# 64-041 Übung Rechnerstrukturen und Betriebssysteme

## Aufgabenblatt 6 Ausgabe: 23.11., Abgabe: 30.11. 24:00

| Gruppe | |
|---|---|
| Name(n) | Matrikelnummer(n) |
| | |

### Aufgabe 6.1 (Punkte 5+10+10+10)

*Radix-50 Codierung:* In den Urtagen der Informatik, als Speicher noch sehr teuer war, wandte die Firma DEC (Digital Equipment Corporation) folgendes Verfahren an, um Zeichenketten komprimiert im Speicher ablegen zu können:

1. Wie man sieht, wurde damals sogar nur ein Subset des ASCII-Codes mit 40 Zeichen benutzt. Zunächst wurden die Zeichen nach folgender Tabelle in 6-bit codiert.

| Bits | 2…0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5…3 | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 000 | space | A | B | C | D | E | F | G |
| 001 | H | I | J | K | L | M | N | O |
| 010 | P | Q | R | S | T | U | V | W |
| 011 | X | Y | Z | $ | . | % | 0 | 1 |
| 100 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

   Beispielsweise würde der Buchstabe „Q“ zu $010001_2$ umcodiert.

2. Im zweiten Schritt wird nach der Formel $rad50 = 40^2 \cdot c_1 + 40 \cdot c_2 + c_3$ aus jeweils drei umcodierten Buchstaben $b_i$ ein 16-bit Wert gemacht, der dann abgespeichert wurde. Die Bezeichnung *Radix-50* kommt wegen der Beziehung $40_{10} = 50_8$.

(a) Wie würde die Zeichenkette „CR7“ in der Radix-50 Darstellung codiert werden? Geben Sie das Ergebnis bitte als Hexadezimalzahl an.

(b) Ersetzen Sie in obiger Formel die Multiplikationen durch Schiebeoperationen und Additionen und schreiben Sie eine möglichst einfache Java-Funktion die Ihre neue Formel implementiert. Dass in Java der Typ `int` 32-bit hat, soll uns dabei nicht weiter stören.

```
int rad50_encode (int c1, int c2, int c3)       // drei Zeichen
{
  ...
}
```

(c) Beschreiben Sie (textuell, ein Programm ist nicht nötig), wie man aus der Radix-50 Darstellung, wieder die drei darin codierten Buchstaben zurückgewinnen kann.

(d) Welche Buchstaben sind in den Radix-50 Werten $737A_{16}$ und $0520_{16}$ codiert?

### Aufgabe 6.2 (Punkte 5+5+5+5)

*Shift-Operationen statt Multiplikation:* Ersetzen Sie die folgenden Berechnungen *möglichst effizient* durch eine Folge von Operationen: <<, +, -. Nehmen Sie für die Variablen $x$ und $y$ den Datentyp `int` (32-bit Zweierkomplementzahl) an.

(a) $y = 18 \cdot x$

(b) $y = 14 \cdot x$

(c) $y = -56 \cdot x$

(d) $y = 62 \cdot (x + 2)$

### Aufgabe 6.3 (Punkte 5+5+10+10)

*Logische- und Shift-Operationen:* Realisieren Sie, die folgenden Funktionen als *straightline*-Code in Java, das heißt ohne Schleifen oder If-Else Abfragen, bzw. ternärer Operator `.. ? .. : ...` Außerdem dürfen nur einige der logischen und arithmetischen Operatoren benutzt werden:

```
! ~ & ^ | + << >> >>>
```

Alle Eingabeparameter und Rückgabewerte sind jeweils (32-bit) Integerwerte.

(a) `bitNor(x,y)` Diese Funktion soll das bitweise NOR liefern: $\overline{x_i \vee y_i}$. Als Operatoren dürfen nur & und ~ (AND, Negation) benutzt werden.

(b) `bitXor(x,y)` Diese Funktion soll die bitweise XOR-Verknüpfung (Antivalenz) realisieren: $x_i \neq y_i$. Als Operatoren dürfen nur & und ~ (AND, Negation) benutzt werden.

(c) `getByte(x,n)` Diese Funktion soll das Byte `n` ($0 \leq$ `n` $\leq 3$) aus dem Wert `x` extrahieren.

(d) `rotateRight(x,n)` Die Funktion soll den in Java nicht vorhandenen Rotate-Right Operator für `x` nachbilden. Für das zweite Argument `n` gilt: $0 \leq$ `n` $\leq 31$.

### Aufgabe 6.4 (Punkte 5+5+5)

*Codierung:* Für eine Messung soll ein einschrittiger Binärcode entwickelt werden, der 11 Elemente umfasst.

(a) Entwickeln Sie den Code aus einem KV-Diagramm heraus. (Diagramm mit abgeben)

(b) Warum kann der Code aus (a) nicht zyklisch-einschrittig sein?

(c) Geben Sie einen zyklisch-einschrittigen Code mit 12 Codewörtern an. Benutzen Sie dazu das rekursive Verfahren aus der Vorlesung.