die Befehlsfolge zunächst wie gewohnt mit einem bedingten Sprung für die IF-Verzweigung nach Assembler. Wie viele Takte sind zur Abarbeitung der Befehlsfolge nötig, wenn der Bedingungsblock ausgeführt werden soll, die Sprungvorhersage aber eine falsche Vorhersage trifft? Visualisieren Sie Ihre Lösung!

2. **Ausführung mit Predicated Instructions**
   Übersetzen Sie jetzt die Befehlsfolge unter Zuhilfenahme von Predicated Instructions. Wie viele Takte sind jetzt zur Abarbeitung der Befehlsfolge nötig?
   Visualisieren Sie Ihre Lösung!

3. **Vergleich**
   Geben Sie an, welche Lösung effizienter ist und unter welchen Bedigungen sich das ändern würde.

Aufgabe 2: Assembler Wrapper

Schreiben Sie einen Wrapper in NASM, der die im KVV gestellte (spezielle[1]) Sortierfunktion für Testzwecke aufruft. Machen Sie sich dazu mit folgenden Punkten vertraut:

- Auslesen von Konsolenparametern in NASM
- Stackframes öffnen und schließen
- Aufrufen von Bibliotheksfunktionen in NASM
- Programmeinstiegs-Label für den Compiler

Der Wrapper soll mindestens die folgende Funktionalität besitzen:

1. Länge des Testarrays aus den Konsolenparametern auslesen
2. Übersetzen des Konsolenparameters von einem String in einen Integerwert mittels Ihrer '`strToInt`' Funktion oder einer C-Bibliotheksfunktion
3. Entsprechende Menge Speicher reservieren (Stackframe)
4. Arrayeinträge mittels der C-Bibliotheksfunktion '`rand`' mit Zufallswerten befüllen
5. Ausgabe des unsortieren Arrays
6. Aufrufen der Sortierfunktion mit entsprechenden Parametern
7. Ausgabe des sortieren Arrays
8. Schließen des Stackframes
9. Programm beenden

Zusätzliche Funktionalitäten, wie z.B. Fehlerabfragen, sind natürlich erlaubt.

Wenn Sie das im KVV gestellte Makefile mit '`make`' aufrufen, wird die '`sort.asm`' mit der von Ihnen geschriebenen '`wrapper.asm`' zusammen compiliert.
Mit '`make clean`' werden alle '`.o`' Dateien und die fertigen '`sort`' Kompilate gelöscht.

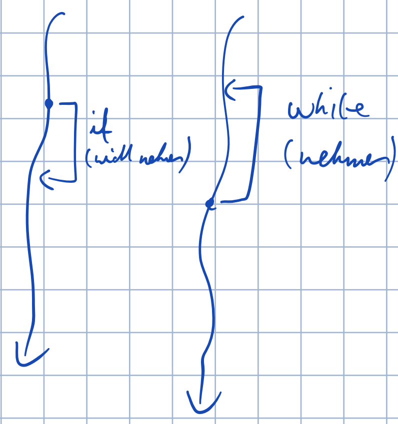
[1] Eine Sortierfunktion bedarf einer Metrik. "Aufsteigend" ist nur eine mögliche Metrik. In diesem Fall handelt es sich um eine Variante von *Dropsort*. Bestimmte Zahlen werden hier quasi *aussortiert*.

Data Hazards → Forwarding / Reordering
Structural Hazards → mehr Hardware
Control Hazards → Sprungvorhersage

versuchen mögliche Unterbr., durch cmp vorausahnen, zu umgehen

statische Sprungvorhersage

- Sprünge nach hinten nehmen
- Sprünge nach vorne nicht nehmen

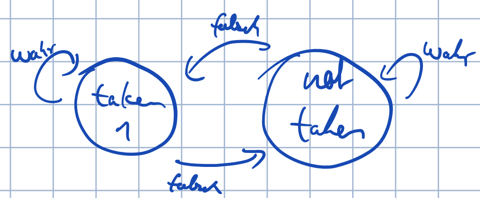


lokale pro Befehl

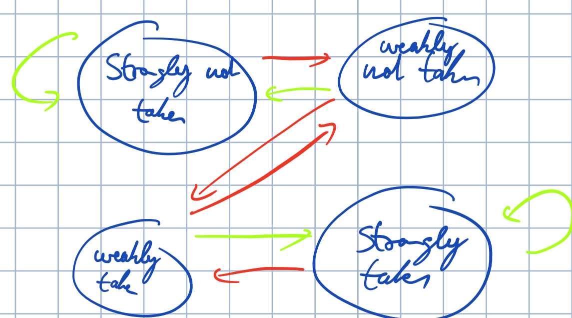
Branch	Target	Prediction
(Vergleich JE)	(Ziel der Schleife)	1 (für jump)
Adresse 1	Adresse 2	0 (für nicht)

globale pro Programm

1-Bit



2-Bit saturated counter Scheme



Hysteresisscheme