10. Assignment

Input/Output

Issue: June 29—Due: July 6

Exercise 16: *RAID* 10 Points

Die Berechnung der einfachen Parität erlaubt die Erkennung eines einzelnen Bitfehlers innerhalb einer Binärsequenz. Die Korrektur eines Fehlers innerhalb einer Binärsequenz benötigt jedoch mehr Informationen, da die Position des "gekippten" Bits identifiziert werden muß, wenn es korrigiert werden soll. Mit einem einzelnen Paritätsbit ist dies nicht möglich, da jeder mögliche Bitfehler in jeder denkbaren Position innerhalb der Binärsequenz genau dieselbe Information erzeugt, nämlich einen Paritätsfehler.

Werden mehrere Paritätsbits für eine Binärsequenz gespeichert und werden diese Paritätsbits so angeordnet, so daß unterschiedliche gekippte Bits unterschiedliche Fehlerergebnisse erzeugen, so kann das gekippte Bit erkannt werden. In einer 7-Bit Nachricht z.B. können 7 mögliche einzelne Bitfehler auftreten, d.h., 3 Paritätsbits genügen, daß einerseits der Fehler erkannt wird und andererseits herausgefunden werden kann, welches Bit gekippt ist.

Das Hamming-Coding-Verfahren ist eine Technik, die ein einzelnes gekipptes Bit innerhalb einer Bitsequenz erkennen und korrigieren kann. Um Datenfehler zu erkennen und zu korrigieren, berechnet das Verfahren sogenannte "code words" anhand folgendem Algorithmus:

1. Markiere alle Bitposition welche gültige Zweier-Potenzen sind als Paritätsbits (Positionen 1, 2, 4, 8, 16, 32, 64, etc.).
2. Alle anderen Bitpositionen stehen für die eigentlichen Daten zur Verfügung (Positionen 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.).
3. Jedes Paritätsbit berechnet die Parität für ausgewählte Datenbits des Codewortes. Die Position des Paritätsbits bestimmt die Sequenz von Bits die dabei alternative geprüft bzw. übersprungen wird:

Position 1: überprüfe 1 Bit, überspringe 1 Bit, überprüfe 1 Bit, überspringe 1 Bit, etc. $(1, 3, 5, 7, 9, 11, 13, 15, \dots)$

Position 2: überprüfe 2 Bits, überspringe 2 Bits, überprüfe 2 Bits, überspringe 2 Bits, etc. $(2, 3, 6, 7, 10, 11, 14, 15, \dots)$

Position 4: überprüfe 4 Bits, überspringe 4 Bits, überprüfe 4 Bits, überspringe 4 Bits, etc. $(4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, \dots)$

Position 8: überprüfe 8 Bits, überspringe 8 Bits, überprüfe 8 Bits, überspringe 8 Bits, etc. $(8-15, 24-31, 40-47, \dots)$

Position 16: ...

4. Setze das Paritätsbit auf 1, wenn die Anzahl von Einsen in den jeweiligen Bitpositionen ungerade ist. Setze das Paritätsbit auf 0, wenn die Anzahl von Einsen in den jeweiligen Bitpositionen gerade ist.

Das folgende Beispiel illustriert den Algorithmus (bitte wenden):

```
8-Bit Binaersequenz: 10011010
Erzeugen des Codewortes, indem Platzhalter fuer die Paritaetsbits
eingefuegt werden: _ _ 1 _ 0 0 1 _ 1 0 1 0
Berechnung der Paritaet fuer jedes Paritaetsbit
(das Symbol ? repraesentiert die zu setzende Bitposition):

    - Position 1 ueberprueft Bits 1,3,5,7,9,11:
      ? _ 1 _ 0 0 1 _ 1 0 1 0. Gerade Paritaet, d.h., Paritaetsbit 1
      wird auf 0 gesetzt: 0 _ 1 _ 0 0 1 _ 1 0 1 0
    - Position 2 ueberprueft Bits 2,3,6,7,10,11:
      0 ? 1 _ 0 0 1 _ 1 0 1 0. Ungerade Paritaet, d.h.,  Paritaetsbit 2
      wird auf 1 gesetzt: 0 1 1 _ 0 0 1 _ 1 0 1 0
    - Position 4 ueberprueft Bits 4,5,6,7,12:
      0 1 1 ? 0 0 1 _ 1 0 1 0. Ungerade Paritaet, d.h.,  Paritaetsbit 4
      wird auf 1 gesetzt: 0 1 1 1 0 0 1 _ 1 0 1 0
    - Position 8 ueberprueft Bits 8,9,10,11,12:
      0 1 1 1 0 0 1 ? 1 0 1 0. Gerade Paritaet, d.h., Paritaetsbit 8
      wird auf 0 gesetzt: 0 1 1 1 0 0 1 0 1 0 1 0
    - Codewort: 011100101010.
```

Das Erkennen und die Korrektur von Bitfehlern erfolgt über eine Validierungsmatrix V. Für diese Matrix gilt, daß jeder $j-te$ Spaltenvektor jeweils die Binärdarstellung der Zahl $j \in 1, \ldots, n$, wobei $n \in \mathbb{N}$ die Länge des Codewortes repräsentiert.

Für das obige Beispiel mit 8 Daten- und 4 Paritätsbits ergibt sich die folgende Matrix:

$$V = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Die Position eines gekippten Bits innerhalb eines fehlerhaften Codewortes c erhält man dadurch, indem man das fehlerhaften Codewort mit der Validierungsmatrix V multipliziert.

Das nachfolgende Beispiel illustriert die Fehlererkennung:

Für die 8-Bit Binärsequenz `10011010` ist das zugehörige "korrekte" Codewort `011100101010`. Nehmt an, daß im gespeicherten Codewort c ein Bit gekippt ist und dieses daher wie folgt aussieht: `011100101110`. Bilden wir nun das Produkt von c und V, so erhalten wir den Spaltenvektor e, welcher die Position des gekippten Bits im Codewort anzeigt.

$$e = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Der Ergebnisvektor indiziert, daß das $10-te$ Bit fehlerhaft ist und von 1 auf 0 gesetzt werden muß.

Nach der Einführung nun die eigentliche Aufgabenstellungen:

1. In der Vorlesung wurde erwähnt, daß RAID-2 Systeme das Hamming-Coding-Verfahren zur Fehlererkennung und -vermeidung einsetzen. Wie viele Paritätsbits muß ein solches System mit 4 Datenfestplatten umfassen? Begründet Eure Antwort.

2. Bestimmt den Hamming-Code für die beiden 4-Bitsequenzen (nibbles) `0110` und `1010`.

3. Gegeben sind folgende Hamming-Codewörter, welche möglicherweise jeweils ein fehlerhaftes Bit beinhalten können:

   (a) `0100011`

   (b) `1111111`

   (c) `0110010`

   Überprüft diese Codewörter hinsichtlich ihrer Korrektheit und gebt gegebenenfalls die Position des fehlerhaften Bits an.