Übung 7

Abgabe bis **Donnerstag, 27. Juni 10:15** via EPIIC: `https://ep.iic.jku.at`.

Triff für den MIPS-Prozessor mit fünfstufiger Pipeline folgende Annahmen:

- Ein Jump wird in der Stufe *decode* ausgeführt.
- Ein Branch wird in der Stufe *execute* ausgeführt.
- Geschriebene Werte sind erst verfügbar nachdem die Stufe *write back* abgeschlossen ist.
- Der Prozessor hat keine Mechanismen um Pipeline-Konflikte zu vermeiden.

1. Pipelining Speedup (8 Punkte)

Nimm an, dass die Komponenten des MIPS-Prozessors die in Tabelle 1 angegebenen Verzögerungen haben (eine Verzögerung gilt zwischen jedem Eingang und Ausgang der Komponente). Für nicht gelistete Komponenten soll eine Verzögerung von 0 ps angenommen werden.

(a) Zeichne die *kritischen Pfade* aller Pipelining-Stufen des MIPS-Prozessors in Abbildung 1 ein und bestimme deren Länge (in ps).

(b) Bestimme die minimale Länge des Taktzyklus für den MIPS-Prozessor mit fünfstufiger Pipeline (Abbildung 1). Welcher Speed-up kann im Vergleich zum Prozessor ohne Pipeline (Abbildung 2, vgl. kombinatorischer Prozessor im letzten Übungszettel) theoretisch erreicht werden? Warum ist dieser Speed-up nicht gleich der Anzahl der Pipeline-Stufen?

Tabelle 1: Verzögerungen der Prozessorkomponenten

Komponente	Verzögerung [ps]
Addierer	125
ALU	200
Instruction Memory	250
Data Memory	275
Register file lesen	100
Register file schreiben	125
Control Unit	75
Multiplexer	30
UND-Gatter	25
Lesen eines Register	20
Schreiben eines Register	40

2. Datenabhängigkeiten und Pipeline-Konflikte (10 Punkte)

Betrachte das folgende MIPS-Programm und nimm an, dass es auf einem MIPS-Prozessor mit fünfstufiger Pipeline (siehe Abbildung 1) ausgeführt werden soll.

```
s1    sub  $t0, $t1, $t2
s2    addi $t2, $t3, -2
s3    add  $t4, $t2, $t1
s4    beq  $t0, $0, else
s5    addi $t2, $t5, 8
s6    lw   $t0, 8($t3)
s7    sub  $t3, $t0, $t5
s8    j end
    else:
s9    sw   $t3, 4($t0)
s10   or   $t0, $t3, $t4
s11   add  $t5, $t2, $t4
s12   xor  $t4, $t1, $t0
    end:
s13   xor  $t2, $t3, $t0
s14   and  $t6, $t6, $t1
```

(a) Gib alle Datenabhängigkeiten zwischen den Befehlen des Programms an (RAW, WAR und WAW).

(b) Finde alle auftretenden Pipeline-Konflikte (Daten- und Kontroll-Konflikte). Gib jeweils die involvierten Befehle an und erkläre warum es zu einem Konflikt kommt.

(c) Behebe alle Pipeline-Konflikte durch Einfügen der minimalen Anzahl an NOP-Befehlen.

(d) Minimiere die Anzahl der NOP-Befehle durch Umordnen der Befehle (ohne die Semantik des Programms zu verändern). Nimm dabei an, dass der Prozessor auch das Löschen (flushen) von Pipeline-Registern unterstützt.

3. Flushen der Pipeline (6 Punkte)

Erweitere den MIPS-Prozessor in Abbildung 1, sodass das Löschen (*flushen*) der Pipeline bei den Befehlen `beq` und `j` unterstützt wird. Füge dazu möglichst wenig Logik zum Datenpfad des Prozessors hinzu.

Hinweis: Ermittle zuerst wie viele „falsche“ Befehle nach einem `beq` oder `j` in die Pipeline eingereiht werden, bis erkannt wird dass, gesprungen werden soll. Führe dann Anpassungen in den entsprechenden Stufen der Pipeline durch, sodass im Fall eines Sprungs die Abarbeitung dieser Befehle in den weiteren Stufen der Pipeline keinen Effekt mehr hat.

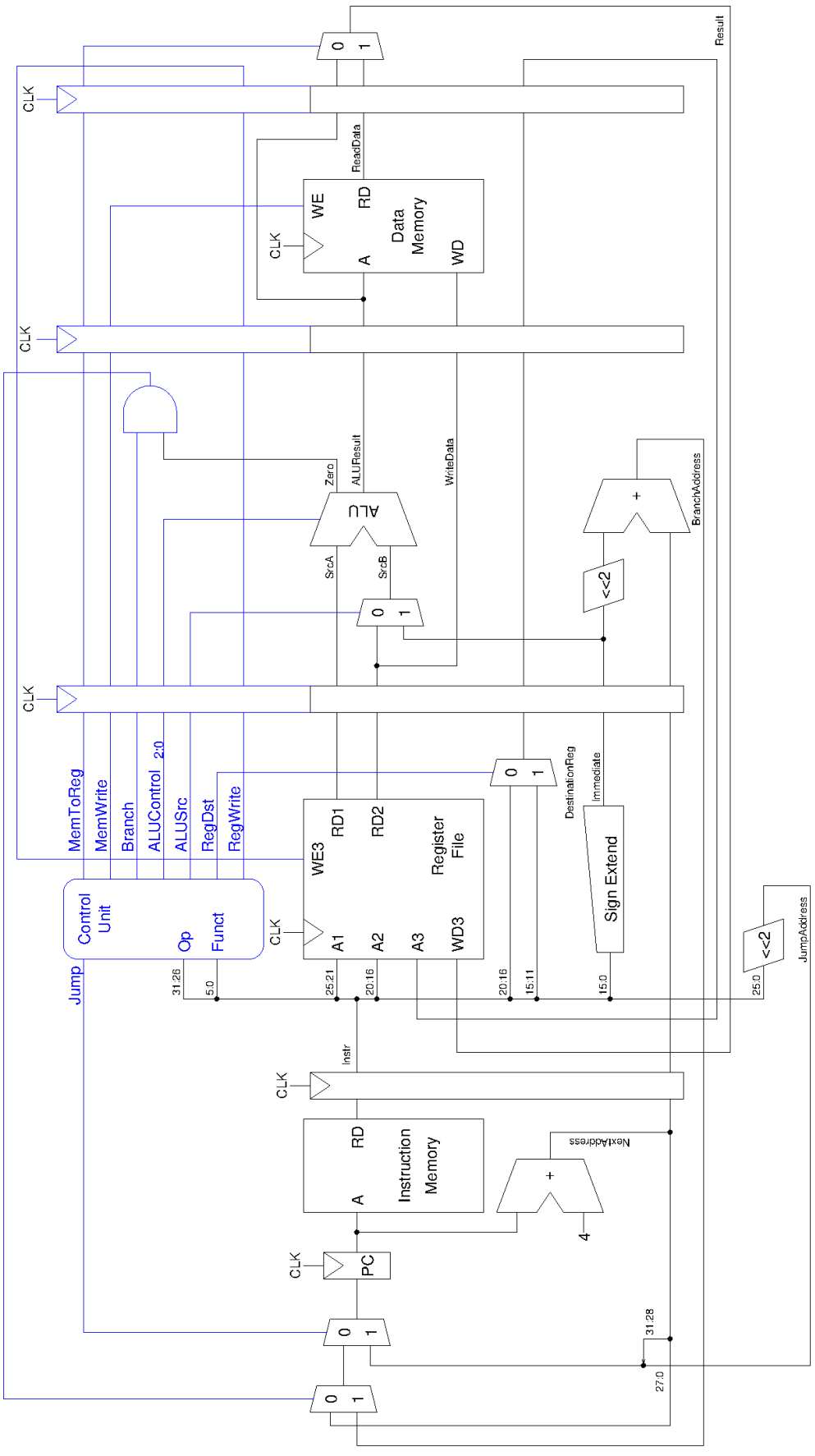

Abbildung 1: MIPS-Prozessor mit fünfstufiger Pipeline

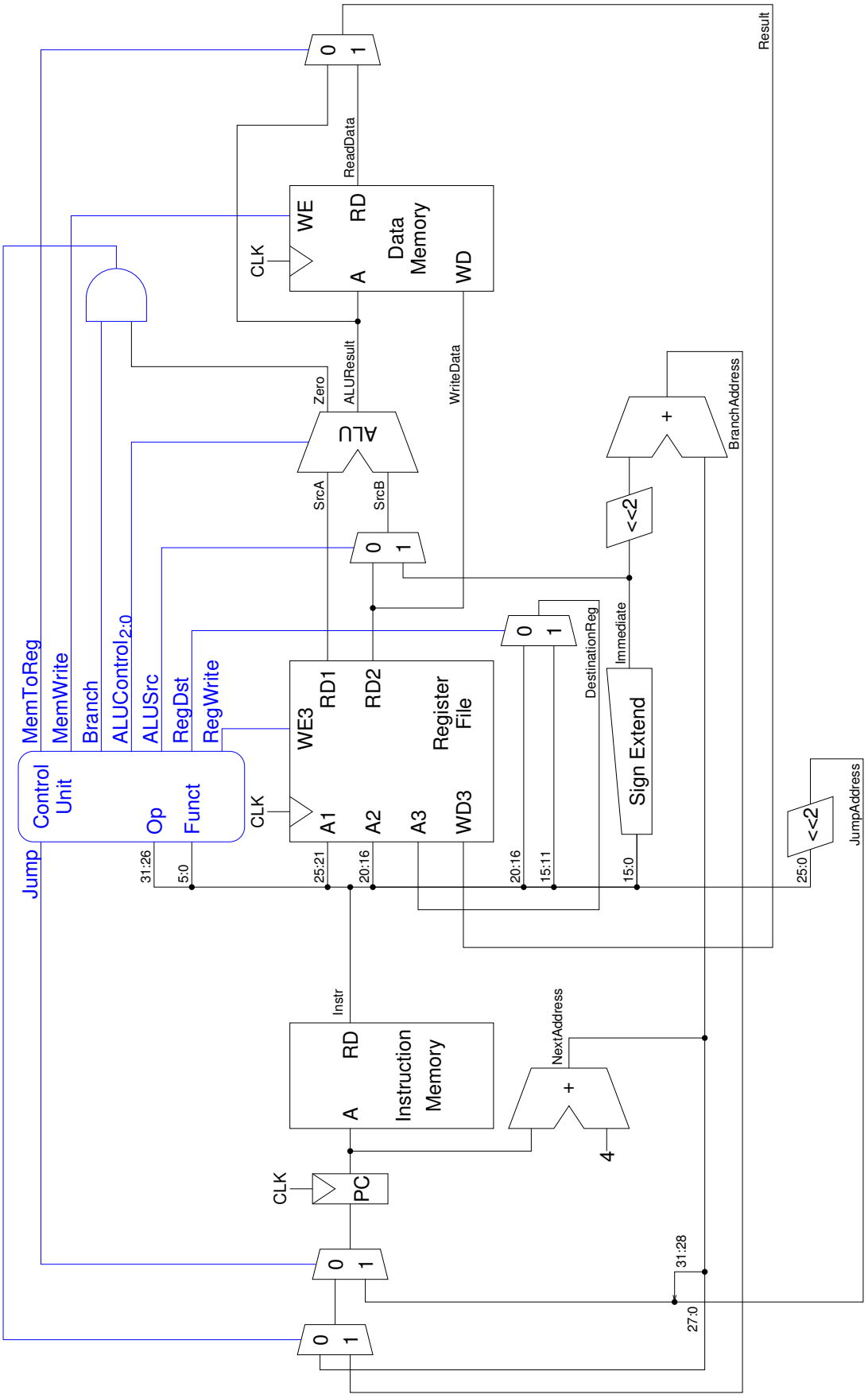

Abbildung 2: MIPS-Prozessor ohne Pipelining