Christian-Albrechts-Universität zu Kiel
Institut für Informatik

Lehrstuhl für Technische Informatik
Prof.Dr. Manfred Schimmler
Dr.-Ing. Christoph Starke
M.Sc. Vasco Grossmann
Dipl.-Inf. Johannes Brünger
M.Sc Sascha Clausen

Computersysteme

Wintersemester 2017/2018

Serie 10

Ausgabetermin: Freitag, 12.1.2018
Abgabetermin: Freitag, 26.1.2018, 08:00 Uhr im Schrein

Bitte klammern oder heften Sie Ihre Abgabeblätter geeignet zusammen und notieren Sie sowohl Ihre Namen als auch Ihre Gruppennummer auf der Abgabe!

Wichtiger Hinweis:
Bei jeder Assembler-Programmieraufgabe gilt, auch wenn nicht ausdrücklich dazu aufgefordert wird:

(a) **Beschreiben Sie Ihr Programm ausführlich und geben Sie separat die Registerbelegung an.**

(b) **Kommentieren Sie Ihr Programm ausführlich.**

(c) **Sie können die folgende URL benutzen, um Ihre Lösungen zu prüfen: https://huesersohn.github.io/dlx/**

(d) **Fertigen Sie zusätzlich zu Ihrer schriftlichen Abgabe im Schrein für jede Assembler-Programmieraufgabe eine Text-Datei an, die Ihre Lösung enthält, und senden Sie diese bis zum angegebenen Abgabetermin per E-Mail an den Hiwi, der ihre Abgaben korrigiert.**

Präsenzaufgaben

Aufgabe 1

Schreiben Sie ein Assemblerprogramm, welches die Fakultät $n!$ für ein $n \in \mathbb{N}$ berechnet.
Gehen Sie dabei davon aus, dass n in Register R1 steht und so gewählt ist, dass

$$n! \leq 2^{31} - 1$$

gilt. Das Ergebnis soll im Speicher an die Adresse 2000 geschrieben werden.

Aufgabe 2

Seien $m, n \in \mathbb{N}$. Schreiben Sie ein Assemblerprogramm, welches das kleinste gemeinsame Vielfache (kgV) von m und n berechnet. Die Zahl m steht in Register R1, die Zahl n steht in Register R2. Sie können davon ausgehen, dass für m und n gilt

$$kgV(m, n) \leq 2^{31} - 1$$

.
Das Ergebnis soll im Speicher an die Adresse 2000 geschrieben werden.

Hausaufgaben

Aufgabe 1

Ergänzen Sie für die beiden nachfolgenden DLX-Assemblerprogramme jeweils Registerbelegung, Kommentare und eine Programmbeschreibung.

(a) Programm 1:

```
        LW      R1, 1024(R0)
        ADDI    R2, R0, #1
        ADD     R4, R0, R0
loop:   AND     R5, R2, R1
        XOR     R4, R4, R5
        SRL     R1, R1, R2
        BNEZ    R1, loop
        SW      1028(R0), R4
        HALT
```

(b) Programm 2:

```
start:  LW      R1, 2020(R0)
        ADD     R2, R0, R0
        ADD     R5, R0, R0
loop:   LW      R3, 1000(R2)
        ADDI    R2, R2, #4
        SEQ     R4, R1, R3
        BEQZ    R4, loop2
        ADDI    R5, R5, #1
loop2:  SEQI    R4, R2, #1000
        BEQZ    R4, loop
end:    SW      2000(R0), R5
        HALT
```

10, 20 Punkte

Aufgabe 2

Die alten Ägypter multiplizierten zwei natürliche Zahlen m und n nach folgendem Verfahren:
Zunächst wird das Zwischenergebnis Erg auf 0 gesetzt. Solange $n > 0$ ist, werden anschließend folgende Schritte wiederholt:
a) falls n ungerade ist, bleibt m gleich, Erg wird um m vergrößert und n um eins reduziert.
b) falls n gerade ist, wird m verdoppelt, n halbiert und Erg bleibt gleich.
Wenn $n = 0$ ist, wird das Verfahren beendet und Erg ist das Ergebnis.
Schreiben Sie ein Assemblerprogramm, das zwei natürliche Zahlen entsprechend dem hier beschriebenen Verfahren multipliziert. Die Faktoren sollen dabei aus dem Speicher ab Adresse 1000 gelesen und das Ergebnis an die nächst größere Speicheradresse geschrieben werden. Sie können dabei davon ausgehen, dass die Faktoren und das Ergebnis kleiner als 2^{30} sind.

30 Punkte

Aufgabe 3

Die Fibonacci-Folge $(1; 1; 2; 3; 5; 8; ...)$ ist eine mathematische Folge ganzer Zahlen, bei der die ersten beiden Folgenglieder 1 sind und alle weiteren Folgenglieder sich aus der Summe der beiden jeweils vorhergehenden Folgenglieder ergeben. Für die n-te Fibonacci-Zahl f_n, wobei $n \in \mathbb{N}$, gilt also $f_n = f_{n-1} + f_{n-2}$, wobei $f_1 = 1$ und $f_2 = 1$.

Schreiben Sie ein Assemblerprogramm, das alle Fibonacci-Zahlen $< 2^{31}$ in aufsteigender Reihenfolge in den Speicher ab Adresse 1000 schreibt. Achten Sie dabei auf einen korrekten Umgang mit dem Zahlenbereich, insbesondere auch darauf, dass kein Element der Fibonacci-Folge ungewollt als negative Zahl interpretiert wird. In Register R1 soll am Ende das zum letzten gespeicherten Folgenglied zugehörige n stehen, also dasjenige n, für das gilt: $f_n < 2^{31} \leq f_{n+1}$.

30 Punkte

Aufgabe 4

(a) Wie viele Steuereingänge braucht ein Datenwegschalter mit 16 Dateneingängen?

(b) Geben Sie die größte und die kleinste Zahl an, die sich binär im 2-Komplement mit 6 Bit darstellen lässt.

(c) Geben Sie eine allgemeine Formel für die Anzahl der Eingänge eines J-auf-1 Mutliplexer an.

(d) Was versteht man in der IEEE-Gleitkommadarstellung unter einer „Hidden-One“? Wann kommt eine „Hidden-Zero“ vor?

2, 2, 3, 3 Punkte

Anhang: DLX-Assembler Befehlssatz

Die Befehle werden in der Form *Instr. / Ziel / Quelle(n)* verwendet.
Bsp: ADDI R3 R2 #15 ≈ R3:=R2+15

Instr.	Description	Format	Operation (C-style coding)
ADD	add	R	Rd = Rs1 + Rs2
ADDI	add immediate	I	Rd = Rs1 + extend(immediate)
AND	and	R	Rd = Rs1 & Rs2
ANDI	and immediate	I	Rd = Rs1 & extend(immediate)
BEQZ	branch if equal to zero	I	PC += (Rs1 == 0 ? 4 + extend(immediate))
BNEZ	branch if not equal to zero	I	PC += (Rs1 != 0 ? 4 + extend(immediate))
J	jump	J	PC += 4 + extend(immediate)
JAL	jump and link	J	R31 = PC + 4 ; PC += 4 + extend(immediate)
JALR	jump and link register	I	R31 = PC + 4 ; PC = Rs1
JR	jump register	I	PC = Rs1
LW	load word	I	Rd = MEM[Rs1 + extend(immediate)]
MULT	Mult	R	Rd = Rs1 * Rs2
OR	or	R	Rd = Rs1 | Rs2
ORI	or immediate	I	Rd = Rs1 | extend(immediate)
SEQ	set if equal	R	Rd = (Rs1 == Rs2 ? 1 : 0)
SEQI	set if equal to immediate	I	Rd = (Rs1 == extend(immediate) ? 1 : 0)
SLE	set if less than or equal	R	Rd = (Rs1 <= Rs2 ? 1 : 0)
SLEI	set if less than or equal to immediate	I	Rd = (Rs1 <= extend(immediate) ? 1 : 0)
SLL	shift left logical	R	Rd = Rs1 << (Rs2 % 32)
SLLI	shift left logical immediate	I	Rd = Rs1 << (immediate % 32)
SLT	set if less than	R	Rd = (Rs1 < Rs2 ? 1 : 0)
SLTI	set if less than immediate	I	Rd = (Rs1 < extend(immediate) ? 1 : 0)
SNE	set if not equal	R	Rd = (Rs1 != Rs2 ? 1 : 0)
SNEI	set if not equal to immediate	I	Rd = (Rs1 != extend(immediate) ? 1 : 0)
SRA	shift right arithmetic	R	as SRL & see below
SRAI	shift right arithmetic immediate	I	as SRLI & see below
SRL	shift right logical	R	Rd = Rs1 >> (Rs2 % 32)
SRLI	shift right logical immediate	I	Rd = Rs1 >> (immediate % 32)
SUB	subtract	R	Rd = Rs1 - Rs2
SUBI	subtract immediate	I	Rd = Rs1 - extend(immediate)
SW	store word	I	MEM[Rs1 + extend(immediate)] = Rs2
XOR	exclusive or	R	Rd = Rs1 ^ Rs2
XORI	exclusive or immediate	I	Rd = Rs1 ^ extend(immediate)

Beachten Sie: Die Befehle SRA und SRAI füllen die vorderen Bits des Registers mit dem aktuellen Vorzeichenbit auf.
Ergänzung: Die Befehle HALT oder TRAP #0 beenden das Programm.

11.1.2018