

## Rechnerarchitektur

**Abgabetermin:** 30.06.2008, 12:00 Uhr

**Lesen:**

### Aufgabe 32: (H) Fehlererkennung und Fehlerkorrektur (10 Pkt.)

Es soll das Speicherwort 0111 übertragen werden. Dazu wird es mit nachfolgenden Paritätsbits versehen, in der Struktur:  $m_1 m_2 m_3 m_4$  nachfolgende Paritätsbits. Es wird gerade Parität verwendet.

- Wie viele Paritätsbits (nachfolgende Paritätsbits) sind minimal nötig, um alle Einbitfehler zu korrigieren?
- Hängen Sie die Paritätsbits an das Speicherwort an. Dies ergibt das zu übertragende Codewort. Zeichnen Sie dazu ein Venn-Diagramm, in dem Sie wie in der Vorlesung die Paritätsbits und die Bits des Speicherworts in Mengen modellieren.
- Der Empfänger erhält fälschlicherweise das Code-Wort 0011001. Zeigen Sie anhand ihres Diagramms, wie der Fehler erkannt und korrigiert werden kann.
- Argumentieren Sie kurz, ob diese Darstellung mit Venn-Diagrammen für beliebige Codewörter sinnvoll erscheint.

### Aufgabe 33: (H) Arbeitsweise Caches (2+2+2+2+2 Pkt.)

Nehmen Sie einen Speicher mit 64 und einen Cache mit 16 Blöcken an. Wir benötigen nacheinander folgende Adress-Zugriffe bei anfangs leerem Cache:

1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

- Geben Sie für jede Referenz an, ob ein Cache-Hit oder ein Cache-Miss eintritt. Gehen Sie dabei von dem in der Vorlesung eingeführten Direktabbildungs-Verfahren aus.
- Stellen Sie den Inhalt des Caches nach allen Zugriffen dar.
- Wie viel Speicherplatz ist erforderlich, um einen Direct-mapped Cache zu realisieren, der 64 KByte Daten zwischenspeichern kann, wenn pro Cache-Block ein Validierungs-Bit und ein geeignetes Tag benötigt werden? (Annahme: 32 Bit Blockgröße)

**Aufgabe 34: (H) CASE-Anweisung**

(4 Pkt.)

Schreiben Sie ein MIPS-Programm zur Realisierung der folgenden CASE-Anweisung:

```

1  VAR i: INTEGER;
2
3  CASE i OF
4      1: k := 1; break;
5      2: k := 2; break;
6      3: k := 3; break;
7      4: k := 4; break;
8  ELSE: k := 999; break;
9  END;
```

Überlegen Sie sich dazu ein generelles Verfahren zur Implementierung komplexer CASE-Anweisungen.

*Hinweis:* Beachten Sie, dass die einzelnen CASE-Blöcke nicht gleich lang sein müssen.

**Aufgabe 35: (P) ASCII-nach-Integer-Konversion**

(10 Pkt.)

Schreiben sie ein MIPS-Programm zur Konversion eines ASCII-Strings aus Nummern in eine Ganzzahl. Lesen Sie die umzuwandelnde Zahl von der Konsole ein und geben Sie das Ergebnis aus.

*Hinweis:* Ihr Programm muß keine negativen Zahlen umwandeln können. Wenn der String einen Buchstaben enthält, der keine Ziffer ist, so soll das Programm abbrechen mit dem Wert  $-1$  in Register  $\$v0$ . Beachten Sie besonders, was genau als Eingabe verarbeitet wird (schauen Sie sich den Speicher an!).

**Aufgabe 36: (T) Hamming Codes**

(6+6 Pkt.)

Übertragung von Daten über physische Kanäle (Kabel etc.) ist fehleranfällig. Indem man ein einzelnes Bit, das *Paritätsbit*, zu jedem Datenpaket hinzufügt, kann man Ein-Bit-Fehler entdecken. Mit einem einzelnen Paritätsbit ist es allerdings nicht möglich herauszufinden welches Bit fehlerhaft ist. Wenn man also das fehlerhafte Bit erkennen möchte, benötigt man mehr Paritätsbits.

Das folgende Verfahren wurde mit dem Ziel Ein-Bit Datenfehler, die durch unzuverlässige Übertragungskanäle verursacht wurden, *zu erkennen und zu korrigieren*.

Der Trick besteht darin zusätzliche Paritätsbits in die Datenpakete einzufügen und so ein *Hamming Codewort* zu bilden. Das Hamming Codewort besteht aus den eigentlichen Datenbits, die übertragen werden sollen, und einigen Paritätsbits, die an strategischen Punkten eingefügt wurden.

Die Anzahl der benötigten Paritätsbits ist abhängig von der Anzahl der Datenbits:

Daten Bits :	8	16	32	64	128
Paritäts-Bits:	4	5	6	7	8
Codewort :	12	21	38	71	136 bits

Allgemein gilt: Für Daten den Länge  $2^n$  Bits werden  $n + 1$  Paritätsbits eingefügt, um das Codewort zu bilden.

**Algorithmus:**

1. Die Bits des Codeworts werden von 1 an nummeriert. Bit 1 (das am weitesten links stehende) ist das *most significant bit*.
2. Die Paritätsbits sind die Bits des Codeworts, deren Nummer eine Potenz von 2 ist, also 1,2,4,8, ... Alle anderen Bits sind Datenbits.
3. Jedes Paritätsbit prüft mehrere genau festgelegte Bits des Codeworts, sich selbst eingeschlossen. Ein Bit des Codewortes kann von mehr als einem Paritätsbit geprüft werden.  
Ein Bit B wird von den Paritätsbits  $P_1, P_2, P_3, \dots, P_k$  geprüft, wenn  $B = P_1 + P_2 + \dots + P_k$  ist.  
Beispiel: Bit 11 wird von den Paritätsbits 1,2 und 8 geprüft.

Parity Bit	Getestete Bits											
1 :	1	3	5	7	9	11	13	15	17	19	21	
2 :	2	3	6	7	10	11	14	15	18	19		
4 :	4	5	6	7	12	13	14	15	20	21		
8 :	8	9	10	11	12	13	14	15				
16 :	16	17	18	19	20	21						

4. Die Paritätsbits werden so gesetzt, dass die Summe der geprüften Bits (sich selbst eingeschlossen) gerade ist.

Beispiel:

Um das 8-Bit Datenwort 1001 0110 zu kodieren, benötigen wir vier Paritätsbits; das Codewort ist also 12 Bits lang und sieht folgendermaßen aus:

P P D P D D D P D D D D (P = Paritätsbit, D = Datenbit)

	1	2	3	4	5	6	7	8	9	10	11	12
Codewort :	?	?	1	?	0	1	1	?	0	1	1	0
Parity Bit 1:	?		1		0		1		0		1	? = 1 damit Summe gerade
Parity Bit 2:		?	1			1	1			1	1	? = 1 damit Summe gerade
Parity Bit 4:				?	0	1	1				0	? = 0 damit Summe gerade
Parity Bit 8:								?	0	1	1	0 ? = 0 damit Summe gerade
Codewort :	1	1	1	0	0	1	1	0	0	1	1	0

### Fehlererkennung:

- Um einen Fehler zu erkennen und zu korrigieren berechnet man die Checksumme für jedes Paritätsbit. Wenn alle Checksummen gerade (bzw. ungerade) sind, dann ist das Codewort fehlerfrei.
- Identifiziere alle Paritätsbits mit fehlerhaften Checksummen. Bilde die Schnittmenge aller von nicht korrekten Paritätsbits geprüften Bits. Eliminiere alle Bits, die auch von korrekten Paritätsbits geprüft werden. Das fehlerhafte Bit bleibt übrig.

Alternative Methode: Die Summe der Nummern der fehlerhaften Paritätsbits ergibt die Nummer des fehlerhaften Bits.

Beispiel:

Finde und korrigiere einen eventuell vorhandenen Fehler in dem Codewort 1100 0110 0110.

Dieses 12-Bit Codewort hat vier Paritätsbits, Bits Nummer 1,2,4 und 8.

	1	2	3	4	5	6	7	8	9	10	11	12
Codewort:	1	1	0	0	0	1	1	0	0	1	1	0
Parity Bit 1:	1		0		0		1		0		1	:3 X
Parity Bit 2:		1	0			1	1			1	1	:5 X
Parity Bit 4:				0	0	1	1					0 :2
Parity Bit 8:								0	0	1	1	0 :2

Die Prüfsummen der Bits 1 und 2 sind falsch. Die Schnittmenge der Bitlisten sind die Bits 3, 7 und 11, also ist eins von diesen fehlerhaft.

Bit 11 wird auch von Bit 8 geprüft, dessen Checksumme korrekt ist, also ist auch Bit 11 OK.

Bit 7 wird auch von Bit 4 geprüft, dessen Checksumme korrekt ist, also ist auch Bit 7 OK.

Bit 3 wird nur von den Bits 1 und 2 geprüft, also ist Bit 3 falsch.

Korrigiert:	1	1	1	0	0	1	1	0	0	1	1	0
8-Bit Daten:				1		0	1	1		0	1	0

### Aufgaben:

- a. Kodieren Sie die folgenden 8-Bit Daten in 12-Bit Hamming Codes:
- (i) 1010 1110
  - (ii) 0101 0001
- b. Dekodieren Sie die folgenden 12-Bit Codewörter. Wenn sie Fehler enthalten, identifizieren Sie das fehlerhafte Bit und korrigieren Sie den Fehler. Das Ergebnis muss ein 8-Bit Datenwort sein.
- (i) 1101 0110 0111
  - (ii) 1101 1110 0111