Aufgabe 1: Advanced Pipelines: Register Renaming und simple superskalare Pipeline

Gegeben ist folgender RISC-V Code:

```
1 lw   x10,0(x6)
2 mul  x11,x10,x10
3 sw   x11,4(x6)
4 slli x13,x12,2
5 addi x12,x13,4
6 div  x12,x12,x7
```

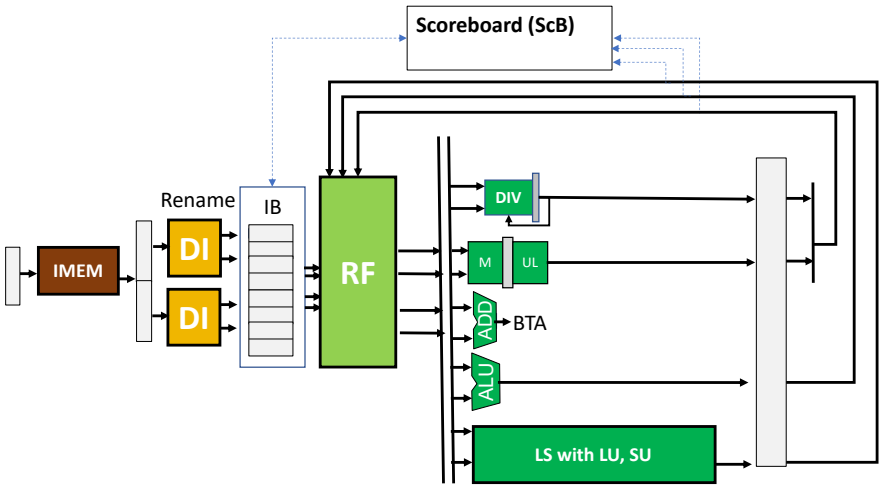
Die Ausführung auf der Pipeline aus Übung 4, Aufgabe 7 ergibt folgendes Pipelinediagramm:

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
`lw x10,0(x6)`	IF	IS	LU	LU	WB										
`mul x11,x10,x10`		IF	IS	IB	MUL	MUL	WB								
`sw x11,4(x6)`			IF	IS	IB	IB	SU	SB							
`slli x13,x12,2`				IF	IS	ALU	WB								
`addi x12,x13,4`					IF	stall	IS	ALU	WB						
`div x12,x12,x7`						stall	IF	stall	IS	DIV	DIV	DIV	DIV	WB	

a) Um Abhängigkeiten zu reduzieren, soll Register Renaming eingesetzt werden. Führen Sie die Änderungen direkt im Code durch, wobei Sie annehmen können, dass die Register `x28-x30` bisher nicht verwendet wurden und damit zur Verfügung stehen. Zeigen Sie das resultierende Pipelinediagramm für die Pipeline aus Übung 4, Aufgabe 7 und ermitteln Sie den erreichten IPC-Wert. Welche Probleme wurden durch das Register Renaming gelöst?

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
`lw`															
`mul`															
`sw`															
`slli`															
`addi`															
`div`															

b) Gegeben ist nun folgende aus der Vorlesung bekannte simple superskalare RISC-V Pipeline mit Dual Fetch, Decode und Issue, die über Forwarding verfügt (im Vergleich zur Konfiguration aus der Vorlesung wird lediglich die Größe des Issue Buffer auf 8 erhöht):

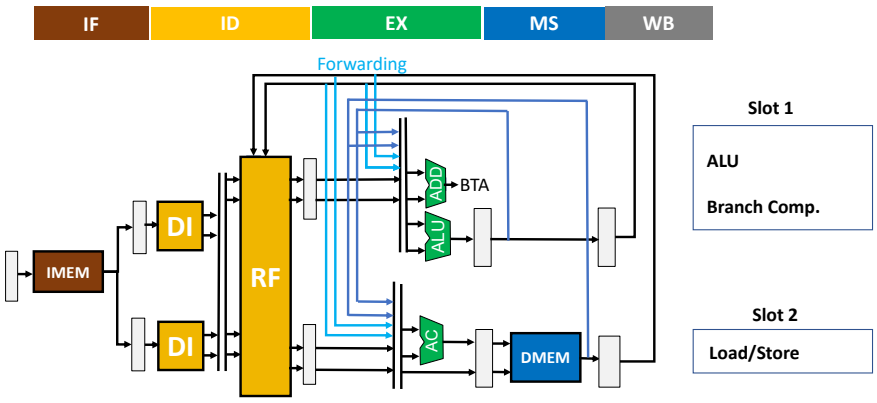


Bringen Sie den Code mit Verwendung von Register Renaming auf dieser Pipeline zur Ausführung, d.h. zeigen Sie das resultierende Pipelinediagramm und ermitteln Sie den erreichten IPC-Wert.

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
`lw`															
`mul`															
`sw`															
`slli`															
`addi`															
`div`															

Aufgabe 2: Advanced Pipelines: VLIW

Gegeben ist folgende RISC-V Pipeline mit Static Dual Issue, wobei Slot 1 ALU-/Branch-Instruktionen und Slot 2 Load-/Store-Instruktionen ausführen kann:



Gegeben ist zudem folgender RISC-V Code:

```
1        addi  a5,a0,0
2        addi  a2,a0,4
3        addi  a0,zero,0
4  loop: lbu   a4,0(a5)
5        lbu   a3,0(a1)
6        addi  a5,a5,1
7        addi  a1,a1,1
8        sll   a4,a4,a3
9        add   a0,a0,a4
10       bne   a5,a2,loop
```

a) Schedulen Sie *diesen* Code für die gegebene Pipeline.

	Slot 1: ALU/Branch	Slot 2: Load/Store

Geben Sie die Performance (IPC) an, wenn der hier erstellte Schedule auf der beschriebenen Pipeline ausgeführt wird. Nehmen Sie hier an, dass eine statische Branch Prediction (BTFNT) zum Einsatz kommt und möglicherweise benötigte Einträge im BTB bereits vorhanden sind.

IPC:

b) Schedulen Sie den Code für die gegebene Pipeline, wobei zusätzlich – soweit möglich – Loop Unrolling zum Einsatz kommen soll. Sie können annehmen, dass die Register `t0-t6` sowie das Register `a7` verwendet werden können, falls zusätzliche Register benötigt werden sollten.

Führen Sie als Zwischenschritt zunächst das Loop Unrolling durch und erstellen Sie danach den Schedule für die oben angegebene Static Dual Issue RISC-V Pipeline durch Ausfüllen der Tabelle.

Loop Unrolling:

Schedule:

	Slot 1: ALU/Branch	**Slot 2: Load/Store**

Geben Sie die Performance (IPC) für den Fall an, dass der hier erstellte Schedule auf der beschriebenen Pipeline ausgeführt wird. Nehmen Sie hier an, dass eine statische Branch Prediction (BTFNT) zum Einsatz kommt und möglicherweise benötigte Einträge im BTB bereits vorhanden sind.

IPC:

Aufgabe 3: Heterogeneous SoCs: On-chip Bus

Gegeben ist die folgende Architektur für einen Layered Bus:

- Zwei Initiatorkomponenten I_1 und I_2
- Vier Zielkomponenten T_1, T_2, T_3, T_4, wobei sich T_1 und T_2 auf Layer 1 befinden und T_3 und T_4 auf Layer 2.

a) Erstellen Sie eine Abbildung der Komponenten und der Verbindungen.

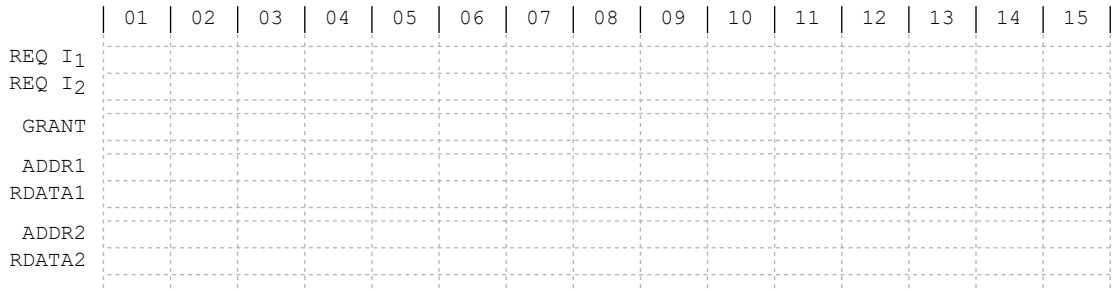
b) Nehmen Sie an, dass I_1 in Zyklus 1 lesend auf T_3 zugreifen will. I_2 fragt ebenfalls in Zyklus 1 Daten (lesend) von T_2 an. In Zyklus 8 fragt I_2 erneut Daten (lesend) von T_2 an. In Zyklus 10 fragt I_1 Daten (lesend) von T_1 an.

Zeichnen Sie ein Diagramm mit den Buszugriffen. Nehmen Sie an, dass T_2 einen und T_3 drei Wartezyklen erfordert. T_1 und T_4 erfordern keine Wartezyklen. Die Arbitrierungsreihenfolge lautet: I_1, I_2 (d.h. I_1 wird zuerst berücksichtigt). Pipelining wird nicht unterstützt.

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
REQ I_1															
REQ I_2															
GRANT LAYER1															
GRANT LAYER2															
ADDR LAYER1															
RDATA LAYER1															
ADDR LAYER2															
RDATA LAYER2															

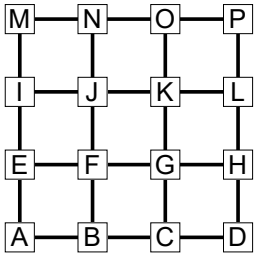
c) Zeichnen Sie ein Diagramm für die gleichen Buszugriffe wie in Aufgabenteil (b) beschrieben, nun soll jedoch statt des oben angegebenen Layered Bus ein Shared Bus (mit Pipelining) betrachtet werden.

Nehmen Sie an, dass alle Targetkomponenten zwei Wartezyklen erfordern. Die Arbitrierungsreihenfolge lautet: I_1, I_2 (d.h. I_1 wird zuerst berücksichtigt).

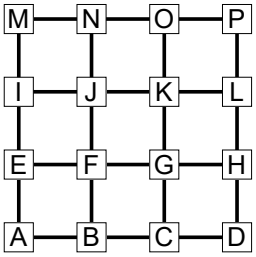


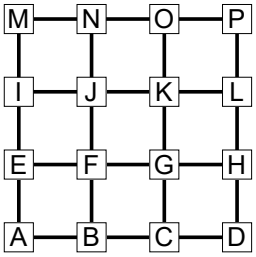
Aufgabe 4: Heterogeneous SoCs: NoC – Routing

Gegeben ist das folgende On-Chip Interconnection Network (4-ary 2-cube):

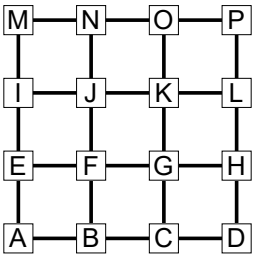


a) Zeigen Sie den Pfad von `E` nach `P` bei Verwendung von Dimension-order Routing für (i) die Variante XY und (ii) die Variante YX.

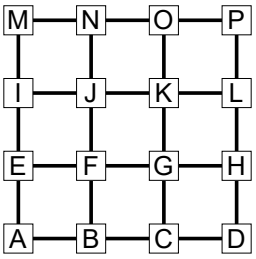




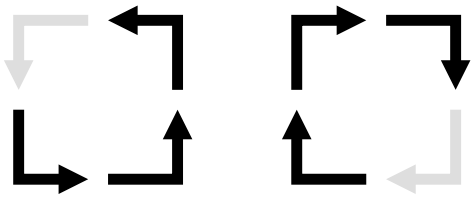
b) Gibt es noch weitere minimale Pfade von `E` nach `P`? Falls *ja*, geben Sie ein Beispiel, falls *nein* begründen Sie.



c) Geben Sie ein Beispiel für einen minimalen Pfad von `E` nach `P`, der bei der minimalen Version von Valiant's Algorithm in Verbindung mit XY Dimension-order Routing nicht verwendet wird, auch wenn alle möglichen Knoten als Zwischenknoten d' in Betracht gezogen werden. Begründen Sie Ihre Lösung.

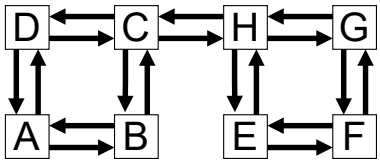


d) Um Deadlocks zu vermeiden, besteht eine Möglichkeit darin beim Routing das Abbiegen in bestimmte Richtungen zu verbieten und so abstrakte Zyklen zu verhindern (vgl. “turn model”). Ist die Auswahl der nachfolgend angegebenen untersagten Richtungsänderungen (grau) in dieser Hinsicht erfolgversprechend? Begründen Sie Ihre Einschätzung.



Aufgabe 5: Heterogeneous SoCs: NoC – CDG

Gegeben ist das folgende On-Chip Interconnection Network:



a) Im Vergleich zu bisher vorgestellten Topologien (wie z.B. Meshes) sind hier nicht alle benachbarten Router miteinander verbunden: Welche Argumente könnten für die Wahl dieser Topologie sprechen?

b) Erstellen Sie den CDG (Channel Dependence Graph) für das Netzwerk (nehmen Sie an, dass 180°-Wenden nicht erlaubt sind).

c) Basierend auf dem CDG: Besteht hier die Möglichkeit für Deadlocks und wie kann dies anhand des CDG festgestellt werden? Begründen Sie Ihre Einschätzung anhand des CDG.

d) Falls Deadlocks auftreten können: Schlagen Sie eine Lösung unter Zuhilfenahme des CDG und der Ergebnisse von Aufgabenteil (c) vor. Zeigen Sie den resultierenden CDG.

Aufgabe 6: Heterogeneous SoCs: Vector

Gegeben ist der folgende C-Code, bei dem die Elemente der Arrays `x` und `y` vom Typ `int` sind:

```
1  for ( int i = 0; i < N; i++ )
2  {
3    y[i] = a * x[i] + y[i];
4  }
```

Dieser Code wird auf einem RISC-V-Vektorprozessor ausgeführt. Nehmen Sie an, dass die Laufzeiteinstellungen wie folgt sind:

- *VLEN* (Anzahl der Bits in einem einzelnen Vektorregister) beträgt 128 Bits.
- *LMUL* (Anzahl der Vektorregister, die zu einer Vektorregistergruppe kombiniert werden) beträgt 2.
- *SEW* (ausgewählte Elementbreite) beträgt 32 Bits.

a) Bestimmen Sie die maximale Anzahl von Elementen, auf die mit einer einzelnen Vektorinstruktion zugegriffen werden kann ($VLMAX$). Angenommen vl (die Vektorlänge) beträgt 6, zeichnen Sie die Datenlayouts der Vektorregister basierend auf den obigen Einstellungen.

b) Gegeben ist nun eine weitere C-Funktion:

```
1  void vecaddscalar (int *x, int a)
2  {
3    for ( int i = 0; i < 8; i++ )
4    {
5      x[i] = x[i] + a;
6    }
7  }
```

Die entsprechenden RISC-V-Skalarinstruktionen für den gegebenen C-Code lauten wie folgt:

RISC-V Skalarinstruktionen:

```
1  vecaddscalar:
2      li   t0,8
3  loop:
4      lw   t1,0(a0)    # The first parameter x is provided in a0.
5      add  t1,t1,a1    # The second parameter a is provided in a1.
6      sw   t1,0(a0)
7      addi t0,t0,-1
8      addi a0,a0,4
9      bnez t0,loop
10     ret
```

Erklären Sie den Zweck der folgenden zwei RISC-V-Instruktionen:

li t0, 8 und **vsetvli t2, t0, e32, m2, ta, ma**

Ergänzen Sie weiterhin die fehlenden Vektorinstruktionen im folgenden Code, so dass die Funktionalität der `vecaddscalar`-Funktion aus dem oben gezeigten C-Code implementiert wird. Erklären Sie jeweils den Zweck der hinzugefügten Instruktionen.

RISC-V Vektorinstruktionen:

```
1  vecaddscalar:
2    li       t0,8
3    vsetvli  t2,t0,e32,m2,ta,ma
4    ... #Ergänzen Sie den Code hier
5    ret
```