Christian-Albrechts-Universität zu Kiel
Institut für Informatik

Multimedia Information Processing Group
Prof. Dr. Reinhard Koch
Dr.-Ing. Christoph Starke, Dr.-Ing. Vasco Grossmann
M.Sc. Sascha Clausen

Computersysteme

Wintersemester 2018/2019

Serie 11

Ausgabetermin: Freitag, 11.01.2019
Abgabetermin: Freitag, 25.01.2019, 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) **Auf der letzten Seite befindet sich ein Befehlssatz für den DLX-Assembler. Verwenden Sie für Ihr Programm ausschließlich die hier aufgelisteten Instruktionen.**

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

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

(d) **Sie können die folgende URL benutzen, um Ihre Lösungen zu prüfen:**
`http://huesersohn.de/cau/dlx/`

(e) **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 Ihre/n Korrektor/in.**

Präsenzaufgaben

Aufgabe 1

In Register R1 stehe der Wert 312, in Register R2 der Wert 492. Schreiben Sie drei verschiedene DLX-Befehle auf, mit denen das Wort an Speicherstelle 492 in das Register R3 geladen wird.

Aufgabe 2

Gegeben sei ein ganzzahliger Zahlenwert (32 Bit) in R1 und ein weiterer in R2. Schreiben Sie ein DLX-Assemblerprogramm, das die Inhalte dieser beiden Register vertauscht.

Aufgabe 3

Schreiben Sie ein DLX-Assemblerprogramm, welches den Inhalt des Registers R2 modulo 15 berechnet und im Register R3 ablegt. Der Inhalt von R2 soll dabei nicht verändert werden. Der Befehl MULT darf nicht verwendet werden.

Aufgabe 4

Gegeben sei das folgende DLX-Assemblerprogramm:

```
        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
```

Geben Sie in einem kurzen Satz an, was das Programm berechnet.
Auch hier sind die oben genannten Hinweise zu beachten.

Hausaufgaben

Aufgabe 1

Schreiben Sie ein Assemblerprogramm, das eine Folge von ASCII-Zeichen mit variabler Länge von Großbuchstaben (A bis Z) zu Kleinbuchstaben (a bis z) konvertiert. Sie brauchen Umlaute und ß dabei nicht zu berücksichtigen. Den verwendeten ASCII-Standard entnehmen Sie bitte dem Vorlesungsskript (Seite 25). Vereinfachend wird kein Paritätsbit verwendet, so dass das MSB für jedes kodierte Zeichen stets 0 ist. Der Buchstabe `C` wird also beispielsweise durch den binären Wert `01000011`, der Buchstabe `D` durch `01000100` kodiert.

Nehmen Sie an, dass die zu konvertierende ASCII-Folge ab Adresse 1000 byteweise gespeichert ist. Ihre Ziel-ASCII-Folge soll ab Adresse 2000 starten. R1 beinhaltet die Länge l der zu konvertierenden Folge. Sie können annehmen, dass $l < 1000$ ist. Zum Beispiel wird “STaRt” zu “start”.

30 Punkte

Aufgabe 2

Gegeben sei das folgende DLX-Assemblerprogramm:

```
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
```

Geben Sie in einem kurzen Satz an, was das Programm berechnet.
Beachten Sie auch hier die oben genannten Hinweise.

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. Die n-te Fibonacci-Zahl $f(n)$, wobei $n \in \mathbb{N}$, lässt sich also folgendermaßen berechnen:

$f(1) = 1$
$f(2) = 1$
$f(n) = f(n-1) + f(n-2)$, falls $n > 2$

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) Der DLX-Befehl *SRA* entspricht welcher mathematischen Funktion?

(b) Welchen dezimalen Wertebereich haben die *immediate*-Zahlen in unseren DLX-Befehlen?

(c) Wie viele Bits und wie viele Bytes werden bei Verwendung des DLX-Befehls *LW* gelesen und warum werden bei der Verwendung des DLX-Befehls *SW* normalerweise nur durch 4 teilbare Adressen verwendet?

(d) Welchen Wert schreibt der *JAL*-Befehl ins Register R31?

$2\frac{1}{2}$, $2\frac{1}{2}$, $2\frac{1}{2}$, $2\frac{1}{2}$ 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