64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 6 Ausgabe: 20.11., Abgabe: 27.11. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 6.1 (Punkte 5+5+5)

UTF-8: Die ISO-8859-1 Codierung benutzt 8-bit für jedes enthaltene Zeichen. Die direkte Codierung der basic-multilingual Plane von Unicode (Java Datentyp `char`) verwendet pro Zeichen 16-bit, während die UTF-8 Codierung Vielfache von 8-bit benutzt.

(a) Wir betrachten einen deutschsprachigen Text mit insgesamt 500 000 Zeichen. Wir nehmen die folgenden Wahrscheinlichkeiten für die Umlaute an, andere Sonderzeichen kommen nicht vor: **Ä/ä** 0,56 %, **Ö/ö** 0,287 %, **Ü/ü** 0,616 % und **ß** 0,308 %.

Wie viele Bytes belegt dieser Text bei Codierung nach ISO-8859-1, in direkter Unicode Darstellung und in UTF-8?

(b) Im Unicode-Standard sind für die CJK-Symbole (chinesisch, japanisch, koreanisch) neben einem Basis-Zeichensatz noch mehrere Erweiterungen definiert. Ein chinesischer Text, mit insgesamt 500 000 Schriftzeichen, besteht aus den Basis-Zeichen (von `U+4E00` bis `U+9FFF`) und Symbolen der Extension-A (von `U+3400` bis `U+4DBF`).

Wie viele Symbole sind das? Berechnen Sie das Ergebnis direkt im Hexadezimalsystem und verwenden Sie dabei das 16-Komplement für die Subtraktion. Die Lösung muss mit Zwischenschritten abgeben werden.

(c) Wie viele Bytes belegt der chinesische Text bei direkter Unicode Darstellung und bei Codierung als UTF-8?

Aufgabe 6.2 (Punkte 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) $y = 6 \cdot x$

(b) $y = 30 \cdot x$

(c) $y = -56 \cdot (x + 4)$

Aufgabe 6.3 (Punkte 5+5+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) `bitNand(x,y)` Diese Funktion soll das bitweise NAND liefern: $\overline{x_i \wedge y_i}$. Als Operatoren dürfen nur | und ~ (OR, Negation) benutzt werden.

(b) `bitXnor(x,y)` Diese Funktion soll die XNOR-Verknüpfung (Äquivalenz) realisieren: $x_i \equiv y_i$. Als Operatoren dürfen nur | und ~ (OR, Negation) benutzt werden.

(c) `rotateLeft(x,n)` Die Funktion soll den in Java nicht vorhandenen Rotate-Left Operator für `x` nachbilden. Für das zweite Argument `n` gilt: $0 \leq \mathtt{n} \leq 31$.

Aufgabe 6.4 (Punkte 10+5)

Base-64 Codierung: Wie in der Vorlesung skizziert, werden bei der Base-64 Codierung jeweils drei 8-bit Eingangswerte durch vier 6-bit Ausgangswerte ersetzt, die dann zur Datenübertragung als (7-bit) ASCII-Zeichen codiert werden.

(a) Beschreiben Sie durch logische- und Schiebe-Operationen, wie bei der Decodierung aus den vier Eingabezeichen `a1`...`a4` (hier schon als Integer Zahlen), die drei 8-bit Ausgangswerte `b1`...`b3` berechnet werden. Vervollständigen Sie dazu die Ausdrücke `b`... im nachfolgenden Java-Code.

```
int a1, a2, a3, a4;          // vier Zeichen, Wertebereich je 0..63

int b1 = ?
int b2 = ?
int b3 = ?

...
```

(b) In dem ersten Aufgabenteil wurde angenommen, dass die Zeichen `a1`...`a4` schon in Zahlen von 0...63 umgesetzt worden sind. Skizzieren Sie, wie diese Konvertierung durchgeführt werden kann. Programmcode muss nicht geschrieben werden, es genügt wenn Sie textuell beschreiben, wie man das machen könnte.

Aufgabe 6.5 (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 „RSB“ 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 $3A76_{16}$ und 0522_{16} codiert?