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 12

Ausgabetermin: Freitag, 18.01.2019
Abgabetermin: Freitag, 01.02.2019, 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

Seien $m, n \in \mathbb{N}$. In dieser Aufgabe soll ein Assemblerprogramm entwickelt werden, 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:

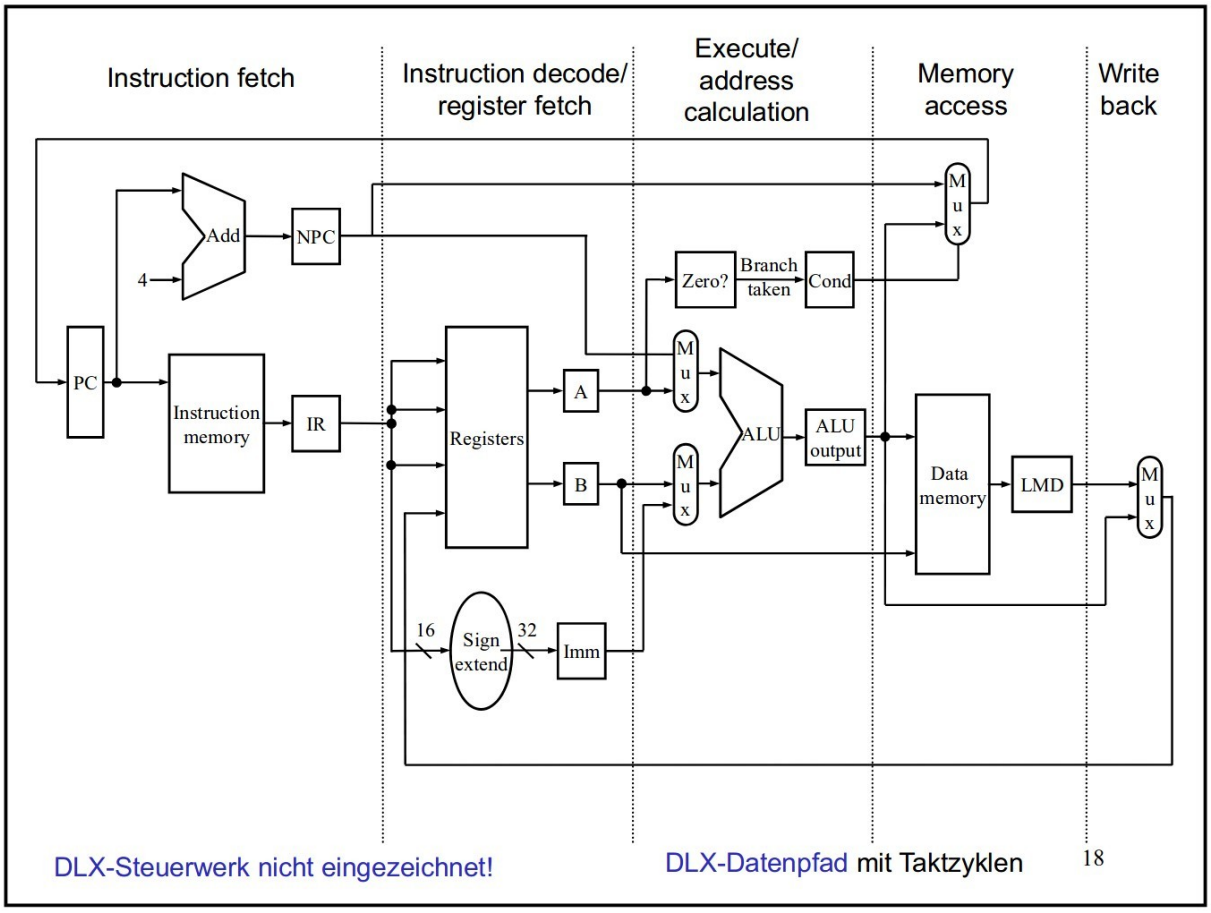
$$kgV(m, n) \leq 2^{31} - 1$$

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

(a) Zeichnen Sie einen Programmablaufplan für das beschriebene Verfahren.

(b) Schreiben Sie das DLX-Assemblerprogramm.

(c) Wie müssen Sie das Programm ändern, wenn Sie es als Unterprogramm verwenden wollen? Die Adresse des ToS sei hierfür in Register R30 gespeichert.



Aufgabe 2

Erklären Sie anhand der gezeigten DLX-Datenpfades detailliert, welche Abläufe in den einzelnen Pipeline-Stufen bei dem Befehl `LW R3, 1000(R1)` ausgeführt werden.

Hausaufgaben

Aufgabe 1

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:

- Falls n ungerade ist, bleibt m gleich, Erg wird um m vergrößert und n um eins reduziert.
- 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 2

In dieser Aufgabe soll die DLX-Architektur anhand einiger Befehle genauer untersucht werden. Beschreiben Sie hierfür detailliert, wie der Prozessor folgende Befehle ausführt:

(a) `ADD R2, R5, R3`

(b) `BNEZ R3, Next`

(c) `SW 1000(R2), R1`

Welche Datenpfade werden vom Steuerwerk ausgewählt? Was berechnet die ALU? Gehen Sie jeweils darauf ein, was in **jeder** Pipelining-Stufe passiert.

10, 10, 10 Punkte

Aufgabe 3

Bubblesort ist ein einfaches Sortierverfahren, welches eine Liste a der Länge n mit $n > 1$ durch wiederholtes Vertauschen benachbarter Elemente in zwei Schleifen sortiert. Das Verfahren ergibt sich durch folgenden Pseudocode:

```
for (i=n; i>1; i--) {                          (b) Bubblesort
  for (j=0; j<i-1; j++) {
    if (a[j] > a[j+1]) {                          (a) Sort
      vertausche a[j] und a[j+1]
    }
  }
}
```

In dieser Aufgabe soll Bubblesort in DLX-Assembler in zwei Teilprogrammen implementiert werden. Hierfür ist eine zu sortierende Liste von Zahlen gegeben, die im Speicher an der Adresse 1200 beginnt. Es wird zusätzlich ein Stack benötigt, der Bottom of Stack ist an der Adresse 1000.

(a) Schreiben Sie ein Assemblerprogramm `Sort`, welches zwei im Speicher aufeinanderfolgende Elemente sortiert. Die Speicheradresse wird als Parameter im Register `R1` übergeben. Die an der Adresse `R1` gespeicherte Zahl x soll also genau dann mit der an der Adresse `R1+4` gespeicherten Zahl y vertauscht werden, wenn $y < x$ gilt. Achten Sie darauf, dass das Programm als Unterprogramm verwendet werden kann. Verwenden Sie also nur Register, deren Werte sie vorher auf dem Stack zwischenspeichern und nachträglich wiederherstellen.

(b) Schreiben Sie ein Assemblerprogramm `Bubblesort`, welches das Unterprogramm `Sort` nutzt. Durch zwei verschachtelte Schleifen soll, wie im Pseudocode dargestellt, eine Liste der Länge n sortiert werden. Diese Liste beginnt an der Speicheradresse 1200, die Länge der Liste soll zu Beginn im Register `R1` stehen.

20, 20 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