Freie Universität Berlin

Übung 6

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

Ausgabe 23.11.2018
Abgabe 07.12.2018, 10:15 (s.t.)

Bitte beachten Sie die allgemeinen Hinweise auf Übungszettel 1

Aufgabe 1: Pipelines

Aufgabe 1.1: NOPs

Gehen Sie im folgenden von einer 6-stufigen Pipeline aus. Die 6 Stufen sind:

1) IF - Instruction Fetch (Befehl holen)
2) ID - Instruction Decode (Befehl dekodieren)
3) FL - Fetch Left Operand (Ersten Operanden holen)
4) FR - Fetch Right Operand (Zweiten Operanden holen)
5) EX - Execution (Ausführung)
6) WB - Write Back (Rückspeichern)

Operanden können erst dann aus den Registern geladen werden, nachdem sie zurückgespeichert wurden.

Bestimmen Sie für die folgenden Befehlsfolgen, ob diese ohne das Einfügen von NOPs (No OPerations) ausführbar sind. Fügen Sie gegebenenfalls eine geeignete Anzahl von NOPs ein, damit die Folgen korrekt bearbeitet werden können.

Beantworten Sie für jede Befehlsfolge die folgenden Fragen:

1. Müssen NOPs eingefügt werden, wenn ja, wo?
2. Wie lange (in Takten) dauert die vollständige Abarbeitung (mit und ohne NOPs)?

add a, b, c ist zu verstehen als $c = a + b$.

a)
```
add a, b, c
add c, d, e
```

b)
```
add x, y, z
add z, b, a
```

c)
```
add z, y, x
mul b, x, a
mul x, b, d
```

d)
```
add a, b, x
add c, d, z
mul x, z, y
add x, z, k
```

e)
```
add a, b, c
mul a, b, d
add e, f, m
mul e, f, k
add p, q, a
```

Aufgabe 1.1: Reordering

Geben Sie ein Reordering an, so dass die Befehle korrekt und ohne NOPs ausgeführt werden können. Gehen Sie von der gleichen Pipeline aus wie in Aufgabe 1.1. Um welchen Faktor ist Ihr Programm mit Reordering schneller als mit dem Einfügen von NOPs?

```
add a, b, c
add c, d, e
mul a, b, k
mul m, n, p
sub b, a, b
```

Aufgabe 2: Alte Klausuraufgabe

Hinweise:

- Es handelt sich um eine Aufgabe, wie sie auch in der Klausur gestellt sein könnte.
- Sie sollten inzwischen in der Lage sein, die Aufgabe selbstständig zu bearbeiten.
- Die Aufgabe ist nicht ganz leicht, aber eine gute Gelegenheit Ihre Herangehensweise zu üben.
- Wichtig: Lesen Sie sich zuerst die Aufgabenstellung vollständig durch. Wahrscheinlich müssen Sie nicht das ganze Programm verstehen um die Fragen zu beantworten.
- Wenn Sie diese und ähnliche Aufgaben unter 15 Minuten vollständig lösen können (Fragen 1-4), sind Sie für die Klausur sehr gut gerüstet!

Gegeben sei folgende NASM-Funktion (Parameter stehen in rdi und rsi, der Rückgabewert steht in rax):

```
01  GLOBAL function

02  function:
03      MOV r8, 1

04  _loop:
05      CMP r8, rsi
06      JGE _endloop

07      MOV al, [rdi+r8]
08      MOV r9, r8
09      MOV r10, r8

10  _innerloop:
11      DEC r9
12      CMP r9, 0
13      JL  _endinnerloop

14      MOV bl, [rdi+r9]
15      CMP al, bl
16      JL  _endinnerloop

17      MOV [rdi+r9], al
18      MOV [rdi+r10], bl
19      DEC r10
20      JMP _innerloop

21  _endinnerloop:
22      INC r8
23      JMP _loop

24  _endloop:
25      MOV rax, rdi
26      RET
```

Aufgerufen wird diese Funktion folgendermaßen:

```
int8_t array[] = "World";
function(array, 5);
```

Sie können diese ASCII-Tabelle zur Hilfe nehmen:

Char	W	d	l	o	r
ASCII (Dezimal)	87	100	108	111	114

Beantworten Sie folgende Fragen:

1. Geben Sie den Inhalt des Arrays nach dem ersten Schleifendurchlauf von *_loop* an.
2. Wie oft wird die innere Schleife (*_innerloop*) insgesamt ausgeführt?
3. Welchen Rückgabewert gibt die Funktion zurück?
4. Wie sieht der Inhalt der Variable *array* nach dem Funktionsaufruf aus?
5. (Optional:) Was tut das Programm?