

Hoang Long Vu - 231688

Duc Huy Nguyen - 231611

Aufgabe 1:

a) 3 Schwächen des Singlecycle MIPS-Prozessors:

- + Es braucht eine genug lange Taktzeit für die langsame Instruktion(en) (am langsamsten `lw`)
 - z.B.: Datenpfad von `lw $a1, -4($sp)`: liest das Instr-Mem → liest das Register `$sp` → berechne Memory-Adresse von `$sp-4` → liest das Data-Mem → speichert Daten ins `$a1`
 - Falls wir die Taktzeit z.B. an 10s setzen, dann läuft alle Instruktionen je 10ns, auch wenn diese Instruktionen nicht so viel Zeit benötigt.
 - Im Vergleich zu `add $a3, $a1, $a2`: liest das Instr-Mem → liest die Register `$a1` und `$a2` → berechnet `$a1 + $a2` → speichert das Ergebnis ins `$a3`.

Angenommen jedes Schritt braucht 2ns Ausführungszeit, dann braucht `lw` 10ns aber `add` nur 8ns.

+ Es braucht insgesamt 3 Addierer (1 bei ALU und 2 für das PC-Logik). Schnelle Addierer sind teuer.

+ Die Instr- und Data-Mem sind separat, da wir jede Komponente nur einmal je Takt zugreifen können (wie z.B. `lw` braucht die beide Memory-Komponenten). Moderne Prozessoren haben aber Instr- und Data-Mem zusammen in einer großen Komponente.

b. Multicycle MIPS-Prozessor breicht einer Instruktion über mehreren Takte.

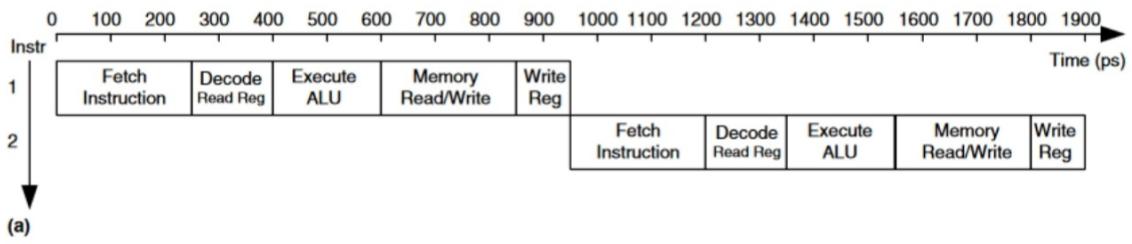
1. Verschiedene Instruktionen brauchen unterschiedliche Anzahl von Takte / Schritte → einfachere Instruktionen können schneller abgearbeitet werden, ohne Zeit zu verschwenden.

2. Der Prozessor benötigt nur einen Addierer, dieser wird für verschiedene Zwecke in verschiedenen Schritte wiederverwendet.

3. Die Instr- und Data-Mem sind in einer Komponente zusammengesetzt. Wir können dann die Instruktion in einem Takt lesen, und Daten lesen / schreiben in einem anderen Takt

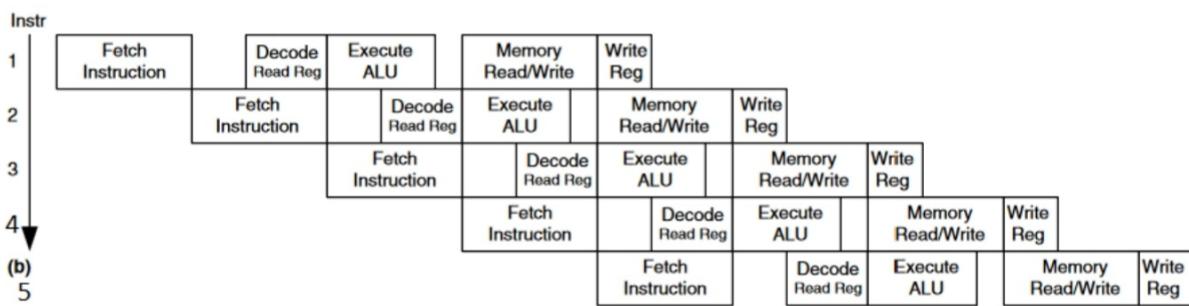
Aufgabe 2:

- a) Bei Singlecycle wird es jeweils nur eine Instruktion ausgeführt (wie die Abbildung 7.43 a). Wir entwerfen einen Pipeline-Prozessor durch Unterteilen den Singlecycle-Prozessor in fünf Pipelinestufen. Dadurch können fünf Instruktionen gleichzeitig ausgeführt werden, einer in jeder Stufe, und keine gleichen Operationen werden parallel ausgeführt



(a)

7.43



(b)

1) Da es bei jeder Stufe nur 1/5 einer Instruktion hat, ist die Taktzeit fast 5mal schneller.

z.B.: oben gezeigt,

bei 1.Takt läuft Fetch von Instr. 1

2. Takt → Decode Instr. 1 + Fetch Instr. 2

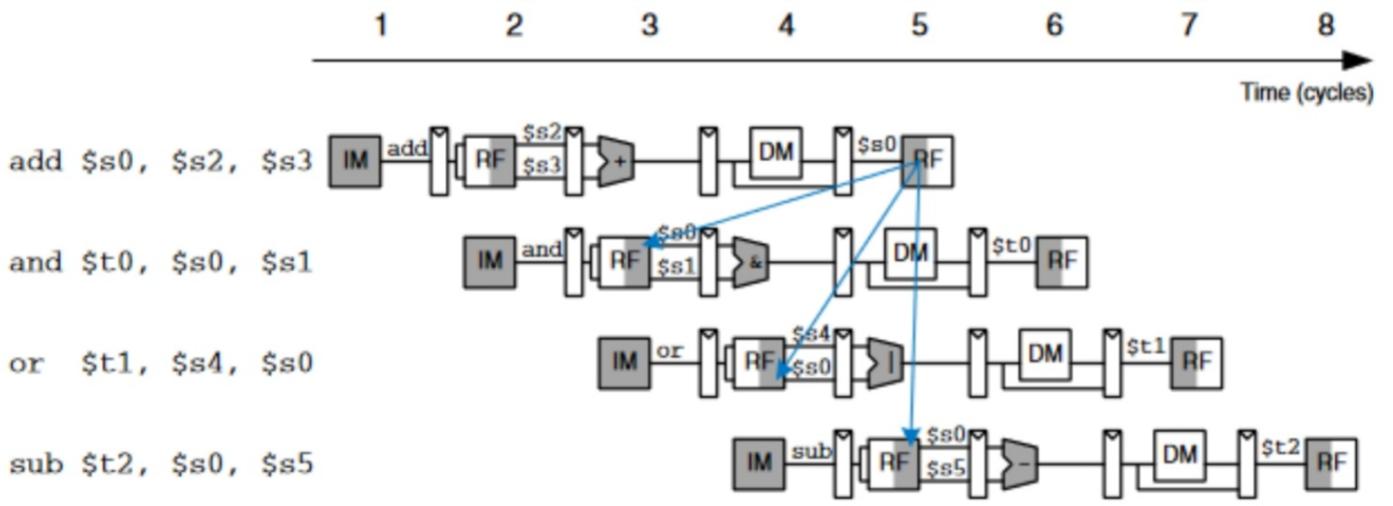
3. Takt → Execute 1 + Decode 2 + Fetch 3 usw...

⇒ Die Taktzeit ist 250ps wegen der langsamsten Teilkonstruktion Fetch. Die Ausgabe ist daher 5mal mehr (250ps/Instr oder 4 Milliarden Instr/s, während bei Singlecycle 950ps/Instr ≈ 1 Milliarden Instr/s), und die Latenzzeit ist (fast) unverändert

⇒ Verbesserte Ausführungs geschwindigkeit

⇒ Effiziente Ressourcennutzung: Im Pipelining-Design werden die Ressourcen des Prozessors effizient genutzt. Während eine Instruktion in einer Pipeline-Stufe verarbeitet wird, können die anderen Stufen der Pipelines für die Bearbeitung anderer Instruktionen verwendet werden. Dadurch werden die Ressourcen des Prozessors optimal ausgelastet und die Gesamteffizienz des Systems verbessert.

2) An dem unten Beispiel wird \$s0 an dem 5. Takt ins Register-File gespeichert, aber bei dem 2. Instruktion braucht es \$s0 schon an dem 3. Takt → diese führt zu Read - After - Write Hazard. Wie in der Abbildung, die and und or Instruktionen haben den falschen Wert von \$s0 gelesen. (da die Werte in \$s0 in den Zeitpunkt der Ausführung von Pipeline-Stufe von Instr. and und or noch nicht ausgerechnet wird)



Aufgabe 3:

- (1) Deep Pipeline: Pipeline-Architektur mit mehrere Stufe (10-20). Sie ist schneller (weniger Logik pro Stufe) aber kostet mehr wegen Hazardhandlung und Registeranzahl.
- (2) Branch Prediction: errate, ob Branch genommen werden sollte \Rightarrow erhöht die Genauigkeit des Vorhersagens und verbessert den Fluss in der Instruktion-Pipeline.
- (3) Superscalar Processor: enthält mehrere Kopien der Datenpfad-Hardware zur gleichzeitigen Ausführung mehrerer Instruktionen \Rightarrow deutlich bessere Leistung als Single- sowie Multicycle-Prozessor, aber ist schwer wegen Datenabhängigkeiten.
- (4) Out-of-Order Execution: schaut viele Instruktionen voraus, um unabhängige Instruktionen so schnell wie möglich zu erteilen oder mit ihrer Ausführung zu beginnen \rightarrow Hazard vermeiden.
- (5) Register Renaming: OoO-Prozessor nutzt diese, um WAR-Hazard zu eliminieren. Der Prozessor nutzt externe Register, welche vom Programmierer nicht genutzt werden kann \rightarrow bessere IPC.
- (6) SIMD: Single Instruction Multiple Data, verarbeitet mehreren Daten mit einer einzigen Instruktion \Rightarrow gleichzeitig Verarbeitung von mehreren arithmetische Instruktionen, besonders bei Grafikbearbeitung.
- (7) Multithreading: Ein Program, welches auf einem Computer läuft, nennt man Prozess. Mehrerer Prozesse können gleichzeitig laufen. Ein Prozess enthält auch mehrere Thread, die auch gleichzeitig laufen \Rightarrow Ein Thread kann Ressourcen nutzen, die im Leerlauf von einem anderen Threads steht \rightarrow höhere Durchsatz.
- (8) Multiprocessor: Ein Multiprozessorsystem besteht aus mehreren Prozessoren und einem Verfahren für die Kommunikation zwischen diesen Prozessoren \Rightarrow noch mehrere Thread gleichzeitig ausführen zu können.