Aufgabe 1: Cache-Adressierung

Ihr Cachingsystem soll 16 GiB an Speicher auf Datenwortebene adressieren können. Die Datenwortlänge beträgt 4 Byte, der Cache umfasst 512 KiB und besteht aus 128 Byte großen Blöcken.

a) Betrachten Sie für den Cache die Varianten *Direct Mapped Cache*, *8-Way Set Associative Cache* und *Fully Associative Cache*. Berechnen Sie für jede dieser Varianten die Längen von *Tag*, *Index*, *Offset* und die Adresslänge.

b) Berechnen Sie zu den Adressen 0xBADCAB1E und 0x23454321 die Werte für *Tag*, *Index* und *Offset* zu jedem der drei Caches aus Teilaufgabe a). Geben Sie die Werte jeweils binär und hexadezimal an, wobei führende Nullen weggelassen werden können.

c) Angenommen, zur Realisierung der Ersetzungsstrategie (z.B. LRU, LFU, etc.) werden beim *8-Way Set Associative Cache* 4 Bit pro Block und beim *Fully Associative Cache* 7 Bit pro Block benötigt.
Bei schreibendem Zugriff soll darüber hinaus bei einem *Hit* das Verfahren *Copy-Back* angewendet werden, anderenfalls *Fetch-on-Write*.

Wie viel Speicher muß für die Verwaltung der drei Caches (alle gespeicherten Informationen außer den Nutzdaten) jeweils berücksichtigt werden?

Aufgabe 2: Multilevel Caching

Gehen Sie von einem Prozessor mit folgenden Eckdaten aus:

- Prozessortakt: 8 GHz
- Zugriffsdauer auf L1-Cache: 4 Taktzyklen
- Zugriffsdauer auf L2-Cache: 40 Taktzyklen
- Zugriffsdauer auf Hauptspeicher: 100 Taktzyklen

Hinweis: In der Zugriffsdauer auf den L2-Cache ist die Zugriffsdauer für den *Miss* auf den L1-Cache bereits inkludiert. Selbiges gilt für die Zugriffsdauer auf den Hauptspeicher.

a) Angenommen, die CPU führt 10.000 Adressanfragen aus. Die *Hit-Rate* (h) für den L1-Cache betrage 90% und für den L2-Cache 80%. Wie viele *Misses* treten beim L1-Cache bzw. beim L2-Cache auf?

b) Geben Sie die Zugriffsdauer auf den L1-Cache (t_{L1}), auf den L2-Cache (t_{L2}) und auf den Hauptspeicher (t_{main}) in Nanosekunden an.

c) Um welchen Faktor verbessert sich die effektive Speicherzugriffszeit durch Verwendung des L2-Caches? Berechnen Sie dazu $t_{\mathrm{eff\_L1}}$ für das Szenario ohne L2-Cache und $t_{\mathrm{eff\_L2}}$ für das Szenario mit L2-Cache. Der Beschleunigungsfaktor ergibt sich dann als Quotient $t_{\mathrm{eff\_L1}}/t_{\mathrm{eff\_L2}}$.

Hinweis: Die effektive Speicherzugriffszeit wird bei Verwendung eines Caches grundsätzlich nach folgender Formel berechnet:

$$t_{\mathrm{eff}} = h \cdot t_{\mathrm{cache}} + (1 - h) \cdot t_{\mathrm{main}}$$

d) Nehmen Sie an, dass die Kapazität des L2 Caches beträchtlich größer ist als die des L1 Caches. Gibt es, unter der weiteren Annahme, dass es sich um einen korrekten Hardwareentwurf handelt, eine Erklärung dafür, dass die Hitrate des L1 Caches höher ist als die des L2 Caches? Wenn ja, geben Sie eine mögliche Erklärung an. Wenn nein, erklären Sie, wieso das nicht möglich ist.

Aufgabe 3: Virtuelle Adressierung

Sie arbeiten auf einem Betriebssystem mit 32 Bit virtuellem Adressraum und 32 KiB großen Pages. Die Adressierung erfolgt byteweise. Nehmen Sie an, dass nur 1/8 (= 12.5%) des adressierbaren Speichers im System tatsächlich physisch verbaut ist.

a) Wie lang ist der Offset?

b) Aus wie vielen Bits besteht die Page-Nummer?

c) Wie lang sind die physikalischen Adressen?

d) Wie viel Speicher ist im System physisch verbaut? Geben Sie den Wert in MiB an!

e) Wie groß ist die Page-Table maximal, wenn Page-Nummern explizit gespeichert werden und jeder Eintrag zusätzlich 8 *Permission Bits* enthält? Geben Sie die Größe der Page-Table in KiB an!

Hinweis: Maximal bedeutet hier, dass zu jeder Page-Nummer ein Eintrag (ggf. mit lauter Nullen initialisiert) in der Page-Table angelegt wird.

f) Wie viele Frames muss sich das Betriebssystem reservieren, um die Page-Table aus Teilaufgabe e) im physikalischen Speicher zu halten?

Aufgabe 4: Basic RISC-V Pipeline: Hazards und Branch Target Buffer

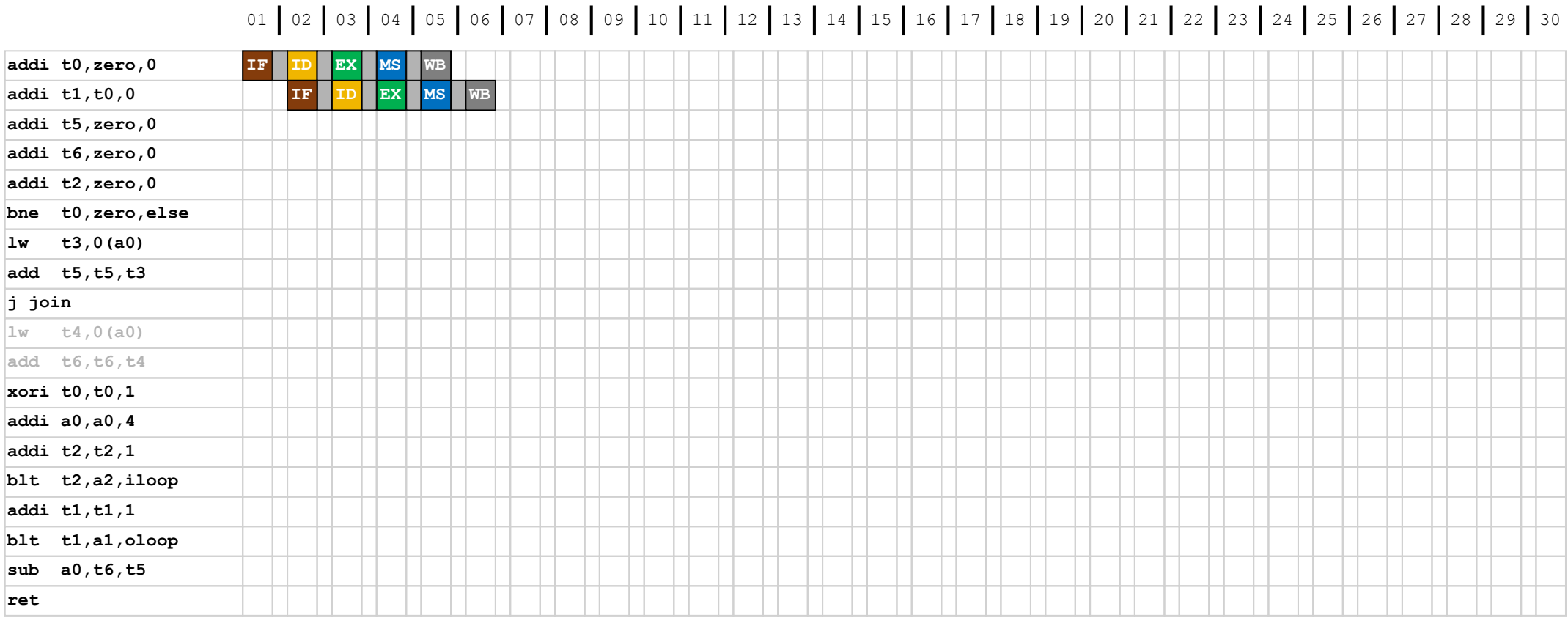
Gegeben ist der folgende RISC-V-Programmcode. Nehmen Sie an, dass dieser auf der aus der Vorlesung bekannten 5-stufigen Pipeline (in-order, skalar) ausgeführt wird, Instruction sowie Data Memory innerhalb des gleichen Zyklus Daten liefern (keine Verzögerung), Forwarding zunächst nicht implementiert ist, Sprünge in der dritten Pipelinestufe verarbeitet werden und zunächst `always not taken` als statische Sprungvorhersage genutzt wird.

```
1  0x00: function1: addi t0,zero,0
2  0x04:            addi t1,t0,0
3  0x08:            addi t5,zero,0
4  0x0C:            addi t6,zero,0
5  0x10: oloop:     addi t2,zero,0
6  0x14: iloop:     bne  t0,zero,else
7  0x18:            lw   t3,0(a0)
8  0x1C:            add  t5,t5,t3
9  0x20:            j join
10 0x24: else:      lw   t4,0(a0)
11 0x28:            add  t6,t6,t4
12 0x2C: join:      xori t0,t0,1
13 0x30:            addi a0,a0,4
14 0x34:            addi t2,t2,1
15 0x38:            blt  t2,a2,iloop
16 0x3C:            addi t1,t1,1
17 0x40:            blt  t1,a1,oloop
18 0x44:            sub  a0,t6,t5
19 0x48:            ret
```

a) Ermitteln Sie alle Codestellen, bei denen Datenhazards (DH) und Kontrollhazards (KH) bei Ausführung des gegebenen Codes auftreten könnten. Nutzen Sie dafür folgende Notation: $DH_{reg}(ZQI-ZZI)$ bzw. $KH(ZQI)$, wobei *reg*=Registernummer, *ZQI*=Zeile der Quellinstruktion und *ZZI*=Zeile der Zielinstruktion.

Beispiel: $DH_{t0}(1-2)$, d.h. ein Datenhazard, der das Register `t0` betrifft und durch das Schreiben von `t0` in Zeile 1 und das nachfolgende Lesen von `t0` in Zeile 2 ausgelöst wird.

b) Erstellen Sie ein Pipelinediagramm für den gegebenen Code, das die Ausführung auf der oben beschriebenen Pipeline zeigt, wobei diese jedoch um Forwarding erweitert wurde. Sonstige Kontroll- und Datenhazards sollen automatisch in der Pipeline durch Einfügen von Stalls behandelt werden. Kennzeichnen Sie in Ihrem Pipelinediagramm die auftretenden Stalls. Nehmen Sie an, dass $a1 = 1$ und $a2 = 1$.


01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
addi t0,zero,0
addi t1,t0,0
addi t5,zero,0
addi t6,zero,0
addi t2,zero,0
bne t0,zero,else
lw t3,0(a0)
add t5,t5,t3
j join
lw t4,0(a0)
add t6,t6,t4
xori t0,t0,1
addi a0,a0,4
addi t2,t2,1
blt t2,a2,iloop
addi t1,t1,1
blt t1,a1,oloop
sub a0,t6,t5
ret
IF ID EX MS WB
IF ID EX MS WB

c) Die gegebene Pipeline mit Forwarding wird nun um eine Einheit zur dynamischen Sprungvorhersage erweitert, die sich in der ersten Pipelinestufe befindet. In dieser Einheit kommt ein Branch Target Buffer (BTB) zum Einsatz, der für den Fall, dass ein Sprung zu `taken` ausgewertet wird, die Adresse des Sprungs selbst (BIA) sowie die dazugehörige Sprungzieladresse (BTA) speichert. Gehen Sie davon aus, dass der BTB zu Beginn leer ist und über ausreichend Kapazität verfügt, um alle in diesem Codeausschnitt ausgeführten Sprünge speichern zu können (nehmen Sie an, dass die erste Instruktion an Adresse `0x0` liegt). Gehen Sie weiterhin davon aus, dass die Richtung des Sprunges (taken/not taken) durch einen Prädiktor stets korrekt vorhergesagt wird, bei einem BTB Miss jedoch die Vorhersage auf `not taken` geändert wird. Zeigen Sie den Verlauf der Änderungen im BTB, d.h. Zugriffe und mögliche Änderungen der Einträge im BTB, die zugreifende Instruktion (nehmen Sie an, dass nur `branch` und `j/jal` Instruktionen auf den BTB zugreifen und letztere hier im Hinblick auf den BTB wie eine branch Instruktion behandelt werden sollen) und ob der Zugriff ein Hit oder Miss war.

Nehmen Sie hier an, dass $a1 = 1$ und $a2 = 2$. Ein entsprechendes Protokoll der ausgeführten Instruktionen finden Sie im Folgenden:

```
0x00: addi t0,zero,0
0x04: addi t1,t0,0
0x08: addi t5,zero,0
0x0C: addi t6,zero,0
0x10: addi t2,zero,0
0x14: bne  t0,zero,else
0x18: lw   t3,0(a0)
0x1C: add  t5,t5,t3
0x20: j join
0x2C: xori t0,t0,1
0x30: addi a0,a0,4
0x34: addi t2,t2,1
0x38: blt  t2,a2,iloop
0x14: bne  t0,zero,else
0x24: lw   t4,0(a0)
0x28: add  t6,t6,t4
0x2C: xori t0,t0,1
0x30: addi a0,a0,4
0x34: addi t2,t2,1
0x38: blt  t2,a2,iloop
0x3C: addi t1,t1,1
0x40: blt  t1,a1,oloop
0x44: sub  a0,t6,t5
0x48: ret
```

Füllen Sie folgende Tabellen aus:

Initial state:

Valid	BIA	BTA
0		
0		
0		
0		
0		
0		

Valid	BIA	BTA

Accessed BIA:

Hit (H)/Miss (M):

Valid	BIA	BTA

Accessed BIA:

Hit (H)/Miss (M):

Valid	BIA	BTA

Accessed BIA:

Hit (H)/Miss (M):

Valid	BIA	BTA

Accessed BIA:

Hit (H)/Miss (M):

Valid	BIA	BTA

Accessed BIA:

Hit (H)/Miss (M):

Valid	BIA	BTA

Accessed BIA:

Hit (H)/Miss (M):

Valid	BIA	BTA

Accessed BIA:

Hit (H)/Miss (M):

Aufgabe 5: Basic RISC-V Pipeline: Branch Prediction

Gegeben ist der folgende RISC-V-Programmcode (in `a0` wird die Basisadresse eines Arrays übergeben; `a1` enthält den Wert 2 und `a2` enthält den Wert 4). Nehmen Sie an, dass dieser Code auf der aus der Vorlesung bekannten 5-stufigen Pipeline (in-order, skalar) ausgeführt wird, Instruction sowie Data Memory innerhalb des gleichen Zyklus Daten liefern (keine Verzögerung) und Forwarding implementiert ist. Diese Pipeline wurde um eine Einheit zur Sprungvorhersage erweitert, die sich in der ersten Pipelinestufe befindet. In dieser Einheit können unterschiedliche Prädiktoren für *bedingte Sprünge* (conditional branches) verwendet werden, die hinsichtlich ihres Verhaltens analysiert werden sollen (nehmen Sie an, dass unbedingte Sprünge die Sprungvorhersage nicht beeinflussen).

```
1  0x00: function1: addi t0,zero,0
2  0x04:            addi t1,t0,0
3  0x08:            addi t5,zero,0
4  0x0C:            addi t6,zero,0
5  0x10: oloop:     addi t2,zero,0
6  0x14: iloop:     bne  t0,zero,else
7  0x18:            lw   t3,0(a0)
8  0x1C:            add  t5,t5,t3
9  0x20:            j join
10 0x24: else:      lw   t4,0(a0)
11 0x28:            add  t6,t6,t4
12 0x2C: join:      xori t0,t0,1
13 0x30:            addi a0,a0,4
14 0x34:            addi t2,t2,1
15 0x38:            blt  t2,a2,iloop
16 0x3C:            addi t1,t1,1
17 0x40:            blt  t1,a1,oloop
18 0x44:            sub  a0,t6,t5
19 0x48:            ret
```

Geben Sie in den folgenden Tabellen für jeden relevanten Sprung innerhalb der `function1`-Funktion an

(i) um welchen Sprung es sich handelt (Zeilennummer),

(ii) welche Vorhersage der Prädiktor trifft (predict taken (PT) oder predict not taken (PNT) für einen 1-bit Prädiktor bzw. predict strongly taken (PST), predict weakly taken (PWT), predict weakly not taken (PWNT) oder predict strongly not taken (PSNT) für einen 2-bit Prädiktor,

(iii) wie sich der Sprung tatsächlich verhält (taken (T) oder not taken (NT)),

(iv) ob die Vorhersage des Prädiktors korrekt ist oder nicht (Y/N).

a) Es wird ein globaler 1-bit Prädiktor verwendet, d.h. der Prädiktor wird für alle relevanten Sprünge gemeinsam genutzt. Nehmen Sie an, dass die initiale Vorhersage `predict not taken (PNT)` lautet.

	01	02	03	04	05	06	07	08	09
Sprung (Zeile)	Z06								
Vorhersage	PNT								
Sprungrichtung									
Korrekt?									

	10	11	12	13	14	15	16	17	18
Sprung (Zeile)									
Vorhersage									
Sprungrichtung									
Korrekt?									

b) Es wird ein lokaler 2-bit Prädiktor verwendet, d.h. der Prädiktor wird für alle relevanten Sprünge getrennt genutzt. Nehmen Sie an, dass die initiale Vorhersage jeweils `predict weakly not taken (PWNT)` lautet.

	01	02	03	04	05	06	07	08	09
Sprung (Zeile)	Z06								
Vorhersage (Z06)	PWNT								
Vorhersage (Z___)	-								
Vorhersage (Z___)	-								
Sprungrichtung									
Korrekt?									

	10	11	12	13	14	15	16	17	18
Sprung (Zeile)									
Vorhersage (Z06)									
Vorhersage (Z___)									
Vorhersage (Z___)									
Sprungrichtung									
Korrekt?									

c) Welche Fehlvorhersagequote (`misprediction rate`) kann für die beiden oben genannten Konfigurationen der Sprungvorhersage in der `function1`-Funktion beobachtet werden?

Aufgabe 6: Advanced Pipelines: Dependencies

Gegeben ist der folgende RISC-V-Programmcode:

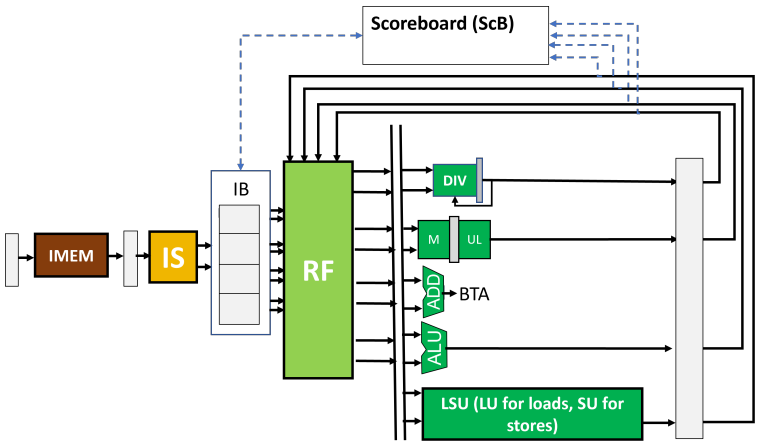
```
1  xori x10,x0,7
2  lw   x22,0(x6)
3  lw   x11,4(x6)
4  sw   x11,0(x7)
5  sw   x22,4(x7)
6  mul  x12,x10,x11
7  andi x13,x10,15
8  div  x14,x8,x13
9  mul  x15,x11,x11
10 div  x16,x8,x9
11 mul  x13,x10,x16
12 addi x17,x14,8
13 mul  x18,x12,x12
14 mul  x19,x13,x9
15 ori  x19,x19,31
16 sw   x17,8(x7)
```

Ermitteln Sie alle echten Datenabhängigkeiten (`true dependencies` (TD)), Gegenabhängigkeiten (`anti dependencies` (AD)) und Ausgabeabhängigkeiten (`output dependencies` (OD)) im oben genannten Code. Nutzen Sie dafür folgende Notation: $\{T|A|O\}D_{reg}(ZQI - ZZI)$, wobei ZQI=Zeile der Quellinstruktion, ZZI=Zeile der Zielinstruktion und reg=Registernummer.

RAW/True Dependencies	**WAR/Anti Dependencies**	**WAW/Output Dependencies**
Beispiel: $TD_{x19}(14-15)$		

Aufgabe 7: Advanced Pipelines: Scoreboard

Gegeben ist die folgende Pipeline:



Diese hat die folgenden Eigenschaften:

- Issue Buffer-Größe (`IB size`) von 4
- 4 Ports um Instruktionen aus dem Issue Buffer zu emittieren (`instruction issue`)
- 4 Ports um Daten zurückzuschreiben (`writeback`)

Es sind folgende funktionale Einheiten vorhanden:

- ALU: 1 Zyklus Latenz; 1 Zyklus Initialisierungsintervall
- ADD: 1 Zyklus Latenz; 1 Zyklus Initialisierungsintervall
- MUL: 2 Zyklen Latenz; 1 Zyklus Initialisierungsintervall (pipelined)
- DIV: 4 Zyklen Latenz; 4 Zyklen Initialisierungsintervall (serial)
- LSU:
  - LU: 2 Zyklen Latenz; 1 Zyklus Initialisierungsintervall (non-blocking)
  - SU: 1 Zyklus Latenz, danach Store Buffer; 1 Zyklus Initialisierungsintervall

Gegeben ist zudem folgender RISC-V-Programmcode:

```
1  xori x10,x0,7
2  lw   x22,0(x6)
3  lw   x11,4(x6)
4  sw   x11,0(x7)
5  sw   x22,4(x7)
6  mul  x12,x10,x11
7  andi x13,x10,15
8  div  x14,x8,x13
9  mul  x15,x11,x11
10 div  x16,x8,x9
11 mul  x13,x10,x16
12 addi x17,x14,8
13 mul  x18,x12,x12
14 mul  x19,x13,x9
15 ori  x19,x19,31
16 sw   x17,8(x7)
```

a) Zeigen Sie für die genannte Pipeline die Ausführung des Codes. Erstellen Sie dazu ein Pipelinediagramm und zeigen Sie den Zustand des Scoreboards und den Status der Funktionseinheiten in jedem Zyklus. Nehmen Sie an, dass kein Register Renaming verwendet wird.

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
xori x10,x0,7	IF	IS	ALU	WB																															
lw x22,0(x6)																																			
lw x11,4(x6)																																			
sw x11,0(x7)																																			
sw x22,4(x7)																																			
mul x12,x10,x11																																			
andi x13,x10,15																																			
div x14,x8,x13																																			
mul x15,x11,x11																																			
div x16,x8,x9																																			
mul x13,x10,x16																																			
addi x17,x14,8																																			
mul x18,x12,x12																																			
mul x19,x13,x9																																			
ori x19,x19,31																																			
sw x17,8(x7)																																			

Cycle: 02

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

Cycle: ____

Instr.	rd	rs1	rs2	Imm	RO	Complete

DIV	MUL	ALU	ADD	SU	LU

b) Bestimmen Sie den erreichten CPI-Wert des Codes bei der Ausführung auf dieser Pipeline.