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

mobile and distributed systems group

Übungsblatt 8
Rechnerarchitektur im SoSe 2020

Zu den Modulen K

Besprechung: Besprechung der Übungsaufgaben in den Übungsgruppen vom 22. – 26. Juni 2020

Aufgabe Ü15: Binomialkoeffizient

(– Pkt.)

Der Binomialkoeffizient dient dazu, die Anzahl k-elementiger Teilmengen einer n-elementigen Menge zu berechenen. Die etwas vereinfachte Definition des Binomialkoeffizienten laute folgendermaßen:

$$\binom{n}{k} := \frac{n!}{k!(n-k)!} = \begin{cases} \textit{Fehler} & \text{wenn } k > n \\ 1 & \text{wenn } k = 0 \\ \frac{n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-k+1)}{k\cdot(k-1)\cdot(k-2)\cdot\ldots\cdot 1} & \text{sonst} \end{cases}$$

Schreiben Sie ein MIPS-Assembler-Programm, welches den Binomialkoeffizienten entsprechend der obigen Definition berechnet. Das Programm soll dazu die zwei Werte n und k über die Konsole einlesen und das Ergebnis der Berechnung wieder auf der Konsole ausgeben.
Halten Sie sich an die folgenden Vorgaben:
Berechnen Sie den Zähler und den Nenner getrennt, aber in ein und derselben Schleife (Hinweis: Verwenden Sie dazu die ausgeschriebene Form und nicht die mit den Fakultätstermen!) Führen Sie die Division als letzten Schritt durch! Werden zwei Zahlen mit $n < k$ eingegeben, so soll eine Fehlermeldung generiert werden. Das Programm soll auch Grenzfälle, z.B. $\binom{0}{0}$, richtig berechnen.
Kommentieren Sie Ihr Programm ausführlich!

Aufgabe Ü16: Addition mit doppelter Genauigkeit

(– Pkt.)

Ein MIPS-Register hat eine Breite von 32 Bit. Dennoch ist es möglich die Addition von Integerzahlen so zu realisieren, dass man eine doppelte Genauigkeit erreichen kann.
Skizzieren Sie die beiden *minimalen* Abfolgen von MIPS-Befehlen, die notwendig sind die Integeraddition mit doppelter Genauigkeit zu realisieren. Die eine Abfolge soll Überläufe ignorieren, die andere soll im Falle eines Überlaufs eine Unterbrechung provozieren. Nehmen Sie dazu an, dass die eine 64-Bit Ganzzahl (im Zweierkomplement) in den Registern `$t4` und `$t5`, die andere in den Registern `$t6` und `$t7` abgelegt sind. Die Summe der beiden Zahlen soll in die Register `$t2` und `$t3` geschrieben werden. Nehmen Sie weiterhin an, dass das höchstwertige Wort der 64-Bit Ganzzahl in den geraden, das niedrigstwertige Wort in den ungeraden Registern liegt.
Kommentieren Sie jede Zeile ihrer Programmfragmente ausführlich!
Hinweis: Es genügen vier Instruktionen.

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
`sltu`	`Rd, Rs1, Rs2`	Rd := 1 if Rs1 < Rs2 else 0; Rs1, RS2 sind unsigned integers
`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