Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

mobile and distributed systems group

Rechnerarchitektur im Sommersemester 2019
Übungsblatt 6

Abgabetermin: 10.06.2019, 12:00 Uhr

Besprechung: Besprechung der T-Aufgaben in den Tutorien vom 03. – 07. Juni 2019
Besprechung der H-Aufgaben in den Tutorien vom 10. – 14. Juni 2019

Aufgabe 29: (T) Wiederholung: Minimierung mittels Karnaugh

Es sei die boolsche Funktion $g(x_1, x_2, x_3, x_4)$ durch die folgende Funktionstabelle gegeben.

	x_1	x_2	x_3	x_4	$g(x_1, x_2, x_3, x_4)$
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

a. Geben Sie die DNF von g an.

b. Minimieren Sie die Funktion g mittels Karnaugh und geben Sie die minimierte Funktion an.

Aufgabe 30: (T) Parameterübergabe bei Unterprogrammaufrufen

(– Pkt.)

Für die Parameterübergabe bei Prozeduraufrufen existieren verschiedene Möglichkeiten.

a. Erläutern Sie zunächst die Begriffe *call by value* und *call by reference*. Geben Sie zu beiden Konzepten je ein Beispiel in einer Hochsprache an.

b. Schreiben Sie nun ein SPIM-Programm, das den Durchschnitt der Werte eines Feldes berechnet. Die Berechnung selbst soll dabei ein Unterprogramm erledigen. Die Übergabe des Feldes soll nach dem Konzept *call by value* erfolgen.

   Achtung: Das Hauptprogramm soll dem Unterprogramm **alle** zur Berechnung notwendigen Werte über den Stack zur Verfügung stellen! Sie dürfen bei Ihrer Implementierung davon ausgehen, dass sich das Feld bereits im Speicher befindet.

c. Schreiben Sie Ihr Programm aus Aufgabe b) so um, dass die Übergabe des Feldes nach dem Konzept *call by reference* funktioniert.

   Achtung: Das Hauptprogramm soll dem Unterprogramm **ausschließlich** Speicheradressen zur Berechnung zur Verfügung stellen! Sie dürfen wieder davon ausgehen, dass sich das Feld bereits im Speicher befindet. Sie dürfen zur Übergabe der Adressen an das Unterprogramm die laut Konvention dafür vorgesehenen Register $a0 - $a3 verwenden. Das Ergebnis des Unterprogrammaufrufes dürfen Sie dem Hauptprogramm über das Register $v0 zur Verfügung stellen.

Aufgabe 31: (H) Assemblerprogrammierung unter SPIM

(6 Pkt.)

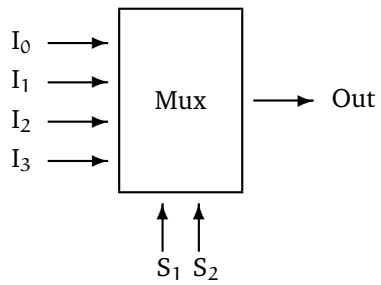
Im Folgenden soll ein MIPS-Assembler Programm vervollständigt werden, welches als Nutzereingabe eine 8-stellige Binärzahl von der Konsole entgegennimmt, sie in ihre dezimale Repräsentation umwandelt und diese auf der Konsole ausgibt. Die Eingabe wird vom Programm als String entgegengenommen. Danach soll in einer Schleife über die Zeichen diese Strings iteriert und geprüft werden, ob es sich beim aktuellen Zeichen um eine „0“ oder eine „1“ handelt und die entsprechende Wertigkeit aufsummiert werden. Zudem muss der Fall behandelt werden, dass das Ende des Strings, welches durch ein Byte mit dem Zahlenwert `0` markiert ist, erreicht wurde. Beachten Sie, dass der Sting nach 8 von der Konsole gelesen Zeichen automatisch übernommen und Null-terminiert wird. Dementsprechend ist kein Zeilenumbruch enthalten.
Laden sie sich die `binarytodecimal.s` von der Homepage herunter und ergänzen Sie den dort angegebenen Coderahmen um insgesamt **6 Zeilen Code**, so dass das Programm wie beschrieben funktioniert. Tragen Sie Ihre Lösung unter den mit “`# Ihre Loesung:`” markierten Stellen direkt in den Coderahmen der heruntergeladenen Datei ein.

Aufgabe 32: (H) Multiplexer

(6 Pkt.)

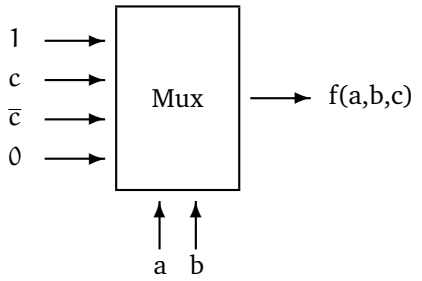
Für einem 4-Eingaben-Multiplexer gilt folgende verkürzte Funktionstabelle:

S_1	S_2	Out
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



Mit Hilfe eines 4-Eingaben-Multiplexers kann die Boolesche Funktion $f(a, b, c)$ dargestellt werden, indem dessen Eingänge bzw. Steuerleitungen wie folgt belegt werden.

	a	b	c	f(a, b, c)
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0



Geben Sie analog zum Beispiel eine Belegung der Eingänge eines 4-Eingaben-Multiplexers $(I_0, \ldots, I_3)$ sowie der Steuerleitungen S_1 und S_2 an, so dass dieser die Boolesche Funktion

$$g(a, b, c) = (\overline{a} \cdot \overline{b} \cdot \overline{c}) + (\overline{a} \cdot b \cdot \overline{c}) + (\overline{a} \cdot b \cdot c) + (a \cdot \overline{b} \cdot c) + (a \cdot b \cdot c)$$

realisiert.
Sie dürfen ausschließlich die Werte a, b, c, $\overline{c}$ sowie 0 und 1 benutzen. Es dürfen keine weiteren Bausteine oder Gatter verwendet werden.

Aufgabe 33: (H) Einfachauswahlaufgabe: Wiederholung

(5 Pkt.)

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwählen („1 aus n“). Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

a) Was bewirkt der Spim-Befehl `li $v0 5`:			
(i) Der Wert 5 wird in das Register `$v0` geladen	(ii) Es wird ein Integer von der Konsole eingelesen	(iii) Es wird eine Zahl vom Typ double von der Konsole eingelesen	(iv) Es wird ein Integer auf der Konsole ausgegeben
b) Welche Aussage ist korrekt? MIPS ist eine...			
(i) Stack-Architektur	(ii) Load-Store-Architektur	(iii) Heap-Architektur	(iv) Last-in-First-Out-Architektur
c) Welche Aussage ist falsch? Die Funktion `syscall`...			
(i) führt eine Funktion des Betriebssystems aus	(ii) besitzt selbst keine Parameter	(iii) erwartet die Nummer der auszuführenden Funktion in `$v0`	(iv) beendet das Programm sofort
d) Bei welcher Belegung (x_1, x_2, x_3) ergibt die Boolesche Funktion $f(x_1, x_2, x_3) = (x_1 \cdot \overline{x_2}) + (x_2 \cdot \overline{x_3})$ den Wert 1?			
(i) $(0, 1, 0)$	(ii) $(0, 1, 1)$	(iii) $(1, 1, 1)$	(iv) $(0, 0, 1)$
e) Wofür steht CISC im Zusammenhang mit Mikroprozessoren?			
(i) Controversy Instruction Set Computer	(ii) Complex Instruction Set Calculator	(iii) Constructive Instruction Set Computer	(iv) Complex Instruction Set Computer

Überblick über die wichtigsten SPIM Assemblerbefehle		
Befehl	Argumente	Wirkung
add	Rd, Rs1, Rs2	Rd := Rs1 + Rs2 (mit Überlauf)
sub	Rd, Rs1, Rs2	Rd := Rs1 - Rs2 (mit Überlauf)
addu	Rd, Rs1, Rs2	Rd := Rs1 + Rs2 (ohne Überlauf)
subu	Rd, Rs1, Rs2	Rd := Rs1 - Rs2 (ohne Überlauf)
addi	Rd, Rs1, Imm	Rd := Rs1 + Imm
addiu	Rd, Rs1, Imm	Rd := Rs1 + Imm (ohne Überlauf)
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
not	Rd, Rs1	Rd := $\neg$ Rs1 (bitweise Negation)
and	Rd, Rs1, Rs2	Rd := Rs1 & Rs2 (bitweises UND)
or	Rd, Rs1, Rs2	Rd := Rs1 | Rs2 (bitweises ODER)
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 (Speichere ein Wort)
sh	Rs, Adr	MEM[Adr] MOD 2^{16} := Rs (Speichere ein Halbwort)
sb	Rs, Adr	MEM[Adr] MOD 256 := Rs (Speichere ein Byte)

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