Christian-Albrechts-Universität zu Kiel
Institut für Informatik

Lehrstuhl für Technische Informatik
Prof.Dr. Manfred Schimmler
Dr.-Ing. Christoph Starke, M.Sc. Vasco Grossmann


# Digitale Systeme
Wintersemester 2014/2015

# Serie 11

Ausgabetermin: Mittwoch, 21.01.2015
Abgabetermin: Montag, 02.02.2015, 08:00 Uhr im Schrein

**Bitte notieren Sie Ihre Namen sowie 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:**
`http://huesersohn.de/cau/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 Ihre/n Übungsleiter/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

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
```

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

# Hausaufgaben

## Aufgabe 1 - Kurzfragen

(a) Wie muss man einen Addierer verändern, so dass man damit Subtraktionen ausführen kann?

(b) Was ist eine „hidden one“?

$2^1/_2$, $2^1/_2$ Punkte

## Aufgabe 2

Das kleinste gemeinsame Vielfache (kgV) zweier positiver ganzer Zahlen kann berechnet werden, indem man die Vielfachen beider Zahlen miteinander vergleicht und von denen, die sich gleichen, das kleinste auswählt. Dazu bietet es sich an, die kleinere der beiden Zahlen so lange aufzuaddieren, bis sie der größeren Zahl gleicht oder größer ist als diese. In letzterem Fall wird die größere Zahl einmal zu sich selbst addiert und anschließend wird die kleinere Zahl weiter aufaddiert, bis sie wieder entweder größer oder gleich der größeren Zahl ist. Dieses Verfahren wird so lange angewendet, bis eine Zahl gefunden wurde, die sowohl durch das Aufaddieren der kleineren Zahl als auch durch das Aufaddieren der größeren Zahl entsteht.

Beispiel: kgV von 35 und 49

| *kleinere Zahl* | *größere Zahl* |
|---|---|
| 35 | 49 |
| 70 | 49 |
| 70 | 98 |
| 105 | 98 |
| 105 | 147 |
| 140 | 147 |
| 175 | 147 |
| 175 | 196 |
| 210 | 196 |
| 210 | 245 |
| 245 | 245 |

Schreiben Sie ein DLX-Assemblerprogramm, das das kgV zweier positiver ganzer Zahlen berechnet.
Zu Beginn der Programmausführung stehen die beiden 32-Bit-Zahlen, deren kgV berechnet werden soll unter den Adressen 1000 und 1004 im Speicher. Das kgV selbst soll unter der Adresse 1008 gespeichert werden.

30 Punkte

## Aufgabe 3

Gegeben sei das folgende DLX-Assemblerprogramm:

```
start:  ADD     R2,R0,R0
        ADD     R3,R1,R0
        ADDI    R4,R0,#32
        ADD     R5,R4,R0
loop:   ANDI    R6,R3, #1
        ADD     R2,R2,R6
        SRLI    R3,R3,#1
        SUBI    R4,R4,#1
        BNEZ    R4, loop
end:    SUB     R2,R5,R2
        HALT
```

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

25 Punkte

## Aufgabe 4

Gegeben sei die Funktion $f = (a \cdot b)\ mod\ 4$, wobei $a, b \in \mathbb{Z}$.
*mod* meint dabei die Modulo-Funktion, die den ganzzahligen Rest der Division des Dividenden durch den Divisor wiedergibt.

Beispiel: $19 \bmod 4 = 3$,
da $19 : 4 = 4$ Rest 3.

Schreiben Sie ein DLX-Assembler-Programm, das $f$ realisiert, ohne dabei auf Multiplikationsbefehle oder Gleitkommaregister zurückzugreifen. Die Parameter $a$ und $b$ seien an den Adressen 1000 und 1004 als 32-Bit Integer im Speicher hinterlegt. Das Ergebnis soll nach Berechnung an der Adresse 1008 in den Speicher geschrieben werden. Achten Sie darauf, nur die für die Berechnung nötigen Bits der Parameter zu benutzen, um das Programm möglichst einfach zu halten.

40 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 ? extend(immediate) : 4) |
| BNEZ | branch if not equal to zero | I | PC += (Rs1 != 0 ? extend(immediate) : 4) |
| J | jump | J | PC += extend(immediate) |
| JAL | jump and link | J | R31 = PC + 4 ; PC += 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)] |
| 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)] = Rd |
| 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.