FernUniversität in Hagen

Computersysteme II - Kurs 1609

Einsendeaufgaben zur Kurseinheit 1

Aufgabe 1.1 (5 P)

Entscheiden Sie, welche der folgenden Aussagen über die Architektur von Prozessoren korrekt sind.

A Unter der Prozessorarchitektur versteht man den vollständigen inneren Aufbau des Prozessors.

B Zum Programmiermodell eines Prozessors gehören u.a. der Befehlssatz, das Befehlsformat und die Adressierungsarten.

C Die Eigenschaften der Mikroarchitektur werden von der Prozessorarchitektur selbst nicht erfasst und sind deshalb unbedeutend für die Systemsoftware sowie für den Systemprogrammierer.

D Bei einem N-Bit-Prozessor besitzen die allgemeinen internen Register sowie die effektiven Adressen im allgemeinen die Breite von N Bit.

E Bei einem wortadressierbaren Adressraum ist es insbesondere möglich auf jedes Byte des Adressraumes einzeln zuzugreifen.

Aufgabe 1.2 (8 P)

Darstellung von Gleitkommazahlen in einfacher Genauigkeit nach dem IEEE-Standard.
Eine normalisierte Gleitkommazahl f in einfacher Genauigkeit im IEEE-Format ist definiert als

$$f = (-1)^s \cdot 2^{e-b} \cdot 1, m.$$

Dabei ist:

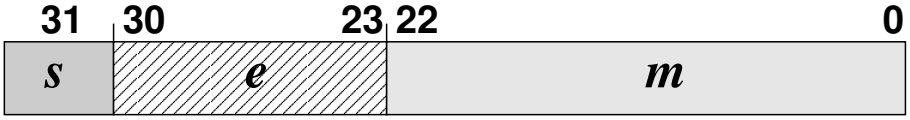
s das Vorzeichenbit (0 für positive, 1 für negative Zahlen),

e mit 8 Bit der um die Verschiebung b = 127 verschobene Exponent und

m mit 23 Bit der gebrochene Teil der Mantisse 1,m (zur Basis 2).

Hier ist zu beachten, dass diese Formel nur für $0 < e < 255$ gilt. Die möglichen Werte $e = 0$ und $e = 255$ sind Sonderfälle. Zum Beispiel wird die Zahl $+0$ durch $s = 0$, $e = 0$ und $m = 0$, und die Zahl -0 durch $s = 1$, $e = 0$ und $m = 0$ dargestellt. Diese Sonderfälle wollen wir hier aber nicht betrachten.

Die Gleitkommazahlen in einfacher Genauigkeit werden rechnerintern in Speicherworten von 32 Bit Länge wie folgt dargestellt:



IEEE–Format von Gleitkommazahlen in einfacher Genauigkeit

Mit den folgenden Teilfragen wollen wir herausfinden wie die gebrochene Dezimalzahl $-2345,375$ als normalisierte Gleitkommazahl in einfacher Genauigkeit im IEEE-Format in einem Speicherwort von 32 Bit Länge rechnerintern dargestellt wird.

a) Wandeln Sie die dezimale Festkommazahl $2345,375$ in eine binäre Festkommazahl um. (2P)

b) Bestimmen Sie für die gebrochene Dezimalzahl $-2345,375$ den binären Wert von s. (1P)

c) Bestimmen Sie für die gebrochene Dezimalzahl $-2345,375$ den binären Wert von e. (2P)

d) Bestimmen Sie für die gebrochene Dezimalzahl $-2345,375$ den binären Wert von m. (2P)

e) Bestimmen Sie für die gebrochene Dezimalzahl $-2345,375$ das zugehörige 32 Bit lange Speicherwort im IEEE-Format. (2P)

f) Bestimmen Sie auch für die gebrochene Dezimalzahl $4690,75$ das zugehörige 32 Bit lange Speicherwort im IEEE-Format. (2P)
(Hinweis: $4690,75 = 2 \cdot 2345,375$)

Aufgabe 1.3 (4 P)

Im Folgenden soll ein bitfolgenorientierter Multimedia-Additionsbefehl auf die zwei Register R1 und R2 angewendet werden. Die in den Registern enthaltenen Zahlen befinden sich im Zweierkomplement.

R1	A1	25	76	FA

R2	10	38	CD	B8

Das Ergebnis der Addition mit Vorzeichen stehe im Register R3. Bestimmen Sie den Inhalt von R3 und geben Sie diesen von links nach rechts in das folgende Feld ein.

Aufgabe 1.4 (6 P)

Entscheiden Sie, welche der folgenden Aussagen zur Adressierung korrekt sind.

A Als effektive Adresse bezeichnet man diejenige Adresse, an der sich das zur Ausführung eines Befehls benötigte Datum tatsächlich befindet.

B Die 0-Adress- und die Akkumulatormaschine verwenden ausschließlich implizite Adressierung.

C Bei der direkten Adressierung besteht der Operand aus der Adresse des zu verarbeitenden Datums.

D Bei der unmittelbaren Adressierung besteht der Operand aus dem zu verarbeitenden Datum.

E Bei der indirekten Adressierung besteht der Operand aus der Adresse des zu verarbeitenden Datums.

F Bei einer indizierten Adressierung sind die Operanden durch zwei Registerinhalte gegeben, wobei sich die effektive Adresse als Modulo-2-Summe dieser beiden Registerinhalte ergibt.

Aufgabe 1.5 (10 P)

Geben Sie eine möglichst kurze Befehlsfolge zur Durchführung der Operation

$$(\,<\text{A}>\,*\,(\,<\text{B}>+2\,)\,) \longrightarrow \text{R0}$$

auf einer LOAD/STORE-Architektur mit

a) Zweiadressbefehlen und

b) Einadressbefehlen

an, wobei A und B Hauptspeicherzellen sind und R0 ein Register. Als weiteres Register kann R1 benutzt werden.

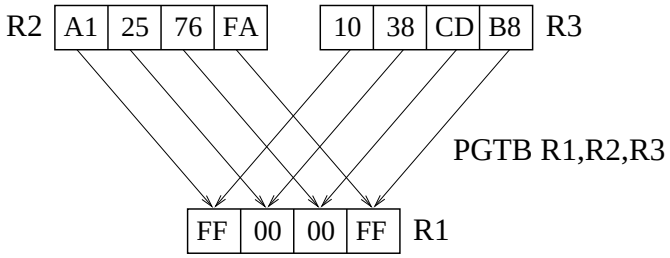
Führen Sie die Operation durch, indem Sie von der innersten Klammer ausgehen und von den beiden Registern R0 und R1 bevorzugt R0 verwenden.
Benutzen Sie die Befehlsbezeichner LOAD, ADD, MUL und STORE.
Adressierungsarten sind *direkt* und *unmittelbar*.
Geben Sie außer zwischen Befehl und Operand keine weiteren Leerzeichen ein, mehrere Operanden werden durch Komma getrennt.

Aufgabe 1.6 (10 P)

MMX (Multi Media eXtension)
In dieser Aufgabe verwenden wir einige vereinfachte MMX-Befehle, deren Wirkungsweise wie folgt erklärt wird:

PAND	Ri,Rj,Rk	$Ri = Rj \wedge Rk$
POR	Ri,Rj,Rk	$Ri = Rj \vee Rk$
PNOT	Ri,Rj	$Ri = \neg Rj$
PGTB	Ri,Rj,Rk	$Ri \leftarrow Rj > Rk$

PGTB bedeutet *Parallel Greater Than Byte*. Wenn ein 8-Bit-Paket in Register R2 (vorzeichenlos) größer ist als das zugehörige Byte in Register R3, wird in Register R1 $FF abgelegt, ansonsten $00.



a) Erstellen Sie ein kurzes Programm mit den oben angegebenen MMX-Befehlen zur Berechnung der byteweisen Maxima von Register R2 und R3. Das Ergebnis der Berechnung, das byteweise Maxima soll in Register R1 abgelegt werden. Zur Ausführung des Programmes stehen die 32-bit-Register R1 bis R3 zur Verfügung. (5P)
Verwenden Sie *ausschließlich* die oben genannten Befehle und geben Sie die einzelnen Befehle chronologisch und syntaktisch exakt in die folgenden Felder ein. In unbenutzte Fehler geben Sie "-" ein.

b) Prüfen Sie Ihr Programm mit den Registerinhalten R2=$A12576FA und R3=$1038CDB8.
Schreiben Sie zu jedem Befehl Ihres Programms den Inhalt des im entsprechenden Befehl veränderten Registers in die entsprechenden Felder. Gehen Sie davon aus, das die Register R2 und R3 bereits die genannten Werte enthalten. (5P)

Aufgabe 1.7 (8 P)

Entscheiden Sie, welche der folgenden Aussagen zum Thema CISC/RISC-Prinzipien korrekt sind.

A RISC-Architekturen zeichnen sich durch wenige, unbedingt notwendige Befehle und Befehlsformate aus.

B Wichtige Voraussetzung für eine effiziente Implementierung einer RISC-Architektur ist die Verfügbarkeit preiswerter Speichertechnologie.

C CISC-Architekturen zeichnen sich durch umfangreiche Befehlssätze und mächtige Maschinenbefehle mit vielen Befehlsformaten, aber wenigen Adressierungsarten aus.

D Sowohl bei CISC- als auch bei RISC-Architekturen erfolgt die Parameterübergabe bei Unterprogrammaufrufen in der Regel über einen LIFO-organisierten Stackspeicher, der innerhalb des Hauptspeichers angelegt wird.

E Bei RISC-Architekturen kann gegenüber CISC-Architekturen die Chip-Fläche auf dem Prozessor effizienter genutzt werden.

F CISC-Architekturen werden wegen der Vielzahl an Adressierungsarten und der daraus resultierenden Möglichkeiten Operanden und Daten zu laden und zu speichern auch LOAD/STORE-Architekturen genannt.

G Wegen des kleineren Befehlssatzes bei RISC-Architekturen wird dort grundsätzlich mehr Speicherplatz für die Unterbringung von Programmen benötigt.

H Die Registerfenster-Technologie (window register organization) dient bei RISC-Architekturen einer einfachen und effizienten Parameterverteilung.

Aufgabe 1.8 (10 P)

Entscheiden Sie, welche der folgenden Aussagen zum Thema Prozessorarchitekturen und Pipelining korrekt sind.

A Das von-Neumann-Prinzip beschreibt eine prinzipielle Rechnerstruktur, bei der der Prozessor die Ablaufsteuerung und die Ausführung der Befehle übernimmt und die Schnittstelle des Prozessors nach außen durch eine spezielle Aus-/Eingabeeinheit gebildet wird.

B Das Operationsprinzip der von-Neumann-Architektur besagt, dass die Befehle in einer sequentiellen Reihenfolge ausgeführt werden müssen, ermöglicht aber auch ein Pipelining in zwei Befehlsphasen.

C Der von-Neumann-Flaschenhals entsteht daraus, dass Befehle und Daten im gleichen Speicherbereich untergebracht sind und daher den gleichen Weg zwischen Prozessor und Speicher nehmen müssen.

D Der Hauptspeicher des von-Neumann-Rechners besteht aus Speicherzellen fester Länge und dient hauptsächlich zur Aufnahme der Programme, Daten werden dagegen wegen des häufigeren und schnelleren Zugriffs hauptsächlich im Cache-Speicher abgelegt.

E Beim Harvard-Rechner werden Programme und Daten im Hauptspeicher voneinander getrennt abgelegt um Befehle und Daten voneinander zu unterscheiden und damit die Programme besser lesen zu können.

F Moderne Rechner enthalten neben dem Hauptspeicher- und Festplatten-Cache zusätzlich einen speziellen Befehlscache auf dem Prozessorchip, durch den das Harvard-Prinzip zumindest ansatzweise realisiert ist.

G Bei der überlappenden Befehlsbearbeitung in den zwei Phasen "Bereitstellung und Decodierung" und "Ausführung" entstehen sowohl in einer von-Neumann- als auch in einer Harvard-Architektur keine Ressourcenkonflikte.

H Pipelining beschreibt die Bearbeitung eines Auftrags (hier einer Maschinenoperation) in Form von überlappendend ausgeführten unabhängigen bzw. nebenläufigen Teilaufträgen in Form eines Fließbandes.

I Pipelining ermöglicht die Beschleunigung der Bearbeitung durch Parallelisierung von nebenläufigen Teilaufträgen.

J Zur Beschleunigung der Pipeline werden die Zwischenergebnisse der einzelnen Pipelinestufen in speziellen Registern innerhalb der Pipeline und nicht in den eigentlichen CPU-Registern gespeichert.

Aufgabe 1.9 (10 P)

Entscheiden Sie, welche der folgenden Aussagen zum Thema Befehls-Pipeling korrekt sind.

A Um eine Befehlspipeline zu implementieren, müssen die einzelnen Phasen der Befehlsausführung so beschaffen sein, dass diese exakt gleich lang sind.

B Enthält der auf der Pipeline auszuführende Befehlssatz Befehle, die nicht LOAD-/STORE-konform sind, muss die Bereitstellung der Operanden in mehreren Takten erfolgen.

C Arithmetisch-logische Operationen werden aus Effizienzgründen und der einfachen Implementierung wegen grundsätzlich in einer einzigen Pipeline-Phase untergebracht.

D Ein Pipeline-Konflikt bezeichnet die Unterbrechung (Störung) des taktsynchronen Durchlaufs der Befehle in der Pipeline infolge von Abhängigkeiten bei Daten oder Ressourcen oder im Steuerfluss.

E Eine Befehls-Pipeline kann durch die Ausführung in vier Stufen Holphase, Decodierphase, Ausführungsphase und Rückschreibphase implementiert werden.

F Bei einem Datenkonflikt wird das in einem Register abgelegte Ergebnis einer früheren Operation von einer Anweisung überschrieben, bevor es weiter verwendet werden konnte.

G Steuerflusskonflikte werden hauptsächlich durch Sprungbefehle verursacht, bei denen das Sprungziel erst unmittelbar vor der Ausführung des Sprungs ermittelt werden kann.

H Betrachtet man zwei Befehle, wobei der eine Befehl einen Operanden aus einem Register holt, in das der andere Befehl ein Ergebnis ablegt, so bezeichnen wir diese Situation als Gegenabhängigkeit.

I Gegenüber Gegen- und Ausgabeabhängigkeiten, die in der Regel durch Umbenennungen von Variablen oder Registern entfernt werden können, verursachen echte Datenabhängigkeiten, bei denen auf Ergebnisse gewartet werden muss, Probleme in der Pipeline.

J Ein Schreibe-nach-Lese-Konflikt ist meist durch einen größeren Abstand der kollidierenden Befehle im Programm vermeidbar, ein Lese-nach-Schreibe-Konflikt ist dagegen nur durch eine Restrukturierung der Pipeline lösbar.

Aufgabe 1.10 (27 P)

(Befehls-)Pipelining
Ausgangspunkt des Befehls-Pipelining ist die Aufspaltung eines Befehls in Teilschritte. Das Befehls-Pipelining setzt voraus, dass für die zugrundeliegende Befehlsarchitektur eine Folge von Teilschritten

gefunden wird, die für *alle* Befehle gleich ist. Für die folgende Aufgabe nehmen wir an, dass alle Befehle (inklusive der zugehörigen Adressierungsarten) in den folgenden fünf Teilschritten, welche Ihnen aus der DLX-Architektur bekannt sind, bearbeitet werden.

1. Instruction Fetch (IF):	Befehl holen (Opcode und Registeradressen können in einem Maschinenwort untergebracht werden).
2. Instruction Decode (ID):	Befehl dekodieren und gleichzeitig die Quelloperanden aus dem Registerblock lesen.
3. Execute (EX):	Führe eine arithmetische bzw. logische Operation mit den Operanden aus.
4. Memory Access (MEM):	Daten holen (*LOAD*) oder Speichern (*STORE*).
5. Write Back (WB):	Ergebnis in einem prozessorinternen Register speichern.

Der zeitliche Ablauf der Teilschritte bei konfliktfreiem Befehls-Pipelining sieht dabei wie folgt aus:

IF	ID	EX	MEM	WB		
	IF	ID	EX	MEM	WB	
		IF	ID	EX	MEM	WB

Bei dieser Definition der Pipeline gehen wir, ohne weitere Annahmen zu machen, davon aus, dass ein aus dem ALU-Ergebnisregister in das Universalregister oder Architekturregister zurückgeschriebener Wert erst am Ende der WB-Stufe stabil und damit weiterverwendbar anliegt.

Pipelinekonflikte können durch zusätzliche Hardware oder Software (Einfügen von NOP, *no-operation-Befehle*) behoben werden. In den folgenden Teilaufgaben soll nur die Software-Lösung betrachtet werden.

Wir untersuchen einen Pipelinekonflikt im folgenden Programmfragment:

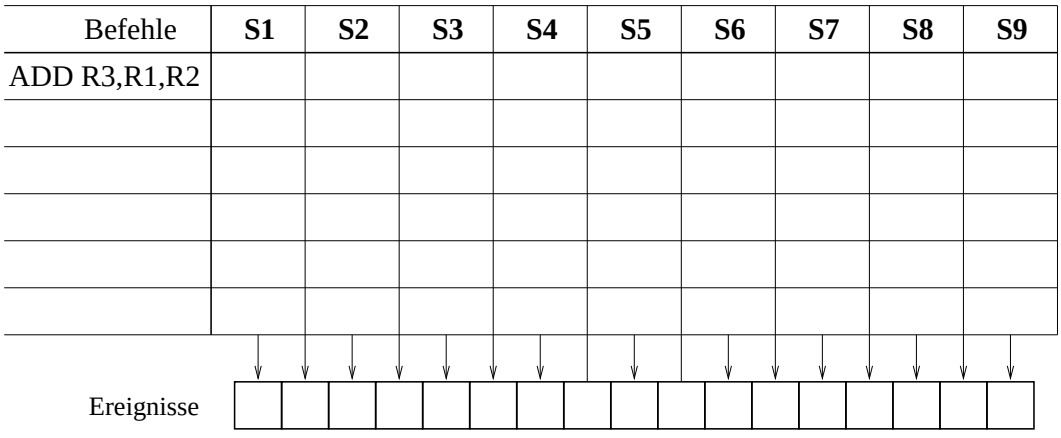
ADD R3,R1,R2	$R3 = R1 + R2$
SUB R4,R1,R3	$R4 = R1 - R3$

a) Welche Art von Pipelinekonflikt liegt hier genau vor? (3P)
Antwort: Bei dem vorliegenden Pipelinekonflikt handelt es sich um einen _____.
Dieser _____ wird durch eine _____ verursacht, die in der Verwendung von _____ liegt.

b) Ausführung des Programmfragments auf der oben definierten Pipeline (12P)

b1) Skizzieren Sie den optimalen Ablauf des Programms in der Pipeline, indem Sie das folgende Pipeline-Ablaufzeitdiagramm unter Beachtung der Kausalität ausfüllen. Dazu soll in die einzelnen Felder der darin ausgeführte jeweilige Pipeline-Teilschritt **Si** eingetragen werden.

Verwenden Sie dazu statt der ausgeschriebenen Worte {**Instruction Fetch, Instruction Decode, Execute, Memory Access, Write Back**} die Kürzel {**IF,ID,EX,MEM,WB**}.



b2) Markieren Sie im Pipeline-Ablaufdiagramm die Ereignisse **E1**, **E2**, **E3** und **E4** indem Sie das entsprechende Kürzel in die an den entsprechenden Stellen vorgesehenen Felder unterhalb des Diagrammes eintragen.

E1 Wert von R3 durch den ADD-Befehl errechnet,

E2 Wert von R3 liegt *vor* dem Registerblock an,

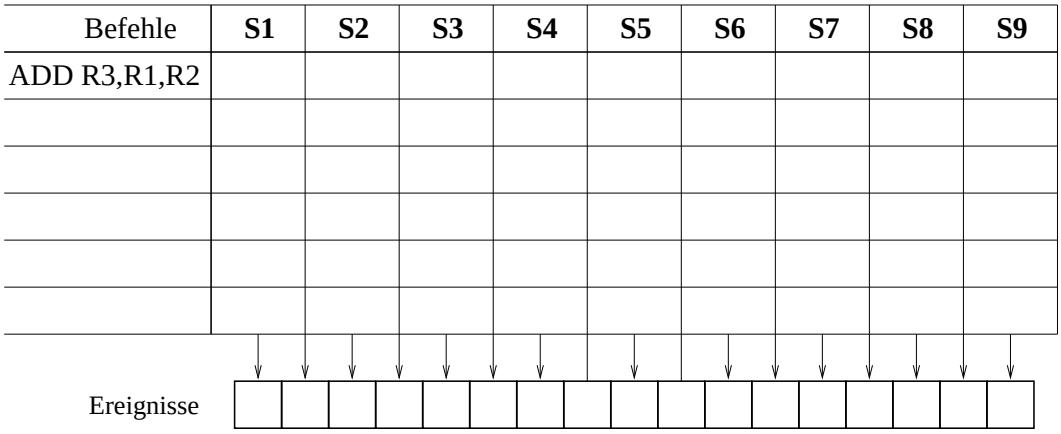
E3 Wert von R3 wurde im Registerblock gespeichert und

E4 R3 liegt als Eingangsoperand für den SUB-Befehl an der ALU vor.

c) Ausführung des Programmfragments auf einer beschleunigten DLX-Pipeline (12P)

In diesem Aufgabenteil kann davon ausgegangen werden, das ein aus dem ALU-Ergebnisregister in das Universalregister zurückgeschriebener Wert bereits nach der ersten Hälfte der WB-Stufe stabil und damit weiterverwendbar anliegt und dass die Operandenwerte erst in der zweiten Hälfte der ID-Stufe von den Universalregistern in die ALU-Eingaberegister übertragen werden. In der WB-Stufe geschriebene Registerwerte können also bereits in demselben Takt wieder in der ID-Stufe gelesen werden.

c1) Skizzieren Sie den optimalen Ablauf des Programms in der Pipeline, indem Sie das folgende Pipeline-Ablaufzeitdiagramm unter Beachtung der Kausalität ausfüllen.



c2) Markieren Sie im Pipeline-Ablaufdiagramm die Ereignisse **E1**, **E2**, **E3** und **E4**.

Aufgabe 1.11 (16 P)

(Befehls-)Pipelining

Für die folgende Aufgabe sei die Befehls-Pipeline mit den folgenden fünf Teilschritten der DLX-Architektur gegeben:

1. Instruction Fetch (IF):	Befehl holen (Opcode und Registeradressen können in einem Maschinenwort untergebracht werden).
2. Instruction Decode (ID):	Befehl dekodieren und gleichzeitig die Quelloperanden aus dem Registerblock lesen.
3. Execute (EX):	Führe eine arithmetische bzw. logische Operation mit den Operanden aus.
4. Memory Access (MEM):	Daten holen (*LOAD*) oder Speichern (*STORE*).
5. Write Back (WB):	Ergebnis in einem prozessorinternen Register speichern.

wobei der Ablauf bei konfliktfreiem Pipelining wie folgt aussieht:

IF	ID	EX	MEM	WB		
	IF	ID	EX	MEM	WB	
		IF	ID	EX	MEM	WB

Wir betrachten einen Pipeline-Konflikt, wie er im folgenden Programmfragment vorliegt:

Nr.	**Befehl**		**Bedeutung**
1000	BEQZ	20	Falls Zero-Bit=1 ist, springe um 20 Wörter relativ zum aktuellen Befehlszählerstand
...	...	...	...
1024	ADD	R1,R2,R3	$R1 = R2 + R3$

a) Welche Art von Pipelinekonflikt liegt hier genau vor? (4P)
Antwort: Bei dem vorliegenden Pipelinekonflikt handelt es sich um einen _____. Dieser _____ wird durch den _____ verursacht, wobei der nächste Befehl erst geladen werden kann, wenn der _____ einen gültigen Stand erreicht hat.

b) Ausführung des Programmfragments auf der oben definierten Pipeline (10P)

Skizzieren Sie den optimalen Ablauf des Programms in der Pipeline, indem Sie das folgende Pipeline-Ablaufzeitdiagramm unter Beachtung der Kausalität ausfüllen. Dazu soll in die einzelnen Felder der darin ausgeführte jeweilige Pipeline-Teilschritt **Si** eingetragen werden.

Verwenden Sie dazu statt der ausgeschriebenen Worte {**Instruction Fetch, Instruction Decode, Execute, Memory Access, Write Back**} die Kürzel {**IF,ID,EX,MEM,WB**}.

Befehle	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
BEQZ 20										

c) Markieren Sie im obigen Pipeline-Ablaufzeitdiagramm die folgenden Ereignisse **E1** und **E2**

E1 Neuer Befehlszählerstand bestimmt und

E2 Wert von R1 wurde berechnet.

indem Sie das entsprechende Kürzel in die an den entsprechenden Stellen vorgesehenen Felder unterhalb des Diagrammes eintragen. (2P)

Aufgabe 1.12 (12 P)

(Befehls-)Pipelining
Für die folgende Aufgabe sei wieder die Befehls-Pipeline mit den folgenden fünf Teilschritten der DLX-Architektur gegeben:

1. **Instruction Fetch (IF):** Befehl holen (Opcode und Registeradressen können in einem Maschinenwort untergebracht werden).
2. **Instruction Decode (ID):** Befehl dekodieren und gleichzeitig die Quelloperanden aus dem Registerblock lesen.
3. **Execute (EX):** Führe eine arithmetische bzw. logische Operation mit den Operanden aus.
4. **Memory Access (MEM):** Daten holen (*LOAD*) oder Speichern (*STORE*).
5. **Write Back (WB):** Ergebnis in einem prozessorinternen Register speichern.

Dabei gehen wir davon aus, dass ein aus dem ALU-Ergebnisregister in das Universalregister oder Architekturregister zurückgeschriebener Wert erst am Ende der WB-Stufe stabil und damit weiterverwendbar anliegt.

Das folgenden Programmfragment sei für die Ausführung auf einem DLX-Prozessor vorgesehen:

J	20	
SUB	R6,R2,R1	$R6 = R2 - R1$
AND	R5,R2,R1	$R5 = R2 \wedge R1$
ADD	R3,R1,R2	$R3 = R1 + R2$
SUB	R4,R1,R3	$R4 = R1 - R3$
ADD	R1,R4,R5	$R1 = R4 + R5$
XOR	R6,R7,R8	$R6 = R7 \oplus R8$

a) Bestimmen Sie für dieses Programmfragment eine konfliktfreie Sequenz zur Ausführung auf einem DLX-Prozessor durch Einfügen einer minimalen notwendigen Anzahl von NOP-Befehlen. Tragen Sie die einzelnen Befehle der modifizierten Programmsequenz chronologisch und syntaktisch exakt in die folgenden Felder ein. In unbenutzte Fehler geben Sie "-" ein. (6P)

b) Durch Umstellen von Befehlen kann ein optimierender Compiler die zur Konfliktbehebung erforderlichen NOP-Befehle durch andere (unabhängige) Befehle ersetzen und so die Effizienz des Pipelining steigern.
Eliminieren Sie die NOP-Befehle in Teilaufgabe a) durch geeignetes Umstellen von Befehlen. (6P)
Tragen Sie die einzelnen Befehle der Programmsequenz chronologisch und syntaktisch exakt in die folgenden Felder ein. In unbenutzte Fehler geben Sie "-" ein.

Aufgabe 1.13 (22 P)

Datenkonflikte bzw. Datenabhängigkeiten können wie im Kurstext durch einen Abhängigkeitsgraphen dargestellt werden. Eine andere Möglichkeit ist die Beschreibung dieser Abhängigkeiten in Form einer Relation, die wie folgt definiert werden kann:

*Ein Ereignis F heisst **abhängig** von einem anderen Ereignis E, sofern F nicht stattfinden kann, ohne dass sich vorher E ereignet hat. Wir schreiben in diesem Fall $E \lessdot F$.*

$$\begin{aligned} E \lessdot F &= \text{„Ereignis } F \text{ ist abhängig von Ereignis } E\text{“} \\ &= \text{„Ereignis } E \text{ findet stets vor Ereignis } F \text{ statt“} \end{aligned}$$

Da die Relation $\lessdot$ die zeitliche Reihenfolge bestimmter Ereignisse festlegt, spricht man auch von einer *Präzedenzrelation*. Präzedenzrelationen sind strenge partielle Ordnungen, d.h. sie sind:

transitiv:	$(A \lessdot B) \wedge (B \lessdot C) \Rightarrow A \lessdot C$
irreflexiv („schleifenfrei“):	es gibt kein Ereignis A mit $A \lessdot A$
und **asymmetrisch**:	$A \lessdot B \Rightarrow \neg(B \lessdot A)$

Irreflexivität schließt aus, dass es ein Ereignis gibt, welches bereits stattgefunden haben muss, bevor es stattfinden kann.

Zwei verschiedene Ereignisse A und B, für die in einem Prozess keine Reihenfolge festgelegt ist, für die also weder $A \lessdot B$ noch $B \lessdot A$ gilt, heißen *nebenläufig* (concurrent), und wir schreiben in diesem Fall:

$$A \; co \; B$$

Nebenläufige Ereignisse eines Prozesses können zeitgleich, überlappend oder in einer beliebigen Reihenfolge stattfinden, da kein ursächlicher Zusammenhang zwischen ihnen besteht. Prozesse ohne Nebenläufigkeit heissen *sequentielle Prozesse*. Die Präzedenzrelation bildet in diesem Fall eine strenge *totale* Ordnung, das heisst für jedes Paar E, F von Ereignissen gilt genau eine der folgenden drei Beziehungen

$$E \lessdot F \;\text{ oder }\; E = F \;\text{ oder }\; F \lessdot E$$

Gegeben sei eine Befehls-Pipeline, die arithmetische Zwei-Adressbefehle (`ADD`, `SUB` und `MUL`) in den folgenden vier Teilschritten bearbeitet, wobei kein Forwarding möglich ist:

TS1: Holen des Befehlscodes, Dekodierung und Bestimmung der Adresse des nächsten Befehls.

TS2: Holen von Operanden aus dem Hauptspeicher und Transport zu den Eingängen der ALU.

TS3: Durchführung der Operation und Schreiben des Ergebnises an den Ausgang der ALU.

TS4: Transport des Ergebnisses in den Hauptspeicher oder in ein Register.

Führen Sie die Rechenoperation

$$((<\text{A}> + <\text{B}>) * (<\text{C}> + <\text{D}>) * (<\text{A}> + <\text{C}> + <\text{E}>)) \;\rightarrow\; \text{A}$$

auf fünf verschiedenen Registerzellen A,B,C,D und E durch. Dabei sollen Zwei-Adressbefehle zur Verfügung stehen. Natürlich dürfen Sie nicht mehr benötigte Registerinhalte überschreiben.

a) Geben Sie die zur Berechnung der angegebenen Operation notwendigen Aktionen bzw. Befehle A_i an. Es stehen dazu 4 Additions- und 2 Multiplikationsbefehle, also insgesamt 6 Aktionen bzw. Befehle A_i, $i \in \{1, \ldots, 6\}$ zur Verfügung.
Um die Lösungsmöglichkeiten einzuschränken, sollen die anzugebenden Operationen die Additionen als auch die Multiplikationen in der Reihenfolge des Auftretens im Ausdruck von links nach rechts auch bzgl. der Register realisieren. (6P)

b) Geben Sie entsprechend der obigen Definition einer Präzedenz die zur Ausführung der Rechenoperation $((<A>+<B>)*(<C>+<D>)*(<A>+<C>+<E>)) \rightarrow A$ erforderlichen Präzedenzen für die Ausführung der unter a) festgelegten 6 Aktionen bzw. Befehle A_i, $i \in \{1 \ldots 6\}$ an. (8P)
Hinweis: Es treten 7 nicht transitive und eine transitive Präzedenz $A_i \lessdot A_j$ auf.

Geben Sie die dazugehörige transitivitätsfreie Präzedenzrelation an. (2P)

c) Zeichnen Sie sich den Präzedenzgraphen zur Prädezenzrelation aus b) auf. Stellen Sie daneben auch den transitivitätsfreien Präzedenzgraphen dar.
Bestimmen Sie die nebenläufigen Befehlspaare der obigen Rechenoperation und geben Sie diese in Form einer Relation an. (3P)

d) Geben Sie aufgrund der Nebenläufigkeitsrelation in c) an, welche Aktionen A_i in welchem Schritt parallel ausgeführt ausgeführt werden können.
Hinweis: Es reichen 3 Schritte aus. (3P)

Aufgabe 1.14 (29 P)

Gegeben sei eine Befehls-Pipeline, die arithmetische Zwei-Adressbefehle (`ADD`, `SUB` und `MUL`) in den folgenden vier Teilschritten bearbeitet, wobei kein Forwarding möglich ist:

TS1: Holen des Befehlscodes, Dekodierung und Bestimmung der Adresse des nächsten Befehls.

TS2: Holen von Operanden aus dem Hauptspeicher und Transport zu den Eingängen der ALU.

TS3: Durchführung der Operation und Schreiben des Ergebnises an den Ausgang der ALU.

TS4: Transport des Ergebnisses in den Hauptspeicher oder in ein Register.

Auf dieser Befehls-Pipeline soll die Rechenoperation

$$((<A>+<B>)*(<C>+<D>)*(<A>+<C>+<E>)) \rightarrow A$$

auf den fünf Registerzellen A,B,C,D und E mit den folgenden 6 Aktionen bzw. Befehle A_i, $i \in \{1, \ldots, 6\}$ möglichst effektiv durchgeführt werden.

$A_1 =$ ADD B, A
$A_2 =$ ADD D, C
$A_3 =$ ADD E, C
$A_4 =$ ADD A, E
$A_5 =$ MUL B, D
$A_6 =$ MUL A, B

a) Geben Sie entsprechend der Definition einer Präzedenz die zur Ausführung der obigen Rechenoperation anhand der 6 Aktionen bzw. Befehle A_i, $i \in \{1 \ldots 6\}$ erforderlichen Präzedenzen für die Ausführung an. (5P)
Hinweis: Es treten 6 nicht transitive und 3 transitive Präzedenzen $A_i \lessdot A_j$ auf.

Geben Sie die dazugehörige transitivitätsfreie Präzedenzrelation an. (2P)

b) Zeichnen Sie sich den Präzedenzgraphen zur Prädezenzrelation aus a) auf. Stellen Sie daneben auch den transitivitätsfreien Präzedenzgraphen dar.
Bestimmen Sie die nebenläufigen Befehlspaare der obigen Rechenoperation und geben Sie diese in Form einer Relation an. (3P)

c) Geben Sie aufgrund der Nebenläufigkeitsrelation in b) an, welche Aktionen A_i in welchem Schritt parallel ausgeführt ausgeführt werden können. (3P)

d) Geben Sie nun entsprechend der obigen Definition einer Präzedenz die zur Ausführung der obigen Rechenoperation erforderlichen Präzedenzen für die Ausführung des k-ten Teilschrittes der Befehle A_i auf der gegebenen vierschrittigen Pipeline an. Dabei bezeichne $E_i(k)$ für $i \in \{1 \ldots 6\}$ und $k \in \{1 \ldots 4\}$ die Ausführung des k-ten Teilschrittes des Befehls A_i.
Da die verschiedenen Teilschritte eines jeden Befehls der Reihe nach ausgeführt werden müssen, gilt natürlich $E_i(k) \lessdot E_i(l)$ für $i \in \{1 \ldots 6\}$ und $k, l \in \{1, 2, 3, 4\}$ mit $k < l$. Diese Präzedenzen sollen nicht mehr angegeben werden. (6P)
Geben Sie zusätzlich an, um welche Art von Konflikt es sich genau handelt.

e) Überlegen Sie sich aufgrund der Präzedenzrelation aus a) und der Nebenläufigkeitsrelation aus b) eine mögliche vollständige Sequentialisierung Q der Befehle A_i. Gemeint ist nur die Angabe einer Befehlsreihenfolge, also nicht die Sequentialisierung der Ausführung *aller* $6 \cdot 4 = 24$ Teilschritte. (2P)
Wie viele mögliche verschiedene Sequentialisierungen gibt es insgesamt? (2P)

f) Geben Sie eine möglichst optimale Ausführung der Befehle auf der gegebenen vierschrittigen Pipeline an. Fügen Sie dabei eine minimale Anzahl von NOP-Befehlen in die Befehlsfolge Q ein. (3P)

g) Vergleichen Sie die parallele Ausführung der Operation mit der Ausführung auf der vierschrittigen Pipeline. (3P)

A Bei einer optimierten Ausführung auf der gegebenen vierstufigen Pipeline werden insgesamt 12 Teilschritte benötigt.

B Bei einer optimalen Parallelisierung werden insgesamt 12 Teilschritte benötigt.

C Für die Parallelisierung ist der Aufwand an Resourcen ca. 3-mal so groß wie der vierstufigen Pipeline.

Aufgabe 1.15 (11 P)

BTC (Branch Target Cache)
Gegeben sei folgendes kurzes Programm:

Nr.	Befehl		Bedeutung
1	LOAD	R1,#10	R1 mit 10 laden
2	ADD	R1,R1,#1	R1 inkrementieren
3	STORE	[R2],R1	R1 indirekt in MEM[R2] (externes RAM) speichern
4	JUMP	1	Bei 1 fortfahren

Für das Befehlsformat gelte, dass sowohl OpCode wie auch Operand 16 bit breit sind und dass der betrachtete Prozessor einen 16 bit breiten Datenbus besitzt. Das gesamte Programm liege im externen Speicher des Prozessors.

a) Wie viele Schreib/Lese-Zugriffe sind bei einem Durchlauf auf das externe RAM zu beobachten, wenn der Prozessor keinerlei Cache besitzt? (3P)
Geben Sie in das erste Feld die Zahl der Befehlszugriffe, in das zweite Feld die Zahl der Datenzugriffe und in das dritte Feld die Gesamtzahl an Zugriffen ein.

b) Nun werde der Prozessor um einen BTC (*branch target cache*) erweitert. Wie viel Zugriffe auf das externe RAM sind nach dem ersten Durchlauf zu messen, wenn pro BTC-Eintrag Platz für eine Maschinen-Instruktion ist? (3P)
Geben Sie in das erste Feld die Zahl der Adressen ein, die im BTC pro Sprung gespeichert werden, in das zweite Feld die Zahl der Befehlszugriffe und in das dritte Feld die Zahl der Datenzugriffe.

c) Entscheiden Sie, welche der folgenden Aussagen zum Sprungziel-Caching korrekt sind. (5P)

A Moderne Prozessoren besitzen einen Instruktions- oder Befehlscache, der das Holen eines Befehls erheblich beschleunigt, da der Befehl sonst direkt aus dem HSP gelesen werden muss.

B Ein zusätzlicher BTC zum Instruktionscache wird die Abarbeitung im Prozessor, insbesondere bei Sprüngen, erheblich beschleunigen.

C Ist ein Instruktionscache vorhanden, dann kann anstelle eines BTC ohne Einschränkung auch ein BTAC verwendet werden.

D Für einen BTAC wird weniger Speicherplatz benötigt als für einen BTC.

E Da einfache Mikrocontroller i.a. ohne Instruktionscache gefertigt werden, ist ein BTC eine einfache Möglichkeit hier den Befehlsdurchsatz zu erhöhen.