Übung 8

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

Ausgabe 07.12.2018
Abgabe 21.12.2018, 10:15 (s.t.)

Bitte beachten Sie die allgemeinen Hinweise auf Übungszettel 1

Vorlesung bei _ = 1

Aufgabe 1: Diverses

Aufgabe 1.1: Carry-lookahead-Addierer

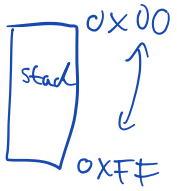
1. Lösen Sie die rekursive Berechnung von $ü_3$ dem Skript entsprechend in Und- und Oder-Verknüpfungen von g_i und p_i mit $i \leq 3$ auf.
2. Berechnen Sie g_i und p_i mit $i \leq 3$ für $a_{3..0} = 1010$ und $b_{3..0} = 0110$.
3. Berechnen Sie die ersten vier Ziffern s_i mit $i \leq 3$ der Summe von a und b aus b) entsprechend der Carry-lookahead-Addierer-Methode.

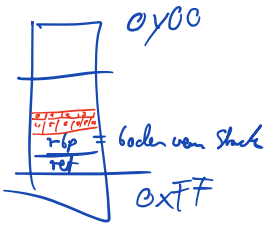
Aufgabe 1.2: Hacking

Gehen Sie davon aus, dass Sie Zugriff auf ein Array haben, das im Stack angelegt wurde.

Press me ↝ 1. Wie können Sie durch dieses Array die Rücksprungadresse der Funktion manipulieren?
2. Wie können Sie trotzdem noch die Kontrolle über das System erlangen wenn Sie nur den alten "rbp" Wert manipulieren können?
3. Würden Ihre Angriffe noch ohne Weiteres funktionieren, wenn der Stack nicht mehr *descending* sondern *ascending* ist?

smashing the stack for fun and profit





Aufgabe 2: Funktionspointer

Implementieren Sie die folgenden Funktionen in Assembler:

- Die Funktion *fold* soll mit einer übergebenen Funktion alle Werte eines gegebenen Arrays miteinander verrechnen und das Ergebnis zurückgeben.

```
int64_t fold(int64_t (*func)(int64_t, int64_t), uint64_t len, int64_t a[len],
    uint8_t dir);
```

  Parameter:

  func - Funktion mit der alle Einträge aus `a` verrechnet werden sollen
  len - Länge von Array `a`
  a - Array mit den zu verechnenden Werten
  dir - Richtung in die *fold* arbeiten soll (0 - von vorne nach hinten, 1 - von hinten nach vorne)

- Die Funktion *zipWith* soll aus zwei gegebenen Arrays jeweils die Elemente mit dem gleichen Index mittels einer übergebenen Funktion verrechnen und das Ergebnis an die gleiche Position in einem dritten Array schreiben.

```
void zipWith(int64_t (*func)(int64_t, int64_t), uint64_t len, int64_t a[len],
    int64_t b[len], int64_t c[len]);
```

  Parameter:

  func - Funktion mit der die Einträge aus den Arrays `a` und `b` verrechnet werden sollen
  len - Länge der Arrays
  a - Array mit den ersten Operanden
  b - Array mit den zweiten Operanden
  c - Array für die Ergebnisse

$\ddot{u}_{i+1} = g_i \vee p_i \ddot{u}_i$

Man erhält

$\ddot{u}_1 = g_0 \vee p_0 \ddot{u}_0$

$\ddot{u}_2 = g_1 \vee p_1 g_0 \vee p_1 p_0 \ddot{u}_0$

$\ddot{u}_3 = g_2 \vee p_2 g_1 \vee p_2 p_1 g_0 \vee p_2 p_1 p_0 \ddot{u}_0$

usw.

Aufgabe 1.1: Carry-lookahead-Addierer

1. Lösen Sie die rekursive Berechnung von $\ddot{u}_3$ dem Skript entsprechend in Und- und Oder-Verknüpfungen von g_i und p_i mit $i \leq 3$ auf.
2. Berechnen Sie g_i und p_i mit $i \leq 3$ für $a_{3..0} = 1010$ und $b_{3..0} = 0110$.
3. Berechnen Sie die ersten vier Ziffern s_i mit $i \leq 3$ der Summe von a und b aus b) entsprechend der Carry-lookahead-Addierer-Methode.

1.1

$$\ddot{u}_4 = g_3 \vee p_3(g_2 \vee p_2(g_1 \vee p_1(g_0 \vee p_0 \ddot{u}_0)))$$

$$= g_3 \vee p_3 g_2 \vee p_3 p_2 (g_1 \vee p_1 (g_0 \vee p_0 \ddot{u}_0))$$

$$= g_3 \vee p_3 g_2 \vee p_3 p_2 g_1 \vee p_3 p_2 p_1 (g_0 \vee p_0 \ddot{u}_0)$$

$$= g_3 \vee p_3 g_2 \vee p_3 p_2 g_1 \vee p_3 p_2 p_1 g_0 \vee p_3 p_2 p_1 p_0 \ddot{u}_0$$

- $g_i = a_i b_i$ (generate carry, erzeuge Übertrag) und
- $p_i = (a_i \oplus b_i)$ (propagate carry, leite Übertrag weiter)
- g_i und p_i können direkt aus den Eingangsvariablen erzeugt werden.

2)

$g_i = a_i \wedge b_i$, $p_i = a_i \oplus b_i$, $a_{3..0} = 1010$, $b_{3..0} = 0110$

$g_0 = 0 \wedge 0 = 0$ $\quad p_0 = 0 \oplus 0 = 0$

$g_1 = 1 \wedge 1 = 1$ $\quad p_1 = 1 \oplus 1 = 0$

$g_2 = 0 \wedge 1 = 0$ $\quad p_2 = 0 + 1 = 1$

$g_3 = 1 \wedge 0 = 0$ $\quad p_3 = 1 \oplus 0 = 1$

3. Berechnen Sie die ersten vier Ziffern s_i mit $i \leq 3$ der Summe von a und b aus b) entsprechend der Carry-lookahead-Addierer-Methode.

3)

- $\ddot{u}_{i+1} = a_i b_i \vee (a_i \oplus b_i) \ddot{u}_i = g_i \vee p_i \ddot{u}_i$
- $s_i = (a_i \oplus b_i) \oplus \ddot{u}_i = p_i \oplus \ddot{u}_i$

$\ddot{u}_0 = 0$

$\ddot{u}_1 = g_0 \vee p_0 \wedge \ddot{u}_0 = 0 \vee 0 \wedge 0 = 0$

$\ddot{u}_2 = g_1 \vee p_1 \wedge \ddot{u}_1 = 1 \vee 0 \wedge 0 = 1$

$\ddot{u}_3 = g_2 \vee p_2 \wedge \ddot{u}_2 = 0 \vee 1 \wedge 1 = 1$

$S_0 = p_0 \oplus \bar{u}_0 = 0 \oplus 0 = 0$

$S_1 = p_1 \oplus \bar{u}_1 = 0 \oplus 0 = 0$

$S_2 = p_2 \oplus \bar{u}_2 = 1 \oplus 1 = 0$

$S_3 = p_3 \oplus \bar{u}_3 = 1 \oplus 1 = 0$

"0000"