RA-Klausur SS 2010 I (Angabe)

von Tobias Munzert

1. Multiple Coice (20 Pkt.)

Darstellung von Informationen		Wahr	Falsch
a	Die Wortlänge des Systems hängt von der Datenbusbreite ab.		
b	Mit einer 2-Bit-Zahl in Einer-Komplement-Darstellung kann man genau 3 verschiedene ganze Zahlen darstellen.		
c	Jede vier Bit lange Binärzahl lässt sie durch eine Oktalziffer ausdrücken.		
d	Nach IEEE-Standard wird der Exponent im Zweierkomplement dargestellt.		
e	Um ein Schwarz/Weiß-Bild mit den Abmessungen m*n Pixel unkomprimiert zu speichern, benötigt man (aufgerundet) $\log_2 (m*n)$ Bit.		

Arithmetik		Wahr	Falsch
a	Bei der Addition von zwei positiven Binärzahlen kann kein Überlauf stattfinden.		
b	Ausgehend von einer festen Anzahl an Bits ist in der Zweierkomplement-Darstellung der Betrag der kleinsten negativen Zahl größer, als der der größten positiven.		
c	Bei Verendung einer Einerkomplement benötigt man zur Durchführung der Subtraktion ein Subtrahierwerk.		
d	Sei x_2 eine 8-stellige Binärzahl. Mit dem Einerkomplement berechnet man die Differenz des Betrags von $–x_2$ und 10000000_2.		
e	Um n m-stellige Dualzahlen zu addieren reichen m-2 Carry-Select-Addierbausteine aus.		

Boolesche Algebra		Wahr	Falsch
a	{NAND} ist funktional vollständig.		
b	Es gibt genau fünf verschiedene zweistellige Boolesche Operatoren.		
c	Jede Boolesche Funktion (mit beliebig vielen Stellen) kann mit dem Verfahren von Quine-McCluskey minimiert werden.		
d	Die boolschen Terme A + B * (A + B) + A * (-A + B) und A + B sind äquivalent.		
e	Es gibt maximal 2^n verschiedene n-stellige booleschen Funktionen.		

Schaltnetze		Wahr	Falsch
a	Alle Schaltkreise in einem Computer können ausschließlich mit NOR-Gattern gebaut werden.		
b	Gegeben sei ein SR-Latch mit der Belegung S=R=0. Setzt man den Eingang S=1, hängt der Zustand von Q von der Belegung von R ab.		
c	Das D-Latch beseitigt den Undeterminismus des SR-Latch.		
d	In einem normalen PLA kann die Und-Ebene einen oder mehrere Addierbausteine enthalten, wenn die zu realisierende Boolesche Funktion in DNF gegeben ist.		
e	Wenn die disjunkte Normalform (DNF) einer n-stelligen Booleschen Funktion ohne Don't-Care-Argumente aus m Termen besteht, dann besteht die konjunktive Normalform (KNF) dieser Funktion immer aus 2^n-m Termen.		

2. Zweierkomplement (14 Pkt.)

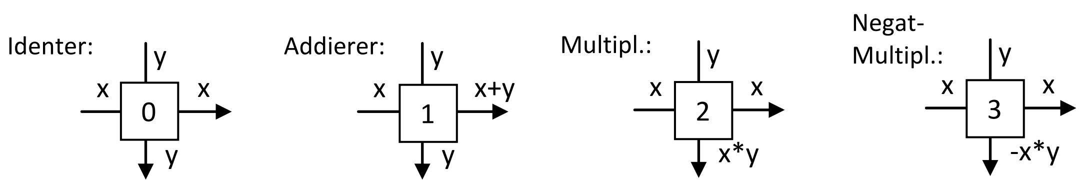
a) Bilden Sie von den Dezimalzahlen x=-73 und y=36 die 8-Bit-Zweierkompliment-Darstellung.
b) Stellen Sie da, wie die Subtraktion mit Zweierkompliment funktioniert.
c) Berechnen Sie (mit Rechenweg) z=x-y.
d) Begründen Sie ob bei c ein Overflow stattgefunden hat?
e) a=1011 1111
b=0100 0001
Begründung Sie ohne das Ergebnis zu berechnen, ob bei der Addition der beiden Zweierkompliment-Darstellungen ein Überlauf stattfindet?
f) Geben Sie die Darstellung von $-5/16_{10}$ als Gleitkommazahl nach IEEE 754 in einfacher (32-Bit) Genauigkeit mit Rechenweg an.

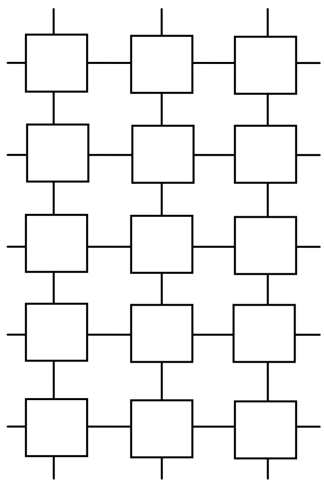
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	…
Wert																	…

…	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
…																

3. boolsche Funktionen durch normiertes PLA (10 Pkt.)

a) Stellen Sie h(a,b,c,d) = (-ab-cd) + (a-d) + (-bc) mittels Identer, Addierer, Multilizierer und Negat-Multiplizierer da.





b) Sei die Schaltfunktion f: B^n -> B^m gegeben.
   i) Geben Sie die Anzahl der Zeilen an, die man maximal benötigt, um diese Schaltfunktion durch ein PLA zu realisieren und begründen Sie Ihre Antwort.
   ii) Angenommen f liegt in disjunkter Form vor: Wovon hängt die Anzahl der Spalten des entsprechenden PLAs ab?

4. Hamming-Code (18 Pkt.)

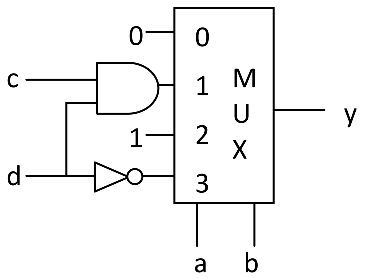
a) Was ist der Hamming-Abstand und wie groß muss er mindestens sein um d Einzelbitfehler erkennen bzw. korrigieren zu können?

b) Codieren sie das 8-Bit Datenwort 1110 1110 nach Hamming-Verfahren, geben Sie das Codewort an (verwenden Sie dazu gerade Parität) und kennzeichnen Sie die Paritätsbits.

c) Dekodieren Sie folgende 12-Bit-Codewörter. Korrigieren Sie diese wenn möglich. Verwenden Sie gerade Parität.
   i) 0101 1011 0000
   ii) 1011 0101 1110

5. Multiplexer (15 Pkt.)

a)



i) Geben Sie die Ausgabewerte von y = f(a,b,c,d) für alle möglichen Eingabewerte an:

a	b	c	d	y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

ii) Geben Sie y in disjunkter Normalform (DNF) an.

b) Gegeben sei $y(x_1x_2x_3x_4) = -x_1x_2-x_3x_4 + -x_1-x_2-x_3x_4 + -x_1-x_2-x_3-x_4 + x_1x_2-x_3-x_4 + x_1-x_2-x_3x_4 + x_1-x_2-x_3-x_4 + x_1x_2x_3x_4 + x_1-x_2x_3-x_4$

Vereinfachen sie $y(x_1x_2x_3x_4)$ mittels Karnaugh-Diagramm.

$X_3x_4 \backslash x_1x_2$	00	01	11	10
00				
01				
11				
10				

6. Pipeline (13 Pkt.)

a) Benennen und erklären Sie verschiedene Arten von Konflikten (Hazards), die durch die Einführung von Pipelining entstehen können. Geben Sie ein Bsp. Für diejenigen Hazards an, die bei der Mips-Architektur auftreten können.

b) 5-stufiges Pipelining
Ausführungszeit: 2ns pro Stufe;

Befehl	IF	ID	EX	MEM	WB
load word (lw)	2ns	1ns	2ns	2ns	1ns
add	2ns	1ns	2ns	-	1ns
Branch (beq)	2ns	1ns	2ns	-	-

```
lw      $2, 100 ($5)
add     $3, $3, $4
add     $1, $4, $5
```

i) Geben Sie die Dauer des Programms mit Pipelining an.
ii) Geben Sie die Dauer des Programms ohne Pipelining an.

c) Delayed Branch
Ein Pipeline-Stall soll nach dem Branch-Befehl vermieden werden, die Semantik aber nicht verändert werden. (manipulieren bzw. modifizieren)

```
sw      $2, 100 ($3)
addi    $3, $3, $4
add     $4, $4, $2
beq     $2, $3, 200
```

7. Mips – Binomialkoeffizient (30 Pkt.)

Binomialkoeffizient: $\binom{n}{k} = \frac{n!}{k!\,(n-k)!} = \begin{cases} Fehler\ wenn\ k > n \\ Fehler\ wenn\ k = 0 \\ \frac{n(n-1)(n-2)\ldots(n-k+1)}{k(k-1)(k-2)\ldots 1} \end{cases}$

a) Schreiben Sie ein Mips-Assembler-Programm das für n und k den Binomialkoeffizienten auf die Konsole ausgibt.
Hinweise: - Nenner und Zähler getrennt, aber in der selben Schleife berechnen;
- die Division als letzten Schritt durchführen;
- Grenzfälle beachten; z.B.: k=0, n=0;
- Fehlermeldung für n<k

b) Eine sehr ineffiziente Art den Binomialkoeffizienten zu berechnen besteht darin, zunächst die drei Fakultätsterme n!, k! und (n-k)! zu berechnen und diese dann der obrigen Formel entsprechend zu kombinieren.
Die Fakultätsfunktion fac(n)=n! ist bekanntlich folgendermaßen definiert:

Fakultätsfunktion: $fac(n) = \begin{matrix} n * fac(n-1) : n \geq 1 \\ 1 : n = 0 \end{matrix}$

Das Mips-Programm, das sich auf dem Din-A3- Bogen *(der uns leider nicht vorliegt)* befindet, realisiert die rekursive Berechnung der Fakultätsfunktion mit Hilfe des Stacks. Tragen Sie den Zustand des Stacks in die Tabelle ein …

Befehl	Argumente	Wirkung
add	Rd, Rs1, Rs2	Rd := Rs1 + Rs2
addu	Rd, Rs1, Rs2	Rd := Rs1 + Rs2
addi	Rd, Rs1, Imm	Rd := Rs1 + Imm
addiu	Rd, Rs1, Imm	Rd := Rs1 + Imm
div	Rd, Rs1, Rs2	Rd := Rs1 DIV Rs2
rem	Rd, Rs1, Rs2	Rd := Rs1 MOD Rs2
mul	Rd, Rs1, Rs2	Rd := Rs1 $\times$ Rs2
b	label	unbedingter Sprung nach label
j	label	unbedingter Sprung nach label
jal	label	unbed.Sprung nach label, Adresse des nächsten Befehls in $ra
jr	Rs	unbedingter Sprung an die Adresse in Rs
beq	Rs1, Rs2, label	Sprung, falls Rs1 $=$ Rs2
beqz	Rs, label	Sprung, falls Rs $=$ 0
bne	Rs1, Rs2, label	Sprung, falls Rs1 $\neq$ Rs2
bnez	Rs1, label	Sprung, falls Rs1 $\neq$ 0
bge	Rs1, Rs2, label	Sprung, falls Rs1 $\geq$ Rs2
bgeu	Rs1, Rs2, label	Sprung, falls Rs1 $\geq$ Rs2
bgez	Rs, label	Sprung, falls Rs $\geq$ 0
bgt	Rs1, Rs2, label	Sprung, falls Rs1 $>$ Rs2
bgtu	Rs1, Rs2, label	Sprung, falls Rs1 $>$ Rs2
bgtz	Rs, label	Sprung, falls Rs $>$ 0
ble	Rs1, Rs2, label	Sprung, falls Rs1 $\leq$ Rs2
bleu	Rs1, Rs2, label	Sprung, falls Rs1 $\leq$ Rs2
blez	Rs, label	Sprung, falls Rs $\leq$ 0
blt	Rs1, Rs2, label	Sprung, falls Rs1 $<$ Rs2
bltu	Rs1, Rs2, label	Sprung, falls Rs1 $<$ Rs2
bltz	Rs, label	Sprung, falls Rs $<$ 0
syscall		führt Systemfunktion aus
move	Rd, Rs	Rd := Rs
la	Rd, label	Adresse des Labels wird in Rd geladen
lb	Rd, Adr	Rd := MEM[Adr]
lw	Rd, Adr	Rd := MEM[Adr]
li	Rd, Imm	Rd := Imm
sw	Rs, Adr	MEM[Adr] := Rs

Funktion	Code in $v0	Funktion	Code in $v0
print_int	1	read_float	6
print_float	2	read_double	7
print_double	3	read_string	8
print_string	4	sbrk	9
read_int	5	exit	10

Bemerkung:
Alle arithmetischen Befehle (mul, ...), Sprungbefehle und Vergleichsbefehlt sind auch mit einem Imm-Argument (Immediate-Argument) statt Rs2 möglich.Zum Beispiel: $t0, $t1, 5 statt li $t5, 5 gefolgt von mul $t0, $t1, $t5. Dies funktioniert, da der Assembler dann den Wert zunächst in sein $at-Register läd.