Rheinische Friedrich-Wilhelms Universität Bonn
Institut für Informatik VI, Technische Informatik

Prof. Dr. Joachim K. Anlauf
Oguzhan Sezenlik

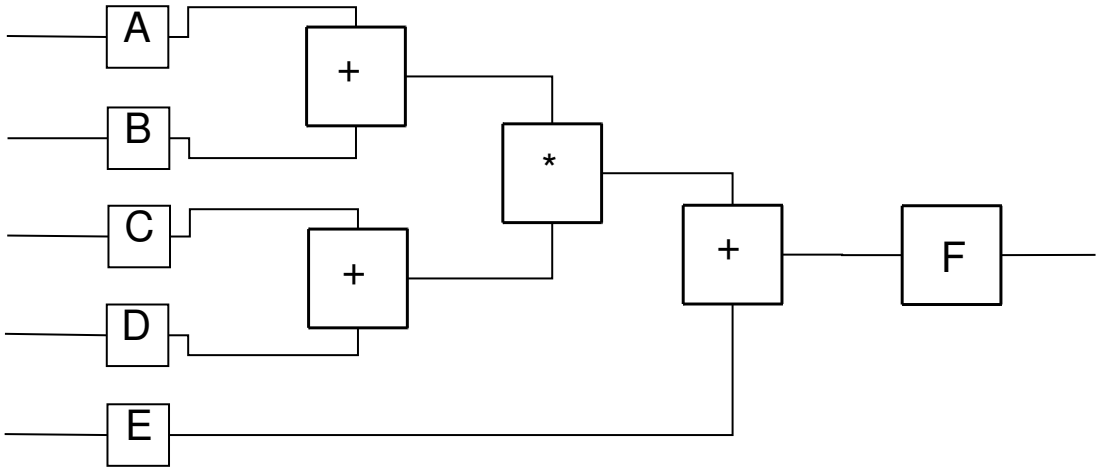
9. Übung für die Vorlesung
Rechnerorganisation

Sommersemester 2019

Abgabe: Donnerstag, 27.6.2019

Aufgabe 1. *Pipeline-Architektur* 6 P.

Gegeben ist folgender Datenpfad



Der Addierer benötigt 10 ns und der Multiplizierer benötigt 20 ns für die Berechnung. Bei den Blöcken A bis F handelt es sich um Register, deren Durchlaufzeiten zu vernachlässigen sind.

1. Maximieren Sie den Durchsatz der Schaltung, indem Sie eine Pipeline entwerfen.
2. Wie lang müssen Sie die Taktperiode für die Pipelineimplementierung mindestens wählen? Welche maximale Taktfrequenz folgt daraus?
3. Welche Latenzzeit weist die Pipeline aus Teilaufgabe 1 mindestens auf? Gehen Sie bei der Berechnung davon aus, dass die Eingabewerte bereits in den Registern A bis E stehen.
4. Was müsste man tun, um den Durchsatz der Pipeline zu verdoppeln?

Aufgabe 2. *Pipeline-Architektur* 6 P.

Gegeben sei die folgende Schaltung. Die quadratischen Kästchen in der Schaltung bezeichnen Gatter, deren jeweilige Zeitverzögerung in *ns* eingetragen ist.

1. Wie groß ist die Zeitverzögerung der gesamten Schaltung ohne Pipelining?
2. Führen Sie Pipelining in der Schaltung ein, ohne die Funktionalität der Schaltung zu verändern. Zeichnen Sie hierzu Register in die Abbildung ein (einfache, unbeschriftete Kästchen genügen). Wichtigstes Kriterium für die Pipeline soll die Maximierung der Taktfrequenz sein. Anschließend soll noch berücksichtigt werden, dass die Latenzzeit möglichst gering sein sollte. Überlegen Sie sich, welches Gatter die maximale Taktfrequenz bestimmt.
3. Wie hoch ist die Latenzzeit Ihrer Pipeline?
4. Wie ist die minimale Periodendauer eines Taktsignals, mit dem Sie die Pipeline takten dürfen?

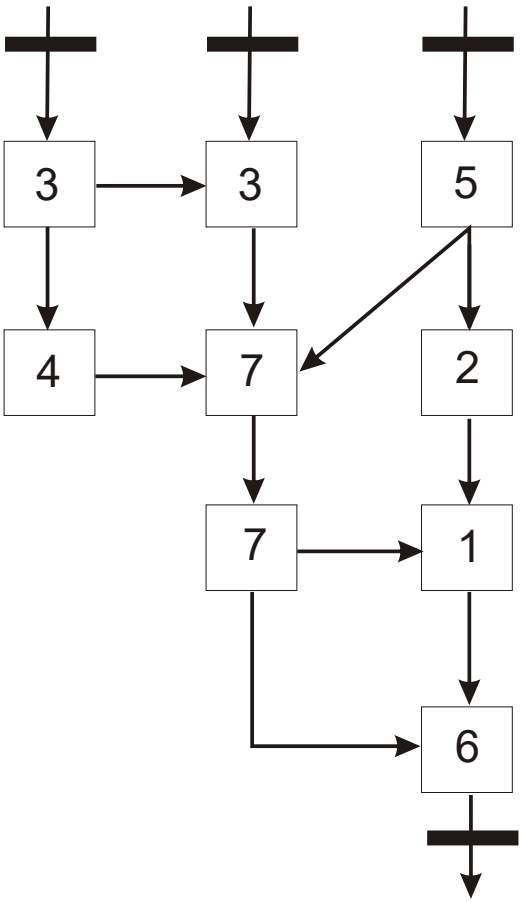


Abbildung 1: Schaltung ohne Pipelining

Aufgabe 3. *Datenabhängigkeiten* 10 P.

Betrachten Sie den folgenden Assembler Code, der von einem MIPS Prozessor mit einer 5-stufigen Pipeline (siehe Abbildung 2) ausgeführt werden soll.

```
1: la   $t2, array
2: lw   $t1, 8 ($t2)
3: addi $t2, $t3, 42
4: sub  $t4, $t1, $t3
5: ori  $t1, $t4, 17
6: add  $t3, $t1, $t2
7: li   $t2, 25
[...]
array: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

1. Untersuchen Sie den Inhalt der genutzten Register ($t1 bis $4) für 12 Taktzyklen. Tragen Sie in die nachfolgende Tabelle den jeweiligen Registerinhalt in jedem Taktzyklus ein. Einfachheitshalber können Sie davon ausgehen, dass die Register initial den Wert 0 enthalten und die Pipeline leer ist. Das Label `array` verweise auf die Adresse `0x4` im Datenspeicher.
   Hinweis: la ist eine Pseudoinstruktion, das in ein addi übersetzt wird, falls der Immidiate-Wert mit 16 Bits dargestellt werden kann

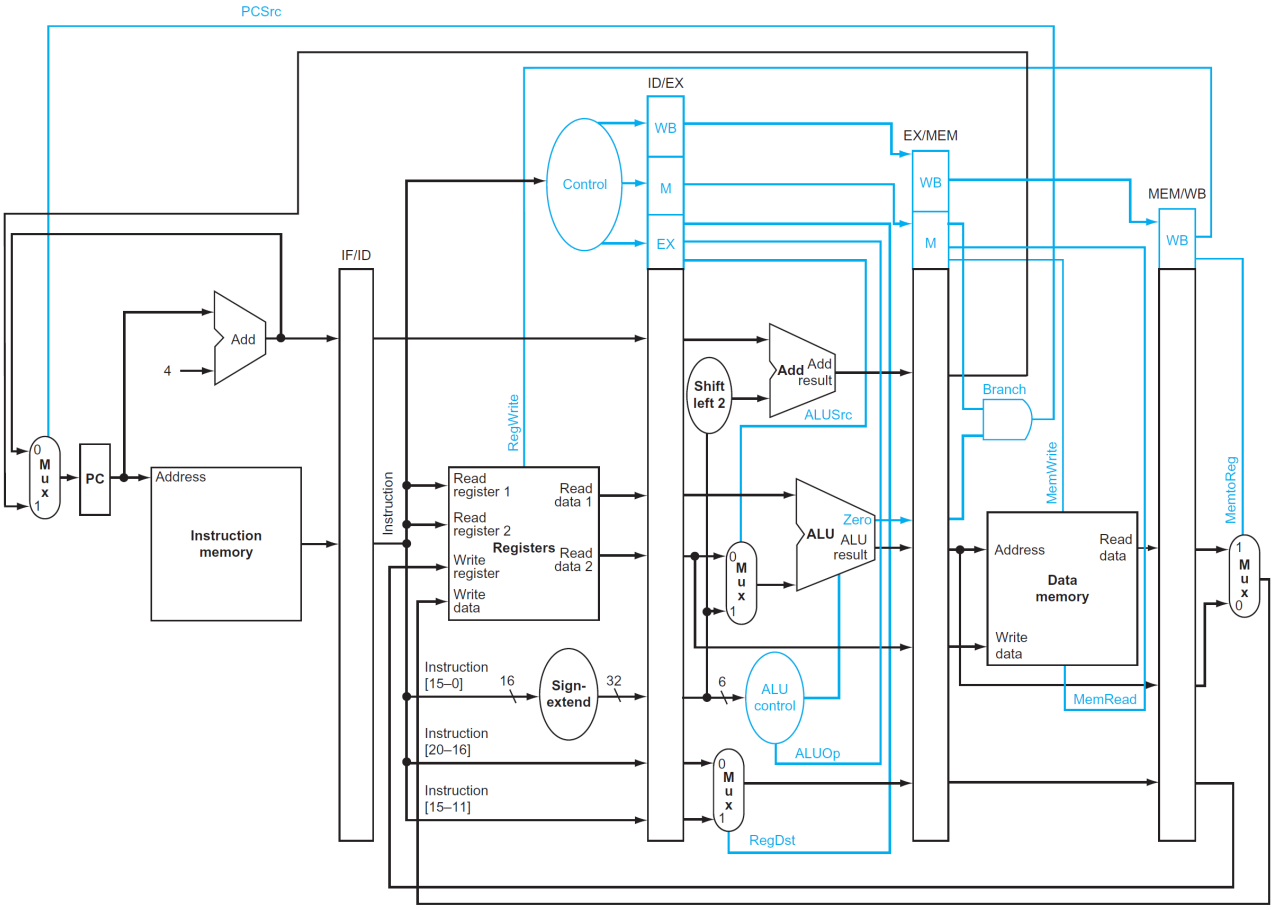


Abbildung 2: Datenpfad mit Pipeline-Registern

Taktzyklus	0	1	2	3	4	5	6	7	8	9	10	11	12
`$t1`	0												
`$t2`	0												
`$t3`	0												
`$t4`	0												

2. Welche Arten von Abhängigkeiten zwischen Maschinenbefehlen kennen Sie?
3. Kennzeichnen und klassifizieren Sie alle Abhängigkeiten im vorliegenden Programmcode.
4. Wie können diese Datenabhängigkeiten von einem Compiler aufgelöst werden?
5. Übernehmen Sie die Aufgabe des Compilers und lösen Sie alle Datenabhängigkeiten auf. Stellen Sie für den verbesserten Assembler Code erneut eine Tabelle wie in Teilaufgabe 1 auf.

Aufgabe 4. *Modifikation des MIPS-Register-Files* 4 P.

Betrachten Sie den MIPS Register File, siehe Vorlesungsfolien Seiten 123-124. Ein Datenabhängigkeitskonflikt kann behoben werden, indem beim Lesen eines Registers, das im selben Takt beschrieben wird, nicht der alte Registerinhalt ausgelesen wird, sondern die neuen Daten, die gerade erst hineingeschrieben werden.

Entwerfen Sie den neuen Register File und geben Sie den vollständigen Schaltplan an.