# Rechnerstrukturen: Übungsblatt 5

Florian Ludewig (185722)

15. Juni 2020

# Aufgabe 1

Aufgabe 1 (Ich nehme an wir können die Zahlen in Bitlänge 8 umrechnen)

$B_0 = 1$ $\quad *30 = 30$

$B_1 = -\frac{1}{2}\left[\binom{2}{0}\cdot 1\right] = -\frac{1}{2}$ $\quad *30 = -15$

$B_2 = -\frac{1}{3}\left[\binom{3}{0}\cdot 1 + \binom{3}{1}\cdot B_1\right] = -\frac{1}{3}\cdot\left(-\frac{1}{2}\right) = \frac{1}{6}$ $\quad *30 = 5$

$B_3 = -\frac{1}{4}\left[\binom{4}{0}\cdot 1 + \binom{4}{1}\cdot\left(-\frac{1}{2}\right) + \binom{4}{2}\cdot\frac{1}{6}\right] = 0$ $\quad *30 = 0$

$B_4 = -\frac{1}{5}\left[\binom{5}{0}\cdot 1 + \binom{5}{1}\cdot\left(-\frac{1}{2}\right) + \binom{5}{2}\cdot\frac{1}{6} + \binom{5}{3}\cdot 0\right] = -\frac{1}{30}$ $\quad *30 = -1$

a) 30: $00011110_2$

$11100001$ EK

$+\ 00000001$

$11100010$ ZK

−15: $00001111_2$

$(1)00011111$

$11110000$ EK

$+\ 00000001$

$11110001$ ZK

5: $00000101_2$

$11111010$ EK

$+\ 00000001$

$11111011$ ZK

0: $00000000_2$

$11111111$ EK

$+\ 00000001$

$(1)00000000$ ZK

... Overflow

−1 $10000001_2$

$01111110$ EK

$+\ 00000001$

$01111111$ ZK

b) 30: $30/8 = 3$ R $6$ ↑

$3/8 = 0$ R $3$

$036_8$

−15: $11110001_2$ ZK(15)

$= 241_{10}$

$241/8 = 30$ R $1$ ↑

$30/8 = 3$ R $6$

$3/8 = 0$ R $3$

$361_8$

5: $5/8 = 0$ R $5$

$005_8$

0: $0/8 = 0$ R $0$

$000_8$

−1: $01111111_2$ ZK(1)

$= 255_{10}$

$255/8 = 31$ R $7$ ↑

$31/8 = 3$ R $7$

$3/8 = 0$ R $3$

$377_8$

c) 30: $30/16 = 1$ R $14$ ↑

$1/16 = 0$ R $1$

$1E_{16}$

−15: $11110001_2$ ZK(15)

$= 241_{10}$

$241/16 = 15$ R $1$ ↑

$15/16 = 0$ R $15$

$F1_{16}$

5: $5/16 = 0$ R $5$

$05_{16}$

0: $0/16 = 0$ R $0$

$00_{16}$

−1: $01111111_2$ ZK(1)

$= 255_{10}$

$255/16 = 15$ R $15$ ↑

$15/16 = 0$ R $15$

$FF_{16}$

## Aufgabe 2

```c
#include <stdio.h>
#include <limits.h>
#include <time.h>
#include <stdlib.h>

unsigned W = sizeof(unsigned) * 8;
unsigned x, y;

int arithmeticRightShift(int x, int n) {
    if (x < 0 && n > 0)
        return x >> n | ~(~0U >> n);
    else
        return x >> n;
}

unsigned a() {
  return ~(arithmeticRightShift((x|(~x+1)), W-1))&1;
}
unsigned b() {
  return ~(arithmeticRightShift(x, W-1) << 1);
}
unsigned c() {
  return ~(~x|(y^(INT_MIN+INT_MAX)));
}
unsigned d() {
  return x^(INT_MIN+INT_MAX);
}
unsigned e() {
  return ((x^y)&~y)|(~(x^y)&y);
}
unsigned f() {
  return arithmeticRightShift(((x<0) ? (x+3) : x), 2);
}

void test_all(unsigned expected_value) {
  printf("a: %0x ?== %0x (match: %d)\n", expected_value, a(), expected_value
       == a());
  printf("b: %0x ?== %0x (match: %d)\n", expected_value, b(), expected_value
       == b());
  printf("c: %0x ?== %0x (match: %d)\n", expected_value, c(), expected_value
       == c());
  printf("d: %0x ?== %0x (match: %d)\n", expected_value, d(), expected_value
       == d());
  printf("e: %0x ?== %0x (match: %d)\n", expected_value, e(), expected_value
       == e());
  printf("f: %0x ?== %0x (match: %d)\n", expected_value, f(), expected_value
       == f());
}

int main(void) {
  srand(time(NULL));
  x = (~rand())+1;
  y = (~rand())+1;

  printf("1:\n");
  test_all(x);

  printf("\n2:\n");
  test_all(x&y);

  printf("\n3 positive x:\n");
  test_all((x<0?1:-1));

```

```
58    x = (~-34546)+1;
59    printf("\n3 negative x:\n");
60    test_all((x<0?1:-1));
61
62    return 0;
63  }
```

Die Ausgabe des obigen C-Programms ist:

```
1  1:
2  a: ab739e86 ?== 0 (match: 0)
3  b: ab739e86 ?== 1 (match: 0)
4  c: ab739e86 ?== 89529682 (match: 0)
5  d: ab739e86 ?== 548c6179 (match: 0)
6  e: ab739e86 ?== ab739e86 (match: 1)
7  f: ab739e86 ?== eadce7a1 (match: 0)
8
9  2:
10 a: 89529682 ?== 0 (match: 0)
11 b: 89529682 ?== 1 (match: 0)
12 c: 89529682 ?== 89529682 (match: 1)
13 d: 89529682 ?== 548c6179 (match: 0)
14 e: 89529682 ?== ab739e86 (match: 0)
15 f: 89529682 ?== eadce7a1 (match: 0)
16
17 3 positive x:
18 a: ffffffff ?== 0 (match: 0)
19 b: ffffffff ?== 1 (match: 0)
20 c: ffffffff ?== 8dd2100a (match: 0)
21 d: ffffffff ?== 86f1 (match: 0)
22 e: ffffffff ?== ffff790e (match: 0)
23 f: ffffffff ?== ffffde43 (match: 0)
24
25 3 negative x:
26 a: ffffffff ?== 0 (match: 0)
27 b: ffffffff ?== ffffffff (match: 1)
28 c: ffffffff ?== 86e2 (match: 0)
29 d: ffffffff ?== ffff790d (match: 0)
30 e: ffffffff ?== 86f2 (match: 0)
31 f: ffffffff ?== 21bc (match: 0)
```

Wodurch sich eindeutig folgende Lösungen ergeben:
1) e
2) c
3) b für negative `x`, es gibt keine Lösung für positive `x`