UE 14	Fachgebiet Architektur eingebetteter Systeme **Rechnerorganisation** Wiederholung	AES Embedded Systems Architecture WS2016/17

Alle Aufgaben, die durch ein ✎ gekennzeichnet sind, sind für die eigenständige Vor- bzw. Nachbereitung der Übung und zur Klausurvorbereitung gedacht. Sie werden in der Regel nicht von den Übungsleitern behandelt, können aber ggf. während des Tutoriums selbständig unter Anleitung des Tutors bearbeitet werden, sofern ausreichend Zeit zur Verfügung steht.

14.1 Rechenleistung

In dieser Aufgabe soll ein Programm betrachtet werden, welches sich aus

- 35% Loads
- 50% ALU-Operationen
- 10% Branches
- 5% Jumps

zusammensetzt. Des Weiteren gilt, dass 35% des Quellcodes parallelisierbar ist.

Das Programm wird auf einem Multicycle-MIPS-Prozessor mit L1-Cache ausgeführt. Der Zugriff auf den L1-Cache dauert 1 Takt und hat eine Trefferquote von 60%. Ein Zugriff auf den Hauptspeicher benötigt 35 Takte.

1. Wie groß ist die durchschnittliche Ausführungszeit eines Load-Befehls?
2. Wie groß ist der CPI des Programms? Bei fehlenden Ergebnissen aus Aufgabenteil (a) kann als Näherungswert $CPI_{Load} = 19$ angenommen werden.
3. Um die Ausführung zu beschleunigen, haben Sie nun die Wahl zwischen zwei Verbesserungsvorschlägen.

   Vorschlag 1: Der Prozessor wird durch ein 4-Kernsystem ersetzt; die Speicherhierarchie bleibt jedoch gleich.

   Vorschlag 2: Der L1-Cache wird so modifiziert, dass er über eine Trefferquote von 90% verfügt.

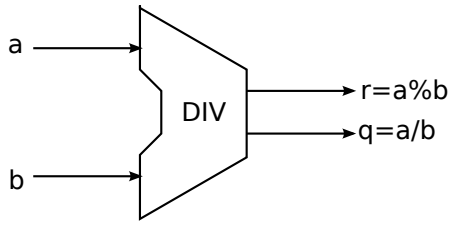
   Welcher Lösungsansatz ist zu bevorzugen, um einen möglichst großen Speed-Up zu erzielen? Begründen Sie Ihre Entscheidung!

14.2 Single Cycle Prozessor

Der Eintaktprozessor aus der Vorlesung, welcher bereits um die Multiplikation erweitert wurde, soll jezt um die `div`- Funktion erweitert werden. Laut MIPS-Green Card verhält sich der Befehl wie folgt:

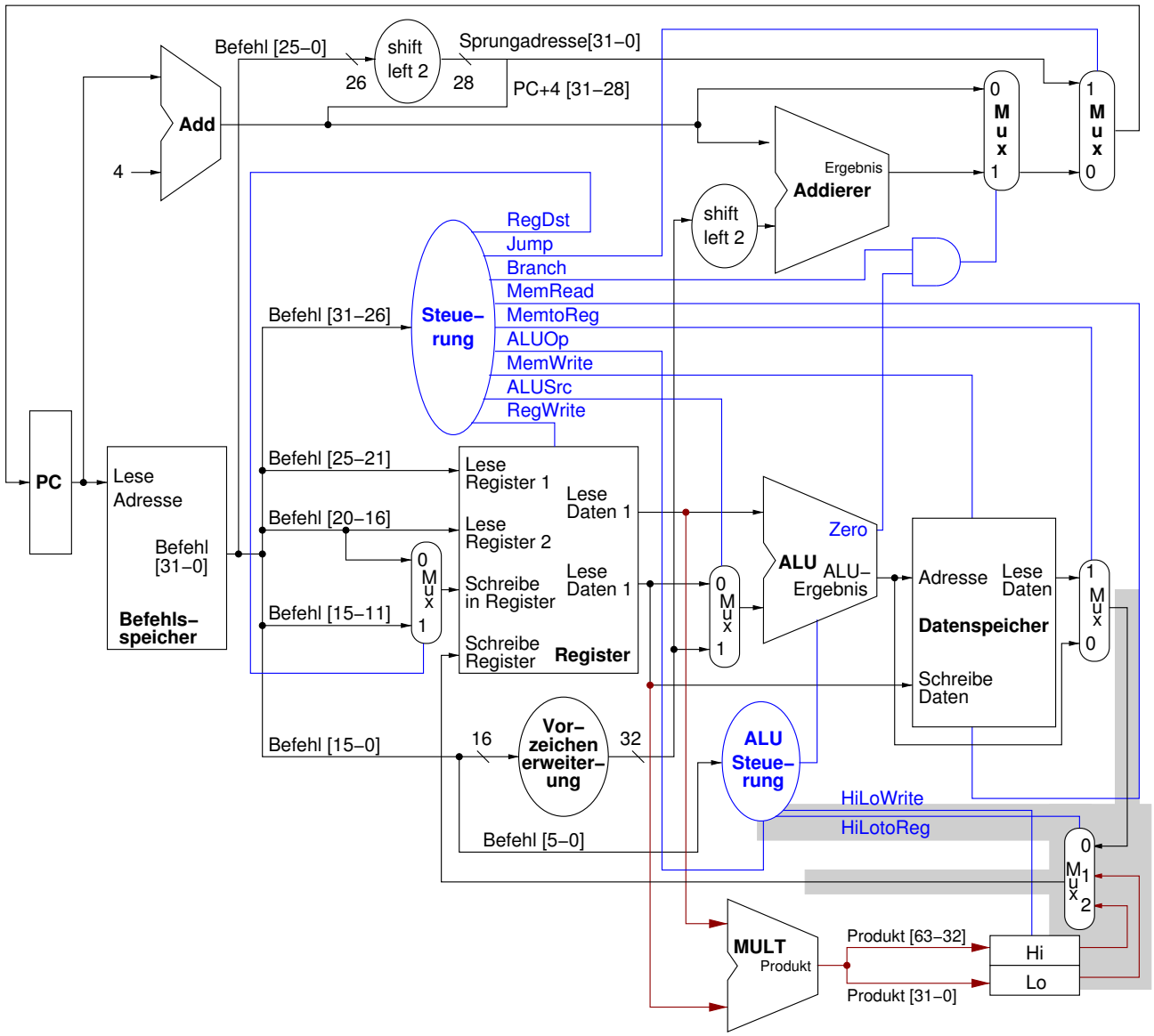
```
0 Hi = R[rs] % R[rt]
1 Lo = R[rs] / R[rt]
```

Für die Erweiterung steht eine spezielle Divisions-Einheit zur Verfügung:



Außerdem soll überpüft werden, ob es sich um eine Division durch Null handelt. Wenn dies der Fall ist, sollen die Ergebnisregister immer auf Null gesetzt werden.

1. Erweitern Sie den Datenpfad auf der nächsten Seite um die Division. Stellen Sie dabei sicher, das bei einer Division durch Null das Ergebnis ebenfalls auf Null gesetzt wird.
2. Geben Sie die Steuersignale für die `div`-Funktion an. Benutzen Sie *don't care*, wenn möglich. Vergessen Sie dabei nicht die Steuersignale `HiLoWrite` und `HiLotoReg`.


Befehl [25–0]
shift left 2
26
28
Sprungadresse[31–0]
PC+4 [31–28]
Add
4
0 M u x 1
1 M u x 0
Ergebnis
Addierer
shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Befehl [31–26]
Steue–rung
PC
Lese Adresse
Befehl [31–0]
Befehls–speicher
Befehl [25–21]
Befehl [20–16]
Befehl [15–11]
Befehl [15–0]
Lese Register 1
Lese Register 2
Schreibe in Register
Schreibe Register
Lese Daten 1
Lese Daten 1
Register
16
Vor–zeichen erweiter–ung
32
Befehl [5–0]
Zero
ALU
ALU–Ergebnis
Adresse
Lese Daten
Datenspeicher
Schreibe Daten
ALU Steue–rung
HiLoWrite
HiLotoReg
MULT
Produkt
Produkt [63–32]
Produkt [31–0]
Hi
Lo

14.3 Caches

Während der Ausführung eines Programms sind folgende Daten im direkt abgebildeten, byte-adressierbaren L1-Cache vorhanden:

Index	V	Tag	Daten
0	0	0x530	0xFFFFFFFFFFFFFFFF
1	1	0x089	0xEEEEEEEEEEEEEEEE
2	0	0x29A	0xDDDDDDDDDDDDDDDD
3	1	0x534	0xCCCCCCCCCCCCCCCC
4	0	0x534	0xBBBBBBBBBBBBBBBB
5	0	0x534	0xAAAAAAAAAAAAAAAA
6	1	0x146	0x9999999999999999
7	0	0x804	0x8888888888888888
8	1	0x733	0x7777777777777777
9	0	0x733	0x6666666666666666
10	1	0x430	0x5555555555555555
11	0	0x660	0x4444444444444444
12	1	0x575	0x3333333333333333
13	1	0x29A	0x2222222222222222
14	1	0x29A	0x1111111111111111
15	0	0xFFF	0x0000000000000000

Gehen Sie bei allen Berechnungen davon aus, dass die dargestellten Zahlen über keine weiteren führenden Nullen verfügen.

1. Wie viele Bits werden zur Implementierung des Caches benötigt?
2. Die Adresslänge beträgt in diesem Fall **nicht** 32 bit. Wie viele Bits werden stattdessen für die Adressierung benötigt? Wie verteilen sich diese Bits auf Index, Tag und Offset?
3. Es erfolgen als nächstes vier Schreibzugriffe auf den Cache:
   a) Adresse `0x29A21`, Daten `0xAAAAAAAAAAAAAAAA`
   b) Adresse `0x29A1B`, Daten `0xBBBBBBBBBBBBBBBB`
   c) Adresse `0x29A19`, Daten `0xCCCCCCCCCCCCCCCC`
   d) Adresse `0x29A10`, Daten `0xDDDDDDDDDDDDDDDD`

   Ein solcher Schreibzugriff überschreibt immer eine komplette Cachezeile. Wie sieht der Inhalt des Caches nach diesen Zugriffen aus? Tragen Sie **ausschließlich** diejenigen Sätze in die Vorlage auf der nächsten Seite ein, die sich verändert haben.
   Tipp: Betrachten Sie die Cache-Adressen als Binärzahl und verwenden Sie ihre Ergebnisse aus Aufgabenteil b.), um die benötigten Werte zu bestimmen.

Ausgangszustand

Index	V	Tag	Daten	Index
0	0	0x530	0xFFFFFFFFFFFFFFFF	0
1	1	0x089	0xEEEEEEEEEEEEEEEE	1
2	0	0x29A	0xDDDDDDDDDDDDDDDD	2
3	1	0x534	0xCCCCCCCCCCCCCCCC	3
4	0	0x534	0xBBBBBBBBBBBBBBBB	4
5	0	0x534	0xAAAAAAAAAAAAAAAA	5
6	1	0x146	0x9999999999999999	6
7	0	0x804	0x8888888888888888	7
8	1	0x733	0x7777777777777777	8
9	0	0x733	0x6666666666666666	9
10	1	0x430	0x5555555555555555	10
11	0	0x660	0x4444444444444444	11
12	1	0x575	0x3333333333333333	12
13	1	0x29A	0x2222222222222222	13
14	1	0x29A	0x1111111111111111	14
15	0	0xFFF	0x0000000000000000	15

Vorlage für Aufgabenteil (c)

Index	V	Tag	Daten