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 11

Ausgabetermin: Freitag, 19.1.2018
Abgabetermin: Freitag, 02.02.2018, 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.**

Präsenzaufgaben

Aufgabe 1

Gegeben sei folgendes Programmsegment:

```
ADD   R1, R2, R3
AND   R4, R5, R1
XOR   R6, R3, R4
```

- Erklären Sie, warum der gegebene Programmabschnitt mit Hilfe von Forwarding funktioniert, ohne Staus zu produzieren.
- Ersetzen Sie die Zeile `ADD R1,R2,R3` durch die Zeile `LW R1,1000(R2)`.
  Funktioniert das Programm immer noch ohne Staus? Begründen Sie bitte Ihre Antwort.

Aufgabe 2

Für unterschiedliche Versionen eines Prozessors soll für einen Mix aus Programmen die CPI berechnet werden. Dabei ist die CPI für alle Befehle außer der Branch-Befehle 1, 3.
Bei den Branch-Befehlen braucht man nach dem ersten Takt, in dem die Instruktion gefetcht wird, 3 Takte, um zu entscheiden, ob zur nächsten Instruktion oder zum Sprungziel verzweigt wird. Es dauert nur einen Takt nach dem Fetch, das Sprungziel zu berechnen. Sprungzielberechnung und Entscheidung können parallel bearbeitet werden.
20% der Befehle sind Branches. Bei 30% der Branches wird gesprungen.

- Wie ist die CPI des Prozessors bei Anwendung von Predict-taken?
- Wie ist die CPI des Prozessors bei Anwendung von Predict-not-taken?
- Wie ist die CPI des Prozessors bei Anwendung der freeze-Strategie?

Hausaufgaben

Aufgabe 1

Betrachten Sie das folgende Programmsegment. An der Adresse 1000 beginnt ein Vektor aus 8 Komponenten, wobei jede Komponente eine Gleitkommazahl (Single-Precision) ist. An der Adresse 2000 steht ein weiterer, gleichartiger Vektor.

```
        ADDI    R1, R0, #32     / Initialisierung von R1 für 8 Komponenten zu je 4 Byte
        SUBF    F1, F1, F1      / Setzen von F1 auf 0
loop:   SUBI    R1, R1, #4      / Verringere R1 um 4, R1 zeigt damit auf die nächste Komponente
        LF      F2, 1000(R1)    / Lade die Komponente des ersten Vektors
        LF      F3, 2000(R1)    / Lade die Komponente des zweiten Vektors
        MULTF   F4, F2, F3      / Multipliziere die beiden Zahlen
        ADDF    F1, F4, F1      / Addiere das Ergebnis auf F1
        BNEZ    R1, loop        / Gehe ein weiteres Mal durch die Schleife,
                                / falls noch weitere Komponenten vorhanden sind.
        SF      3000(R0), F1    / Wegschreiben des Ergebnisses
```

(a) Was berechnet das Programm?

(b) Gehen Sie von folgenden Verarbeitungszeiten aus:
MULTF, ADDF und SUBF sind gepipelined, wie in der Vorlesung angegeben mit einer Latency von 6 bei MULTF und 3 bei ADDF und SUBF und jeweils einer Initierungsrate von 1.

Wieviele Stauzyklen entstehen bei der Ausführung des gesamten Programms?
Wieviele Taktzyklen benötigt es dann insgesamt?

(c) Schreiben Sie das Programm um, so dass die Anzahl der Stauzyklen minimal ist.
Wieviele Taktzyklen benötigt Ihr optimales Programm?

Hinweise:

- Da der DLX-Interpreter keine Floating-Point-Operationen unterstützt, kann für diese Aufgabe der DLX-Interpreter nicht genutzt werden.
- Es stehen Ihnen die 32 Float-Register F0, ..., F31 zur Verfügung.

5, 10, 15 Punkte

Aufgabe 2

Ein Automat $A = (X, Y, Z, \delta, \lambda)$, $X = \{0, 1\}$, $Y = \{0, 1\}$, $Z = \{Z_0, Z_1, Z_2\}$ mit Codierung 00 für Z_0, 01 für Z_1 und 10 für Z_2, soll wie folgt in den Registern eines DLX-Prozessors abgebildet sein:
Für jeden Zustand stehen zwei Register zur Verfügung, eines für jede mögliche Eingabe. Dies sind die Register R(2i+1) und R(2i+2) für Z_i mit $0 \leq i < |Z|$. In beiden niederwertigsten Bits dieser Register sind die Folgezustände eingetragen (niederwertigstes Bit Register = niederwertigstes Bit der Zustandskodierung, zweitniederwertigstes Bit Register = höchstwertigstes Bit der Zustandskodierung). Im drittniederwertigsten Bit des Registers steht die Ausgabe für diese Transition. Das Register R(2i+1) macht diese Angaben hierbei für die Eingabe 0, das Register R(2i+2) für die Eingabe 1.
In R10 steht ein vom niederwertigsten Bit an abzuarbeitender Binärstring von Eingabewerten. In R9 steht die Länge dieses Eingabestrings $L < 32$ (d.h. es sind L Bit von R10 als gültige Eingaben zu betrachten). Der jeweils aktuelle Zustand soll in R8 gespeichert werden.

(a) Zeichnen Sie den zu der Registerbelegung R1 = 5, R2 = 12, R3 = 6, R4 = 22, R5 = 0, R6 = 6 gehörenden Automatengraphen.

(b) Ermitteln Sie anhand des Automatengraphen für R8 = 2, R9 = 5, R10 = 33 den Endzustand und die Ausgabefolge.

(c) Schreiben Sie ein DLX-Assemblerprogramm, das für jede beliebige Belegung sowohl der Register, die für die Zustände zur Verfügung stehen als auch von R9 und R10 die Ausgaben von λ in richtiger Reihenfolge als Binärstring in R11 ablegt (letzte Ausgabe auf dem niederwertigsten Bit).

10, 5, 15 Punkte

Aufgabe 3

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

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

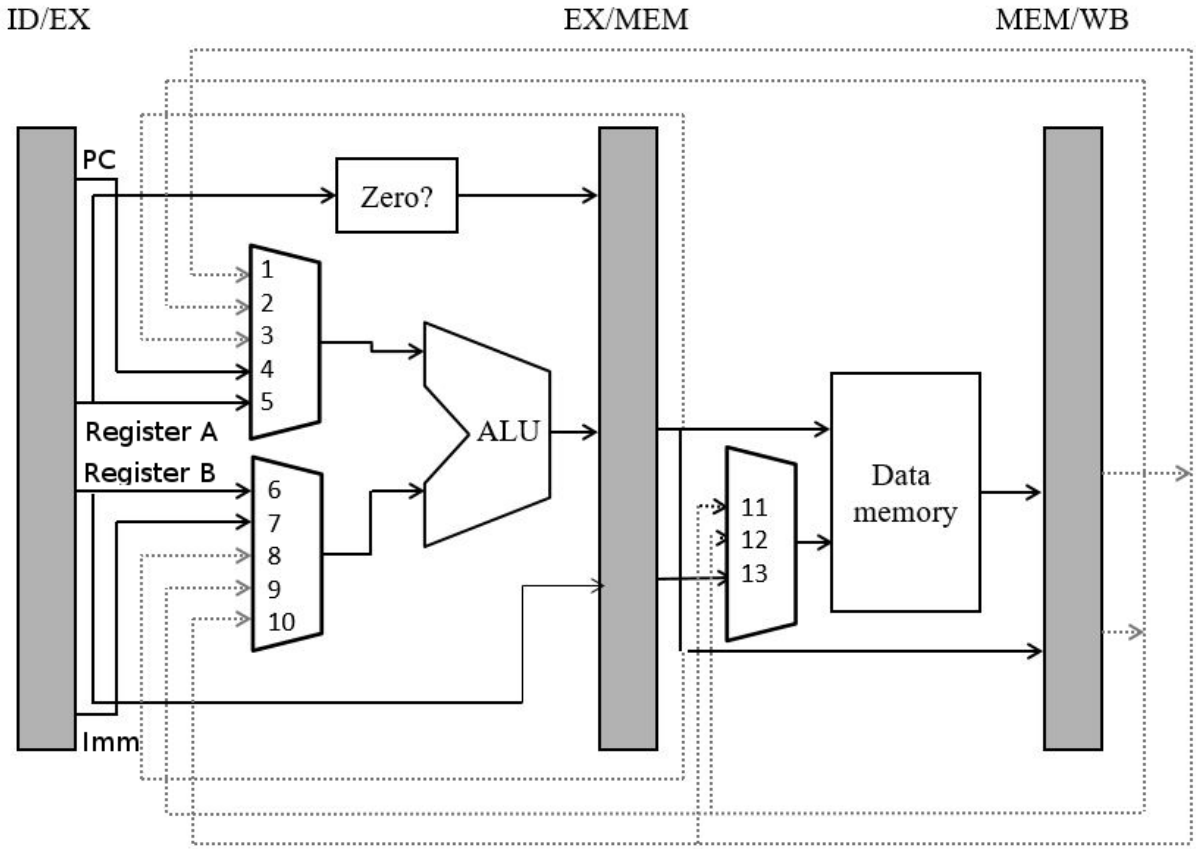
(e) Geben Sie ein CMOS-Gatter an, das die boolesche Funktion *XOR* realisiert.
Bearbeiten Sie diese Aufgabe unter der Voraussetzung, dass die Eingangssignale **nicht** in invertierter Form vorliegen.

(f) Zeichnen Sie einen 1-8-Demultiplexer unter Verwendung eines Decodierers und eines Datenwegschalters.

2, 2, 2, 2, 3, 3 Punkte

Aufgabe 4

Die folgende Abbildung zeigt einen Teil des Datenpfades der DLX-Pipeline mit Forwarding-Logik, bei dem die Eingänge in die Multiplexer mit Zahlen von 1 bis 13 bezeichnet sind.



Geben Sie für jeden der Eingänge eine Befehlsfolge an, bei der dieser Eingang gebraucht wird. Schreiben Sie jeweils dazu, warum dieser Eingang bzw. dieser Forwarding-Pfad hier benötigt wird.

26 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