Hochschule Konstanz
Fakultät Informatik

Prof. Dr.-Ing. Michael Blaich
Robotik und Künstliche Intelligenz

Übung Rechnerarchitekturen AIN 2 SoSe 2025

4. Hazards, Multi Cycle CPU

Die Abgabe erfolgt durch Hochladen der Lösung in Moodle. Zusätzlich wird die Lösung in der Übung nach dem Abgabetermin stichprobenartig kontrolliert.

Bearbeitung in Zweier-Teams

Team-Mitglied 1:

Team-Mitglied 2:

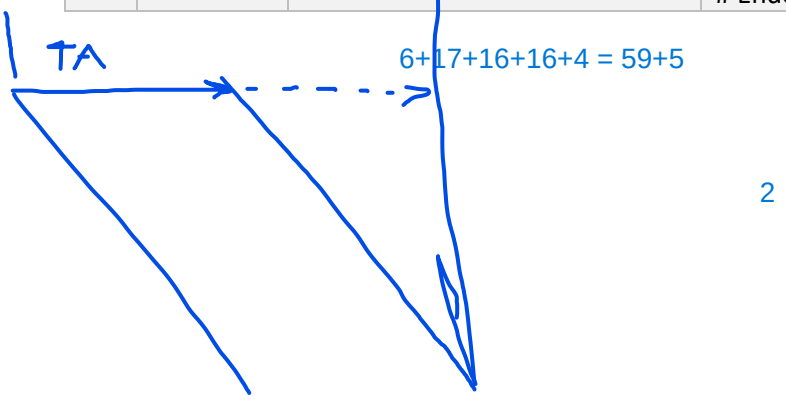
Aufgabe 4.1 Hazards

In dieser Aufgabe soll ein Assembler-Code zur Berechnung des element-weisen Maximums zweier Arrays auf Hazards untersucht werden. Danach sollen die Hazards zunächst mit Forwarding und dann zusätzlich mit Code-Scheduling soweit wie möglich beseitigt werden.

Die Code-Sequenz für diese Aufgabe bestimmt elementweise das Maximum von zwei Arrays B und C und speichert das Ergebnis im Array A ab.

$$A = \max(B, C) \text{ mit } A_i = \max(B_i, C_i) \text{ für alle } i$$

	.data		
	A:	.word 0 0 0	# Array A
	B:	.word 1 2 3	# Array B
	C:	.word 3 2 1	# Array C
	n:	.word 3	# n: Elemente im Array
	.text		
		la $s0,A	# Adresse von A in $s0 laden, $s0=&A[0]
		la $s1,B	# Adresse von B in $s1 laden, $s1=&B[0]
		la $s2,C	# Adresse von C in $s2 laden. $s2=&C[0]
		lw $s3,n	# n in $s3 laden
1		sll $t3,$s3,2	# $t3=4*n
2		add $t7,$t3,$s0	# $t7: Adr. des 1. Worts nach A, $t7=&A[n]
3	loop:	beq $s0,$t7,end	# Ende, falls $s0 über A hinausläuft
4		lw $t1,0($s1)	# B[i] in $t1 laden
5		lw $t2,0($s2)	# C[i] in $t2 laden
6		slt $t3,$t1,$t2	# wenn $t1<$t2
7		bne $t3,$zero,nimm2	# springe zu nimm2
8		sw $t1,0($s0)	# A[i]=B[i]: $t1 an Speicherstelle $s0=&A[i]
9		j next	# überspringe "else"
10	nimm2:	sw $t2,0($s0)	# A[i]=C[i]: $t2 an Speicherstelle $s0=&A[i]
11	next:	addi $s0,$s0,4	# Adr. des nächsten Elem. in A, $s0=&A[i+1]
12		addi $s1,$s1,4	# Adr. des nächsten Elem. in B, $s1=&B[i+1]
13		addi $s2,$s2,4	# Adr. des nächsten Elem. in C, $s2=&C[i+1]
14	23	j loop	# in die nächste Schleife springen
15	end:		# Ende des Programms



Betrachten Sie den eigentlichen Algorithmus mit den nummerierten Programmzeilen.

4.1.1 Identifizieren Sie alle Control und Data Hazards.
Hinweise:
- Gehen Sie davon aus, dass der jump Befehl die Adresse der nächsten Instruktion bereits in der IF-Stage berechnet.
- Gehen Sie davon aus, dass die Pipeline Sprungbedingung und Sprungziel mit zusätzlicher Hardware in der ID Stage bestimmt.

4.1.2 Wie viele Takte benötigt der Algorithmus bei der obigen Eingabe und wo entstehen Bubbles, wenn **kein Forwarding** verwendet wird und die Pipeline bei bedingten Sprüngen auf das Ergebnis wartet. 64

4.1.3 Welche Data Hazards können durch Forwarding umgangen werden? Welche Bubbles bleiben trotz Forwarding bestehen? Alle, außer Z6: braucht 1 Bubble wegen lw

4.1.4 Optimieren Sie das Programm durch Code-Scheduling. Z12-13 zwischen Z5-6

4.1.5 Es treten immer noch Control-Hazards auf. Betrachten Sie den Fall, dass die Pipeline anstatt leer zu laufen, immer die nächste Instruktion im Speicher ausführt (Annahme: kein Sprung). Bei einer Fehlentscheidung wird die Instruktion wieder aus der Pipeline gelöscht. Wie viele „Bubbles“ durch Control Hazards gibt es im Mittel bei einer Arraygröße von 100 Elementen?

Können wir so nicht sagen, weil es abhängig ist ob B[i] < C[i]

min. 2 Bubbles + 2 Bubbles für jedes B[i] < C[i]

Aufgabe 4.2 Multi Cycle CPU - Pipeline

Betrachten Sie folgenden Assembler-Code:

	.data		
	A:	.word 0 0 0	# Array A
	B:	.word 1 2 3	# Array B
	C:	.word 3 2 1	# Array C
	n:	.word 3	# n: Elemente im Array
	.text		
		la $s0,A	# Adresse von A in $s0 laden, $s0=&A[0]
		la $s1,B	# Adresse von B in $s1 laden, $s1=&B[0]
		la $s2,C	# Adresse von C in $s2 laden. $s2=&C[0]
		lw $s3,n	# n in $s3 laden
1		sll $t3,$s3,2	# $t3=4*n
2		add $t7,$t3,$s0	# $t7: Adr. des 1. Worts nach A, $t7=&A[n]
3		lw $t1,0($s1)	# B[i] in $t1 laden
4		lw $t2,0($s2)	# C[i] in $t2 laden
5	loop:	beq $s0,$t7,end	# Ende, falls $s0 über A hinausläuft
6		slt $t3,$t1,$t2	# wenn $t1<$t2

Vollziehen Sie detailliert den Ablauf der Instruktionen von **Zeile 1** bis inklusive **Zeile 6** nach, indem Sie die Inhalte aller Pipeline-Register und Forwarding-Entscheidungen protokollieren. Gehen Sie davon aus, dass Sprungbedingungen mit zusätzlicher Hardware in der ID Stage evaluiert werden. Zudem sei Forwarding sowohl für die Evaluierung der Sprungbedingung in der ID Stage als auch für die Berechnungen in der EX Stage unterstützt. Abbildung 4.1 zeigt ein Single-Clock-Cycle-Pipeline-Diagramm dieser CPU. Das Forwarding für die Evaluierung der Sprungbedingung ist im Diagramm nicht enthalten um die Übersichtlichkeit zu verbessern.

Füllen Sie dazu die in Moodle bereitgestellte Excel-Tabelle aus. Die Instruktionen inklusive der binären Notation sind bereits eingetragen.

Erklärung: Die oberste Tabelle bezieht sich auf die **ID Stage**. In den Spaltenköpfen sollen alle Werte stehen, die am Ende der ID Stage in das ID/EX-Pipeline-Register geschrieben werden. Zudem sollen alle Steuersignale eingetragen werden, die in der ID Stage benutzt werden. Die zweite Tabelle bezieht sich auf die **EX Stage** und die dritte Tabelle auf die **MEM Stage**.

Abbildung 4.2 zeigt Ausschnitte aus dieser Tabelle.

4.2.1 Tragen Sie in die Abbildung 4.2 in die weißen Kreise die entsprechenden Nummern oder Buchstaben aus Abbildung 4.1 ein. Sie finden diese in der Abbildung 4.1 in den grau hinterlegten Kreisen.

4.2.2 Füllen Sie die Tabellen aus, in dem Sie für die jeweilige Instruktion eintragen,

a. wie die Steuersignale pro Stage bei der Berechnung gesetzt waren (das bezieht sich auf die mit Control/Other beschrifteten Spalten)
b. welche Einträge (DataPath-Werte und Control-Signale) am Ende des Taktes im Pipeline-Register stehen
c. aus welchen Registern die Operanden in der ID (Vergleich der Sprungadressen) und EX Stage (ALU Operation) entnommen wurden (das bezieht sich auf die mit Control/Forwarding beschrifteten Spalten)

Hinweis: Es sind schon Zeilen in den Tabellen beispielhaft ausgefühlt. Sie können sich daran für die nächsten Zeilen orientieren.

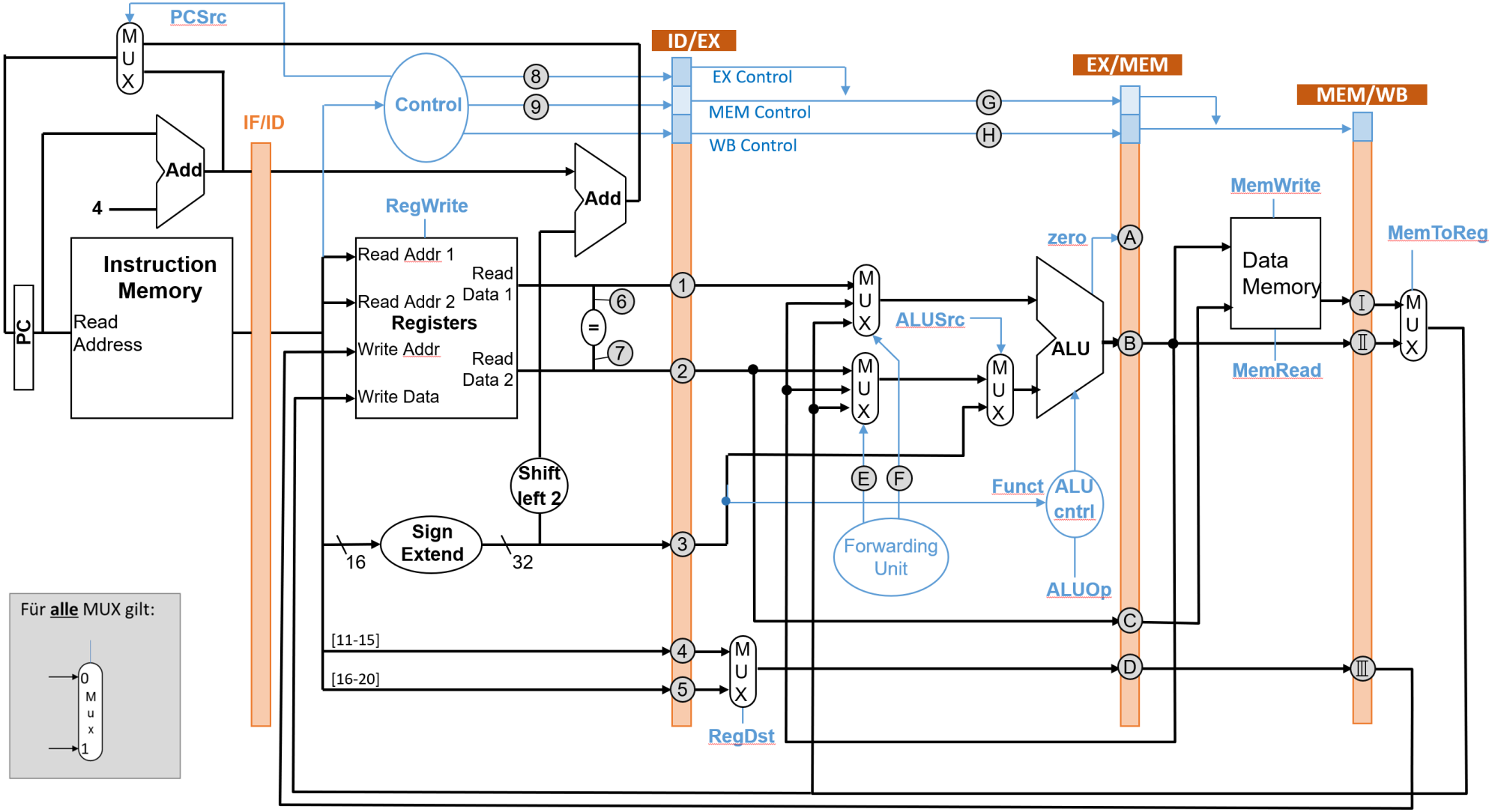


Abbildung 4.1 Single-Clock-Cycle-Pipeline-Diagramm

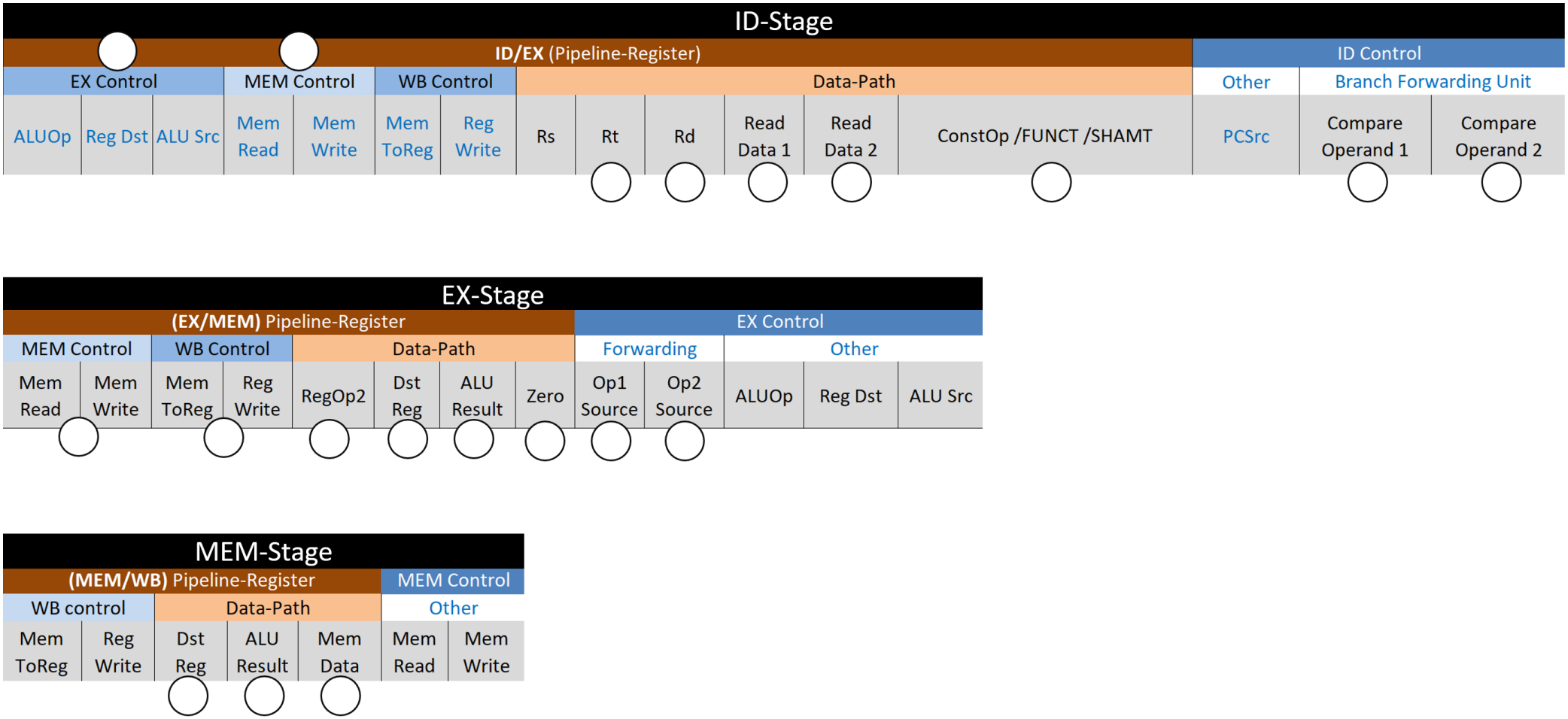


Abbildung 4.2 Zuordnung der Tabellenköpfe