处理器设计

指令流水线的基本原理

主讲人: 邓倩妮

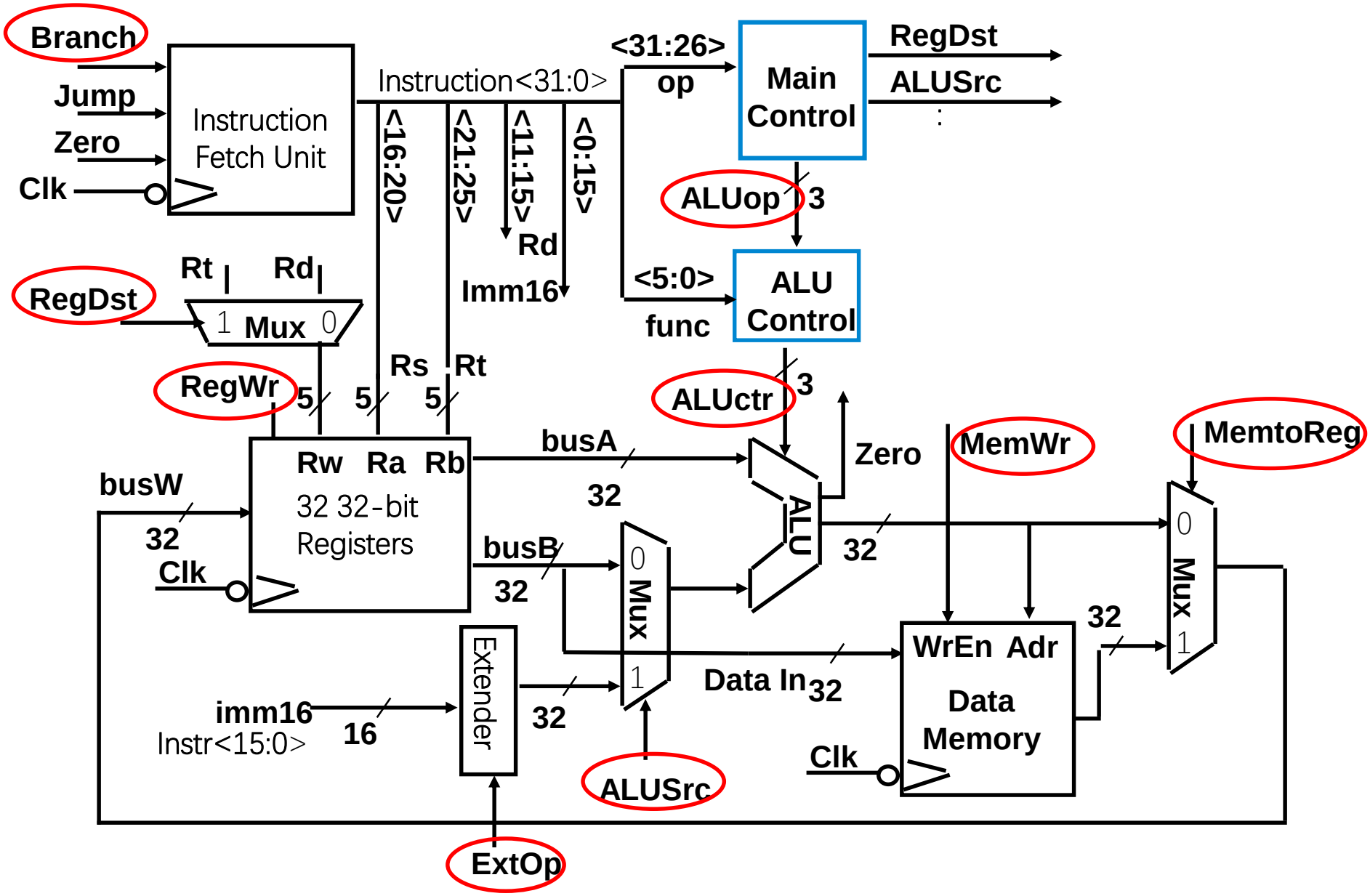
上海交通大学

部分内容来自：

Computer Organization and Design, 4th Edition, Patterson & Hennessy

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

回顾： 单周期处理器


Branch
Jump
Zero
Clk
Instruction Fetch Unit
Instruction<31:0>
<31:26>
op
Main Control
RegDst
ALUSrc
:
<16:20>
<21:25>
<11:15>
Rd
<0:15>
Imm16
ALUop
3
<5:0>
func
ALU Control
Rt
Rd
RegDst
1 Mux 0
RegWr
5
5
5
Rs
Rt
ALUctr
3
busA
32
Rw Ra Rb
32 32-bit Registers
busW
32
Clk
busB
32
0
Mux
1
ALU
Zero
MemWr
MemtoReg
32
0
Mux
1
32
WrEn Adr
Data In
32
Data Memory
Clk
imm16
Instr<15:0>
16
Extender
32
ALUSrc
ExtOp

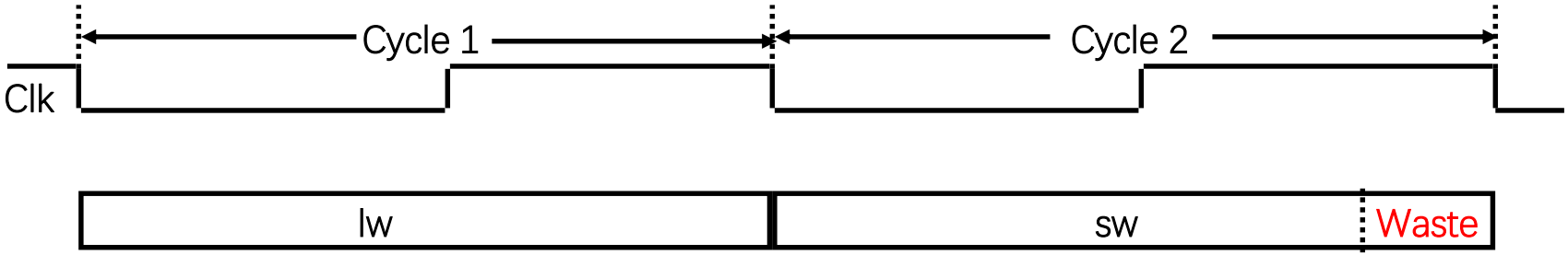
指令的关键路径

- 计算每条指令的延迟，根据：
  - Instruction and Data Memory (200 ps)
  - ALU , adders (100 ps)
  - Register File access (reads or writes) (50 ps)
  - 可忽略的延迟 (for muxes, control unit, sign extend, PC access, shift left 2, wires, setup and hold times)

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type	200	50	100	0	50	400
load	200	50	100	200	50	600
store	200	50	100	200		550
beq	200	50	100	0		350
jump	200					200

单周期： Disadvantages & Advantages

- 时钟周期受限于最慢的一条指令 （slowest instruction）
  - 复杂的指令例如： floating point multiply 会带来问题

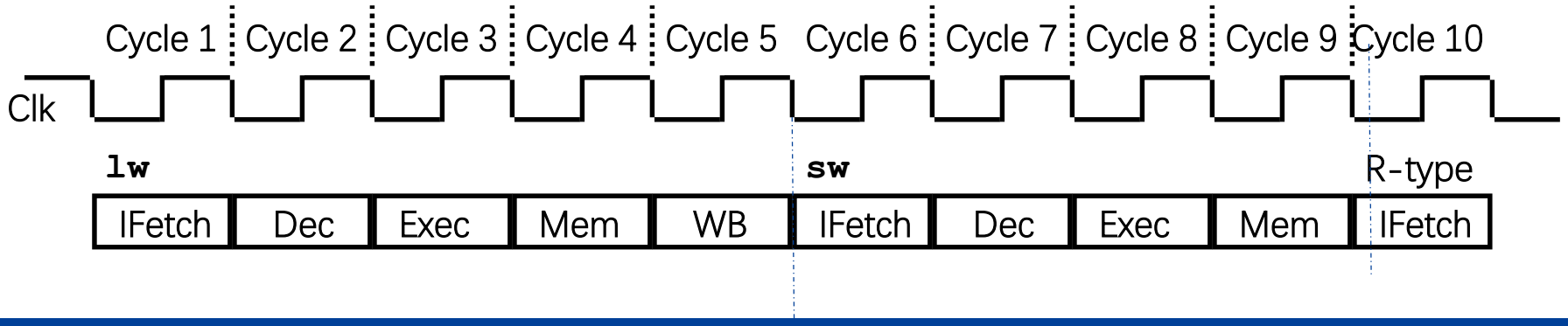


- 缺点： 芯片面积的浪费，某些功能部件 (e.g., adders) 在一条指令周期内只能使用一次，因此同一个部件需要多个。
- 优点： 简单、容易理解

解决方案1：多周期实现一条指令

- 每条指令所花的时钟周期不相同
  - 时钟频率变快
  - 不同指令所需的时钟周期不同
- 一个部件在不同的周期可以重复使用
  - 不需要增加新部件

Multiple Cycle Implementation:

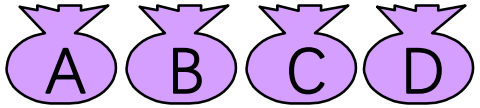


本节内容

- 流水线的原理和特点
- 指令执行的五个阶段
- MIPS五阶段流水线的原理

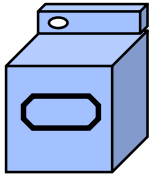
解决方案2：流水线

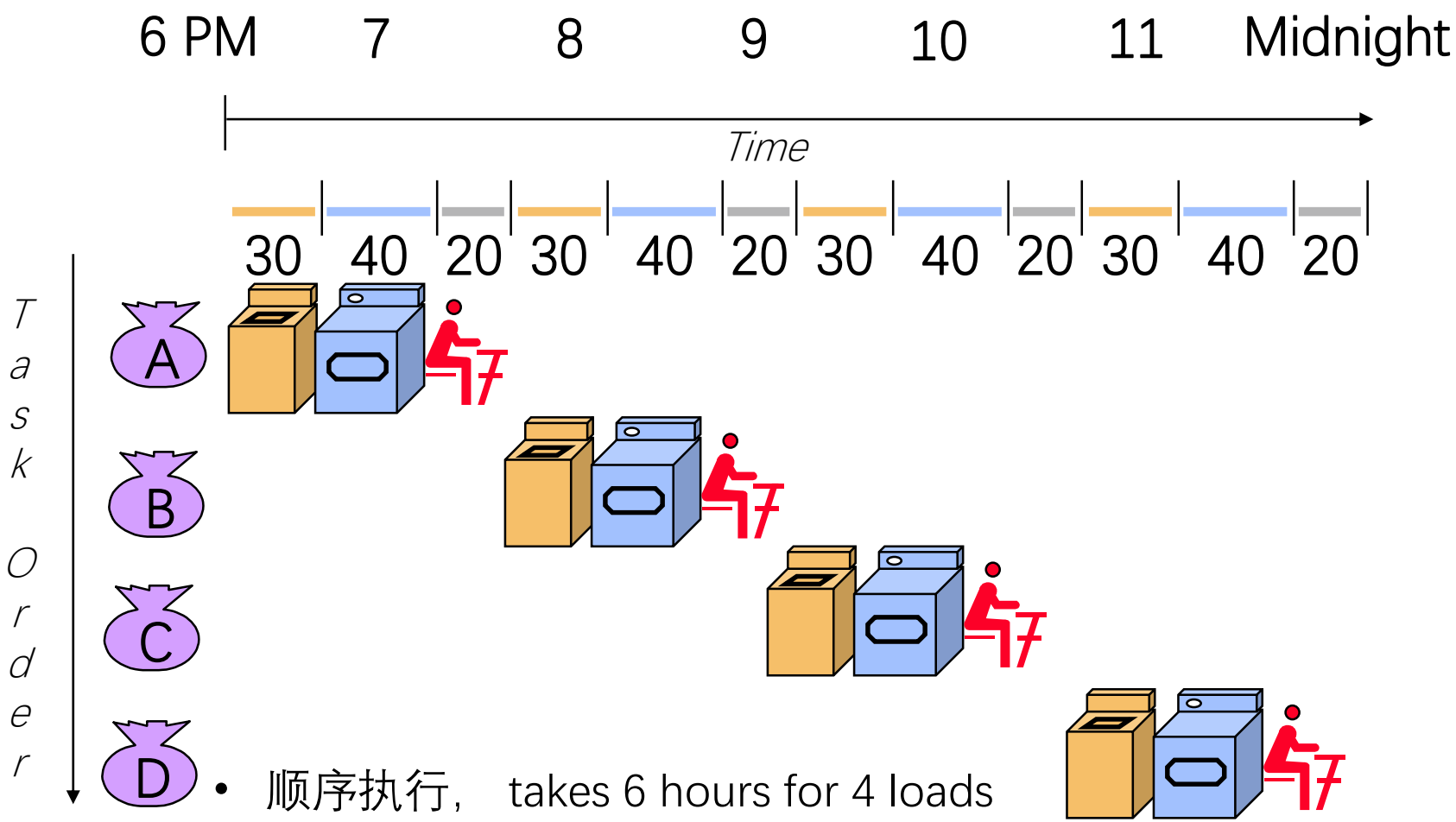
Pipelining is Natural!



洗衣房 Laundry Example:

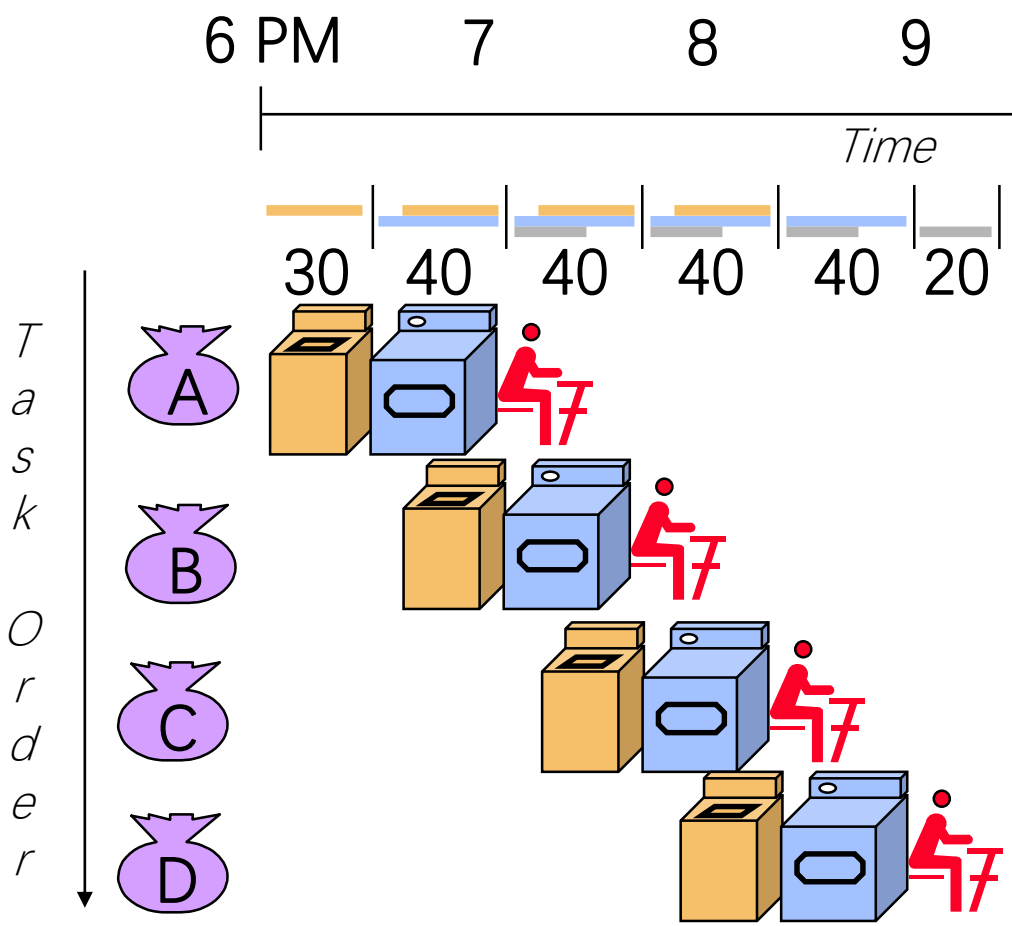
- Washer takes 30 minutes
- Dryer takes 40 minutes
- Folder takes 20 minutes





- 顺序执行， takes 6 hours for 4 loads

Pipelined Laundry: Start work ASAP



- 流水化执行：
  takes 3.5 hours
  for 4 loads

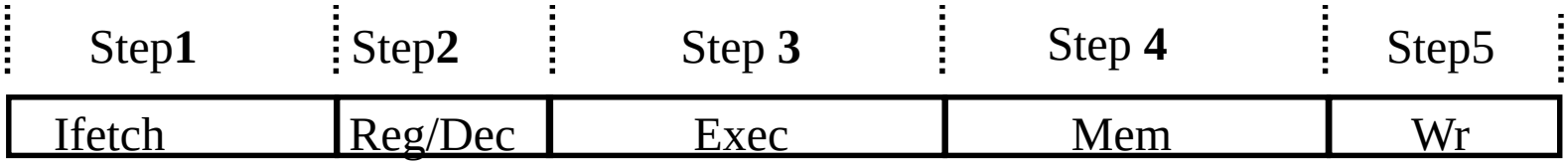
流水线的特点

- 不会改善单个任务的延迟（latency）；它只对整个任务的吞吐量(throughput) 有帮助;
- 流水线的速率受限于最慢的流水段；
- 流水段长度的不均衡也会降低加速比（ speedup ）
- 多个任务使用不同的资源，在不同的流水段同时执行；
- 理想达到的加速比（ speedup ） ：流水段数
- 填充“fill” 和排空“drain”流水线也降低加速比
- 流水线会因为相关性（ Dependences ）而停顿

本节内容

- 流水线的原理和特点
- 指令执行的五个阶段
  - 以LOAD指令为例
- MIPS五阶段流水线的原理

回顾：Load指令的五个阶段　lw rt, rs, imm16



- Ifetch:取指令 fetch instruction, PC+4

  Instruction MEM、Adder（加法器）

- Reg/Dec : 读寄存器、指令译码 Register Reading, Decoding

  Reg（read)、decoder（译码器）

- Exec: 计算存储器地址 Computer MEM Add

  Extender（立即数扩展）、ALU（运算单元）

- Mem ：读数据存储器 read from Mem

  Data Mem

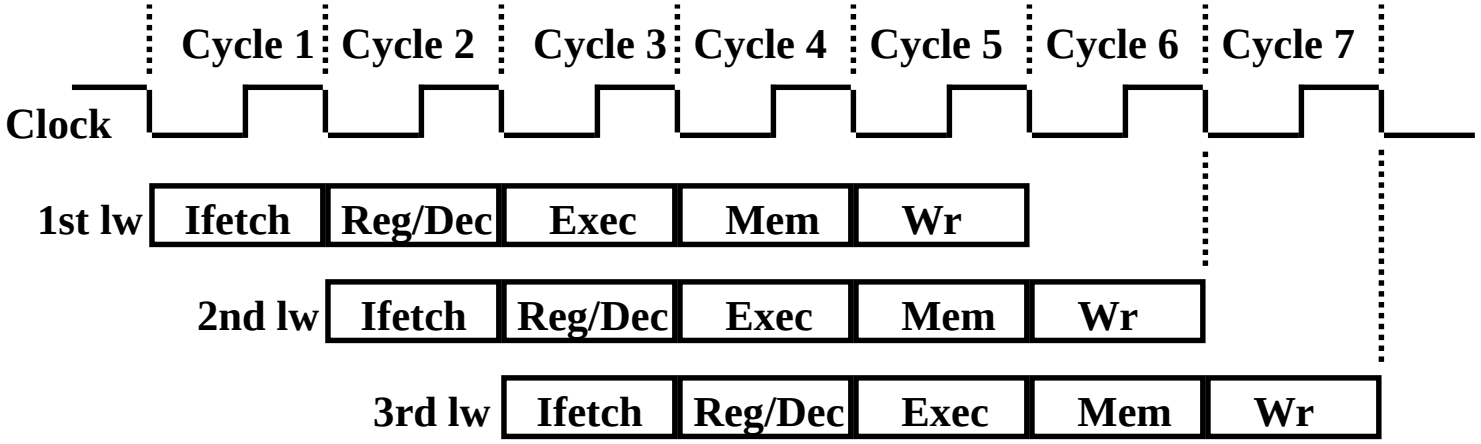
- Wr: 写寄存器 Write to Reg

  Reg (write)

易于流水化：

每一个阶段使用不同的资源（寄存器的读和写一个并行执行，后面再讨论）

将Load指令的执行流水化



- 在数据通路上有五个功能单元:
  - 指令存储器：Instruction Memory ： Ifetch （取指令）阶段
  - 寄存器文件的读端口 (bus A and busB) ：Reg/Dec （读寄存器/译码）阶段
  - 算术逻辑运算单元 ALU： Exec（执行）阶段
  - 数据存储器 Data Memory ： Mem（访存）阶段
  - 寄存器文件的写端口(bus W) ：Wr（写结果） 阶段

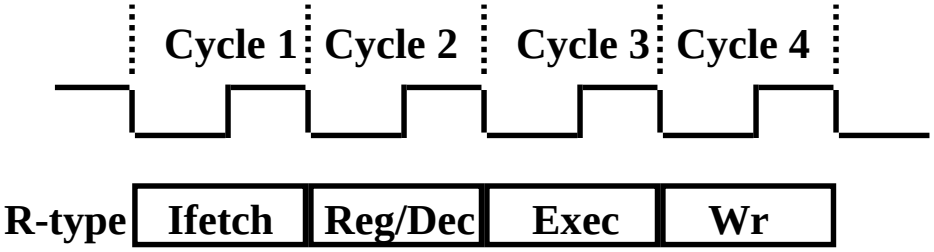
MIPS ISA 流水化

- 容易流水化
  - 1. 所有指令等长 (32 bits)
    - 可以在第一阶段内完成取指， 第二阶段完成译码
  - 2. 指令格式少， （三种）
    - 都可以在第二段读寄存器
  - 3. 只有loads and stores指令能访问存储器
    - 都可以在第三阶段计算存储地址
  - 4. 每条指令最多写一个结果
    - 在最后阶段完成，例如在MEM 或 WB段写结果

本节内容

- 流水线的原理和特点
- 指令执行的五个阶段
- MIPS五阶段流水线的基本工作方式

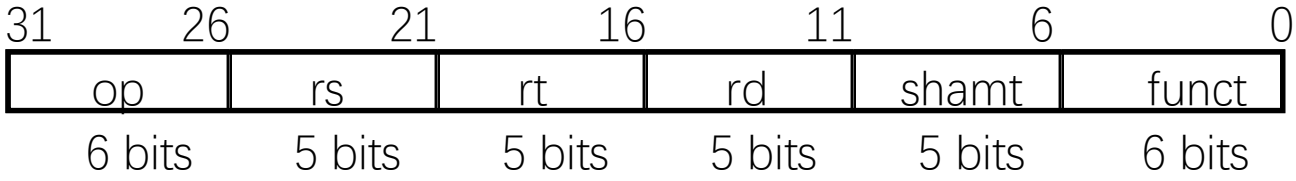
R型指令： 只需要4阶段



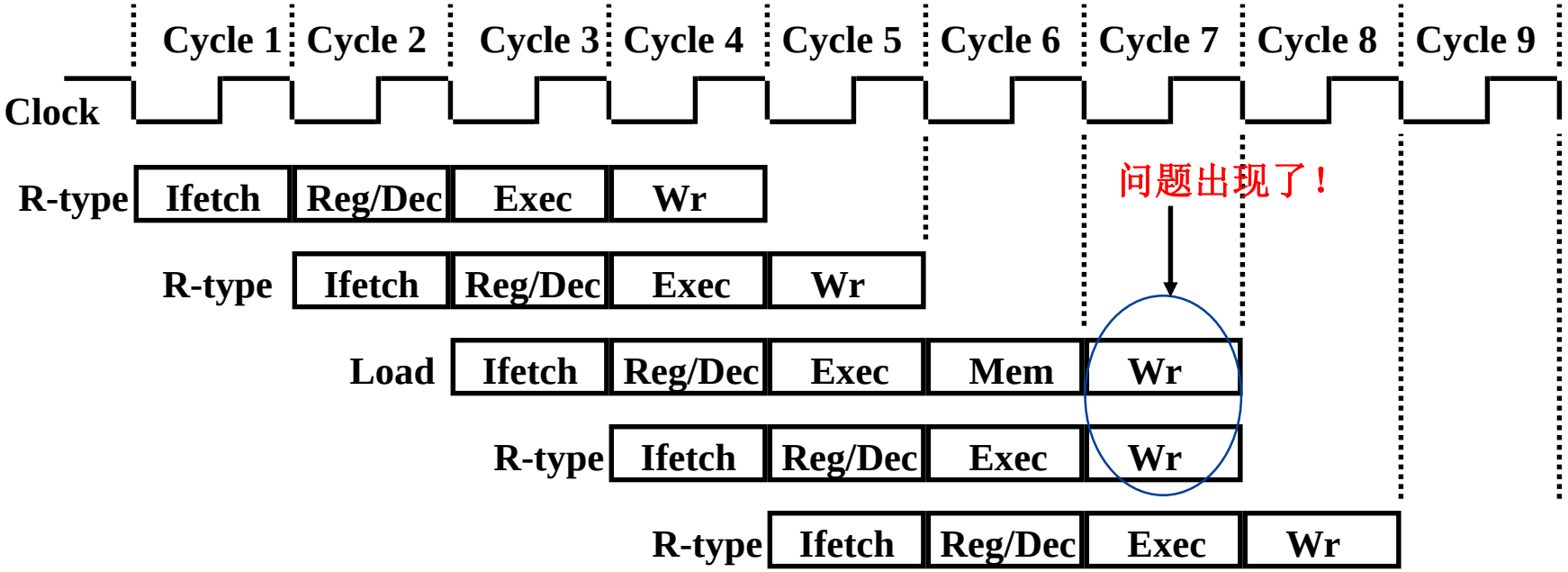
ADD and subtract
add rd, rs, rt
sub rd, rs, rt

- Ifetch: 从指令存储器中取指令
- Reg/Dec: 取寄存器内容 、指令译码
- Exec: ：使用ALU对从两个寄存器中取出的操作数进行运算
- Wr: 将 ALU 结果写入寄存器文件

R型指令格式：



R型指令和 Load指令流水执行



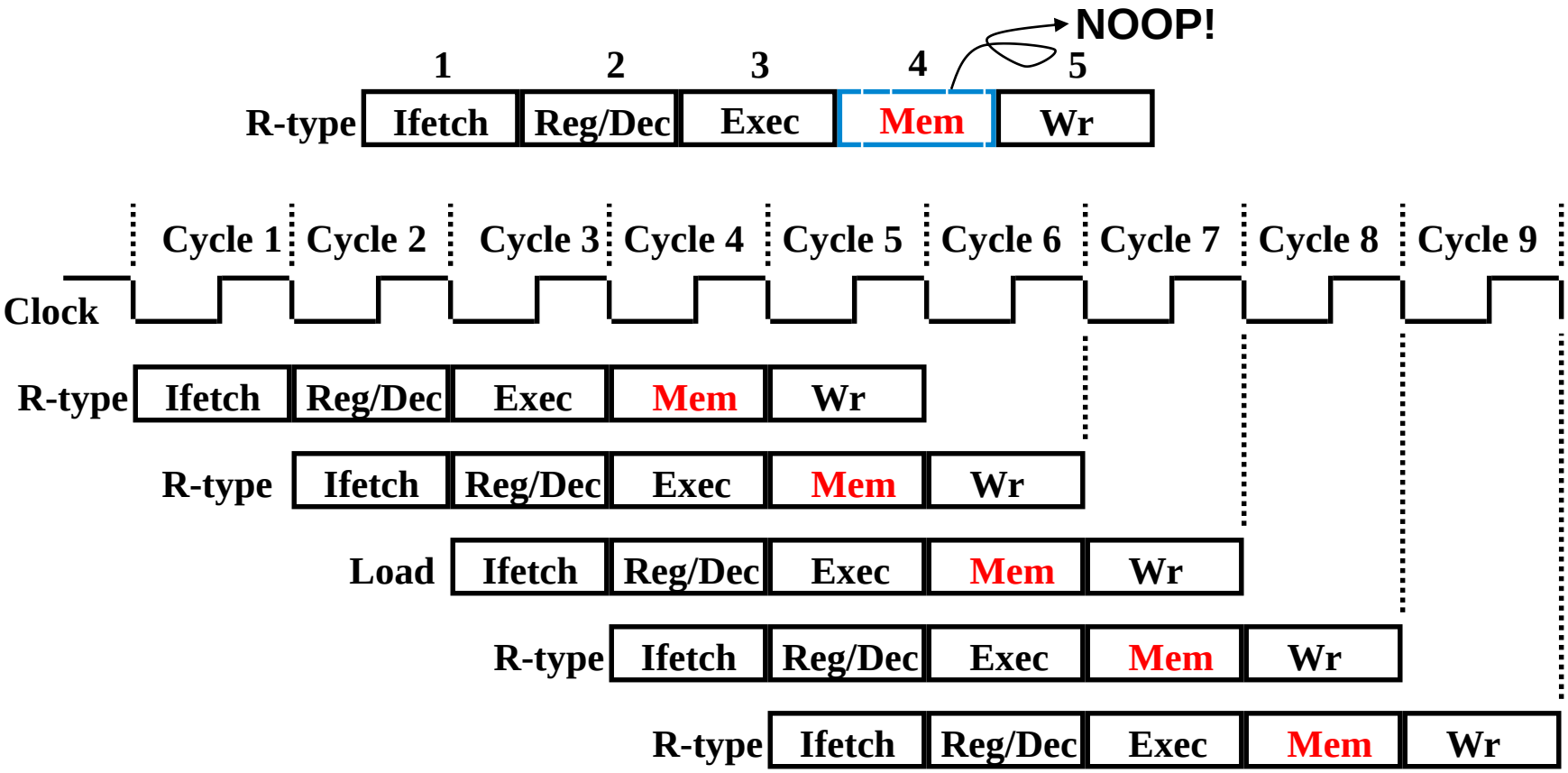
- 出现了资源冲突:

  两条指令都要在同一个时间段使用寄存器写端口

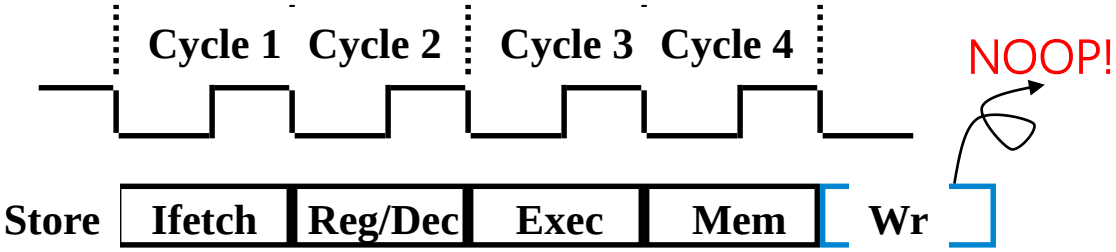
  只有一个寄存器写端口

解决方案：每条指令都有相同的流水段数

- R型指令的 Write 阶段推后一周期:
  - 写寄存器推迟到第五段
  - MEM 段是一个空（NOOP）操作.



Store指令

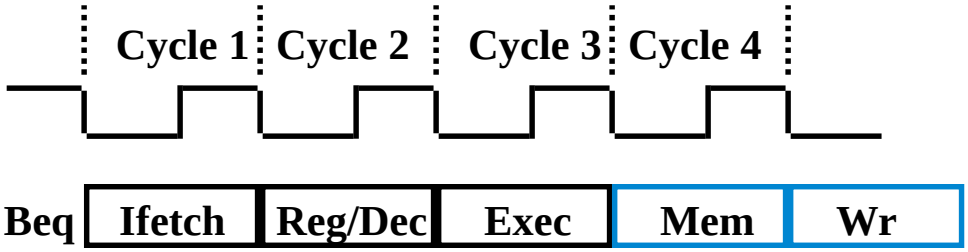


STORE指令：
sw rt, rs, imm16

- Ifetch：取指令
  - Fetch the instruction from the Instruction Memory
- Reg/Dec:：取寄存器内容 、指令译码
- Exec:：计算存储器地址
- Mem：将数据写入数据存储器

加一个空阶段，让所有指令的执行段数保持一致

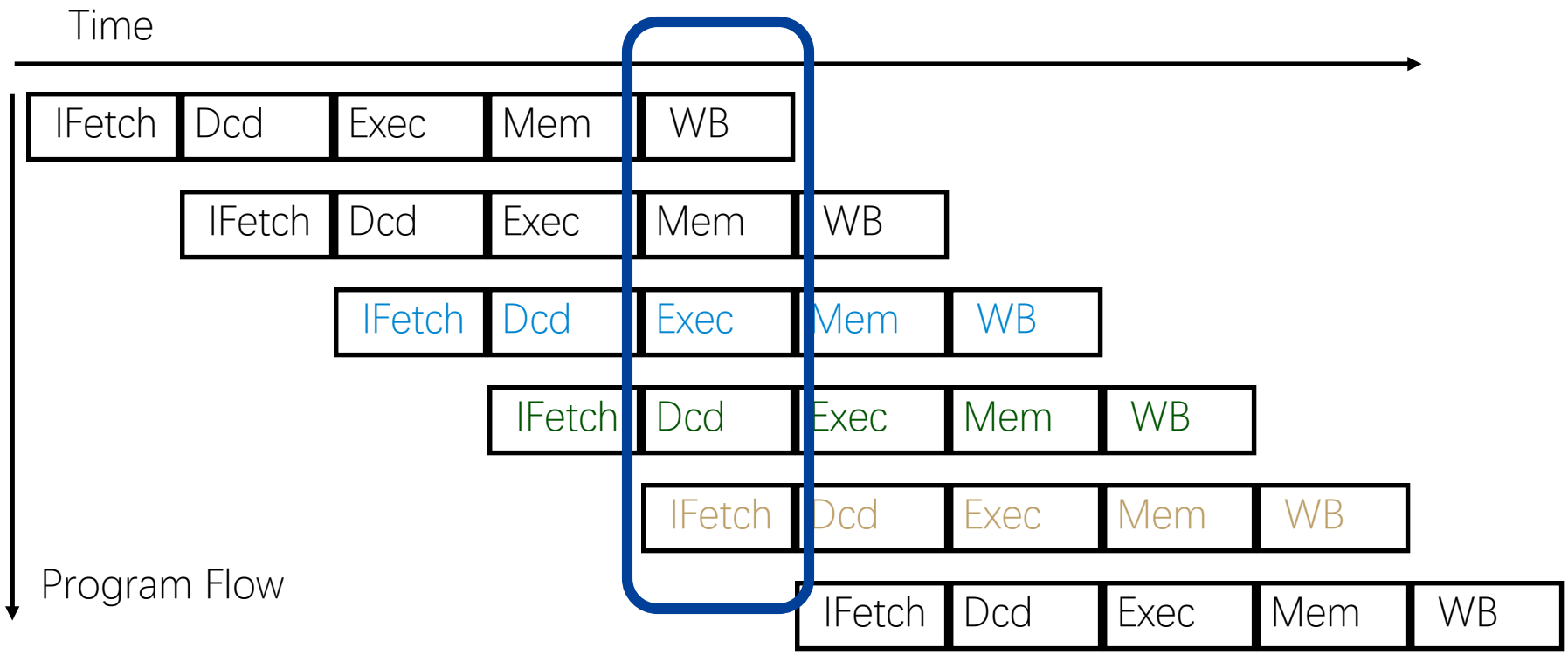
Beq指令



BRANCH
beq rs, rt, imm16

- Ifetch:取指令
- Reg/Dec:取寄存器内容 、指令译码
- Exec:
  - 比较两个操作数,
  - 计算转移目标地址
- Mem:
  - 将 计算好的目标地址送给 PC 输入端
- Wr : 空操作

理想的指令流水线



理想情况下，每一个周期
完成一条老的指令；
开始一条新的指令。达到CPI=1；
CPI：完成一条指令所花的周期数

小结

- 流水线能提高单位时间的任务吞吐量
- MIPS 指令集很合适流水化执行
- 五阶段指令流水线，理想情况CPI=1

谢谢！