



南京大學

NANJING UNIVERSITY

# 指令流水线

殷亚凤

智能软件与工程学院

苏州校区南雍楼东区225

yafeng@nju.edu.cn , <https://yafengnju.github.io/>



# 指令流水线

- 流水线概述
- 流水线处理器的实现
- 流水线冒险及其处理
- 高级流水线技术





# 流水线概述

- 如果将指令的每个阶段看成相应的流水段，则指令的执行过程构成了一条指令流水线。
- 一条指令流水线由如下5个流水段组成：
  - **取指令(IF)**：从存储器取指令；
  - **指令译码(ID)**：产生指令执行所需的控制信号；
  - **取操作数(OF)**：读取操作数；
  - **执行(EX)**：对操作数完成指定操作；
  - **写回(WB)**：将结果写回。





# 流水线概述

- 当后一条指令的第 $i$ 步与前一条指令的第 $i+1$ 步同时进行，可以使**一串指令总的完成时间大为缩短**。

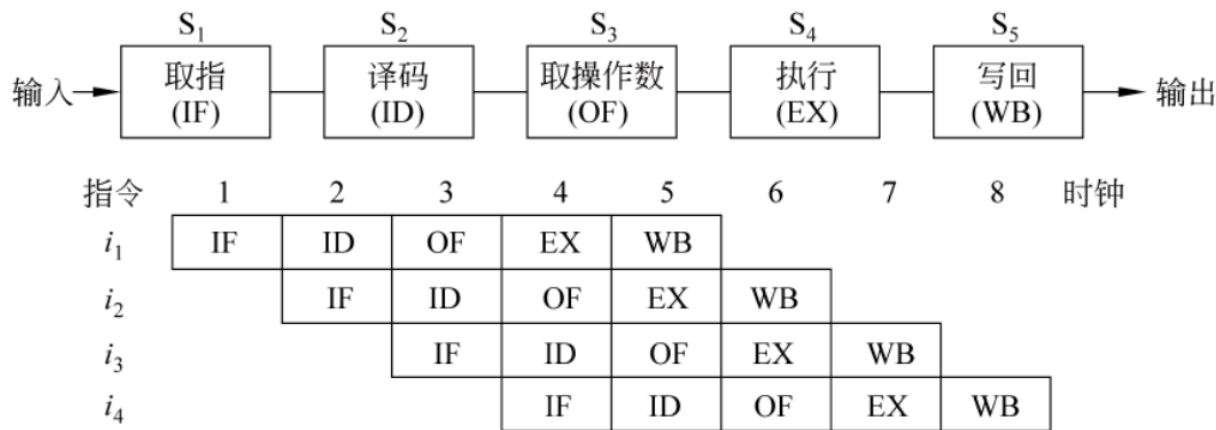


图 6.1 一个 5 段指令流水线

- 理想情况下，每个时钟都有一条指令进入流水线，每个时钟周期都有一条指令完成，每条指令的时钟周期数（即CPI）都为1。



## 回顾：Load指令的5个阶段

阶段1	阶段2	阶段 3	阶段 4	阶段5
Ifetch	Reg/Dec	Exec	Mem	Wr

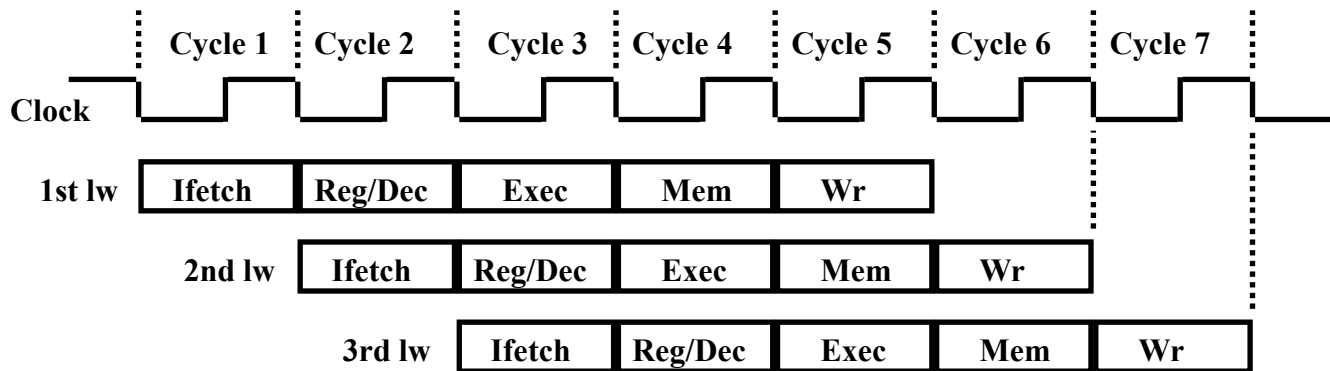
- **Ifetch (取指)**：取指令并计算PC+4 (用到哪些部件？) **指令存储器、Adder**
- **Reg/Dec (取数和译码)**：取数同时译码 (用到哪些部件？) **寄存器堆读口、指令译码器**
- **Exec (执行)**：计算内存单元地址 (用到哪些部件？) **扩展器、ALU**
- **Mem (读存储器)**：从数据存储器中读 (用到哪些部件？) **数据存储器**
- **Wr(写寄存器)**：将数据写到寄存器中 (用到哪些部件？) **寄存器堆写口**

这里寄存器堆的读口和写口可看成两个不同的部件。





# Load指令的流水线



- 每个周期有五个功能部件同时在工作
- 后面指令在前面完成取指后马上开始
- 每个load指令仍然需要五个周期完成
- 但是，吞吐率(throughput)提高许多，理想情况下，有：
  - 每个周期有一条指令进入流水线
  - 每个周期都有一条指令完成
  - 每条指令的有效周期(CPI)为1





# 单周期指令模型与流水模型的性能比较

- 假定以下每步操作所花时间为：

- 取指：2ns
- 寄存器读：1ns
- ALU操作：2ns
- 存储器读：2ns
- 寄存器写：1ns

Load指令执行时间总计为：8ns

(假定控制单元、PC访问、信号传递等没有延迟)

- 单周期模型

- 每条指令在一个时钟周期内完成
- 时钟周期等于最长的lw指令的执行时间，即：8ns
- 串行执行时，N条指令的执行时间为：8Nns

- 流水线性能

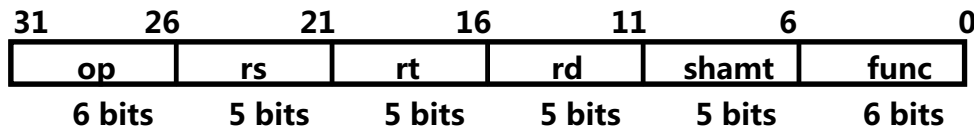
- 时钟周期等于最长阶段所花时间为：2ns
- 每条指令的执行时间为： $2\text{ns} \times 5 = 10\text{ns}$
- N条指令的执行时间为： $(4 + N) \times 2\text{ns}$
- **在N很大时，比串行方式提高约 4 倍**
- 若各阶段操作均衡(例如，各阶段都是2ns)，则提高倍数为5倍。

**流水线方式下，单条指令执行时间不能缩短，但能大大提高指令吞吐率！**



# 适合流水线的指令集特征

- 具有什么特征的指令集有利于流水线执行呢？
  - **长度尽量一致**，有利于简化取指令和指令译码操作
    - MIPS指令32位，下址计算方便:  $PC+4$
    - X86指令从1字节到17字节不等，使取指部件极其复杂
  - **格式少，且源寄存器位置相同**，有利于在指令未知时就可取操作数
    - MIPS指令的rs和rt位置一定，在指令译码时就可读rs和rt的值



若位置随指令不同而不同，则需先确定指令类型才能取寄存器编号

- **load / Store指令才能访问存储器**，有利于减少操作步骤，规整流水线
  - lw/sw指令的地址计算和运算指令的执行步骤规整在同一个周期
  - X86运算类指令操作数可为内存数据，需计算地址、访存、执行
- **内存中“对齐”存放**，有利于减少访存次数和流水线的规整

**总之，规整、简单和一致等特性有利于指令的流水线执行**







# 指令流水线

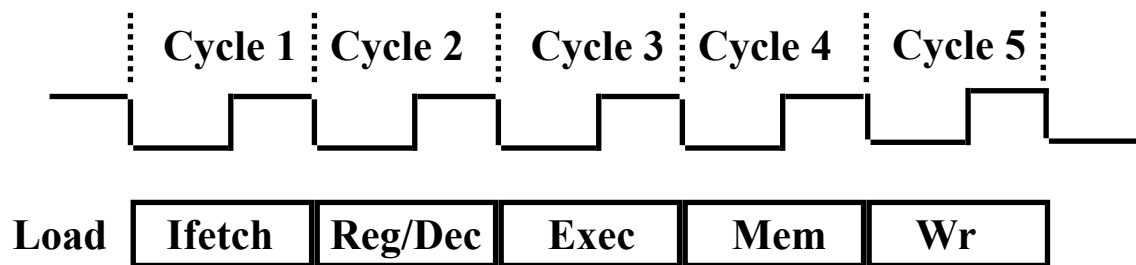
---

- 流水线概述
- **流水线处理器的实现**
- 流水线冒险及其处理
- 高级流水线技术





# Load指令的流水线

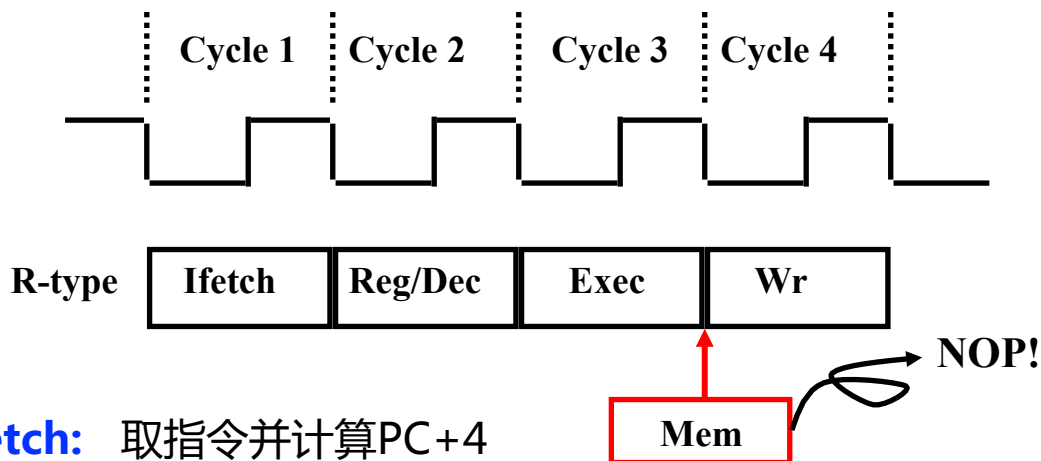


- **Ifetch** : 取指令并计算PC+4
- **Reg/Dec** : 从寄存器取数，同时指令在译码器进行译码
- **Exec** : 16位立即数符号扩展后与寄存器值相加，计算主存地址
- **Mem** : 从存储器中读数据；
- **Wr** : 将数据写入寄存器。





# R型指令功能段划分

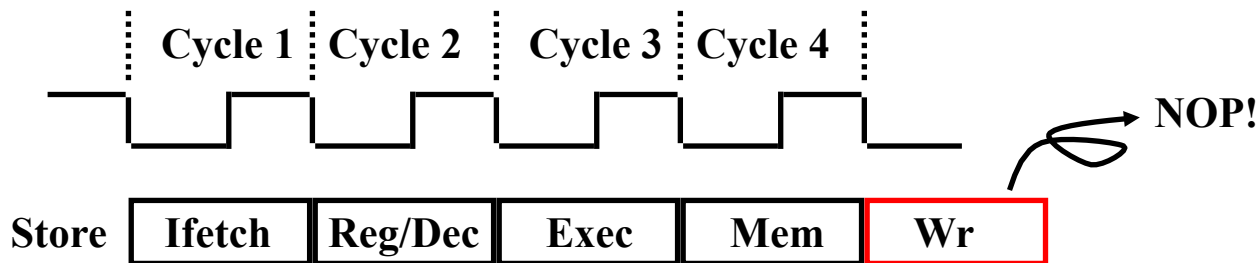


- **Ifetch:** 取指令并计算PC+4
- **Reg/Dec:** 从寄存器取数，同时指令在译码器进行译码
- **Exec:** 在ALU中对操作数进行计算
- **Wr:** ALU计算的结果写到寄存器
- **Mem:** 在Write前加一个空的Mem段，使流水线更规整！

**I型指令功能段划分，与R型指令相同。**



# Store指令功能段划分

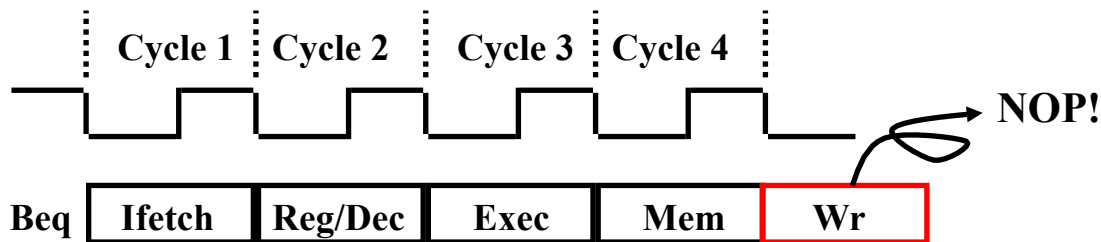


- **Ifetch** : 取指令并计算PC+4
- **Reg/Dec** : 从寄存器取数，同时指令在译码器进行译码
- **Exec** : 16位立即数符号扩展后与寄存器值相加，计算主存地址
- **Mem** : 将寄存器读出的数据写到主存
- **Wr** : 加一个空的写阶段，使流水线更规整！





# Beq指令功能段划分



- **Ifetch:** 取指令并计算PC+4
- **Reg/Dec:** 从寄存器取数，同时指令在译码器进行译码
- **Exec:** 执行阶段
  - ALU中比较两个寄存器的大小（做减法）
  - Adder中计算转移地址
- **Mem:** 如果比较相等, 则转移目标地址写到PC
- **Wr: 加一个空写阶段，使流水线更规整！**

## 与多周期通路有什么不同？

多周期通路中，在Reg/Dec阶段投机进行了转移地址的计算！可以减少Branch指令的时钟数

## 为什么流水线中不进行“投机”计算？

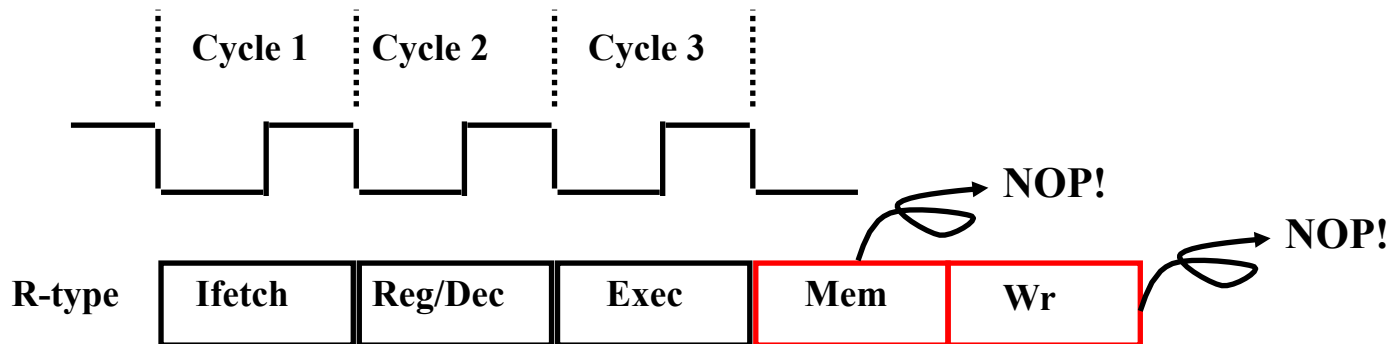
因为流水线中所有指令的执行阶段一样多，Branch指令无需节省时钟，因为有比它更复杂的指令。

按照上述方式，把所有指令都按照最复杂的“load”指令所需的五个阶段来划分，不需要的阶段加一个“NOP”操作





# J指令功能段划分



- **Ifetch** : 取指令并计算PC+4
- **Reg/Dec** : 从寄存器取数，同时指令在译码器进行译码
- **Exec** : 将目标地址送PC
- **Mem** : 加一个空的Mem阶段，使流水线更规整！
- **Wr** : 加一个空的写阶段，使流水线更规整！

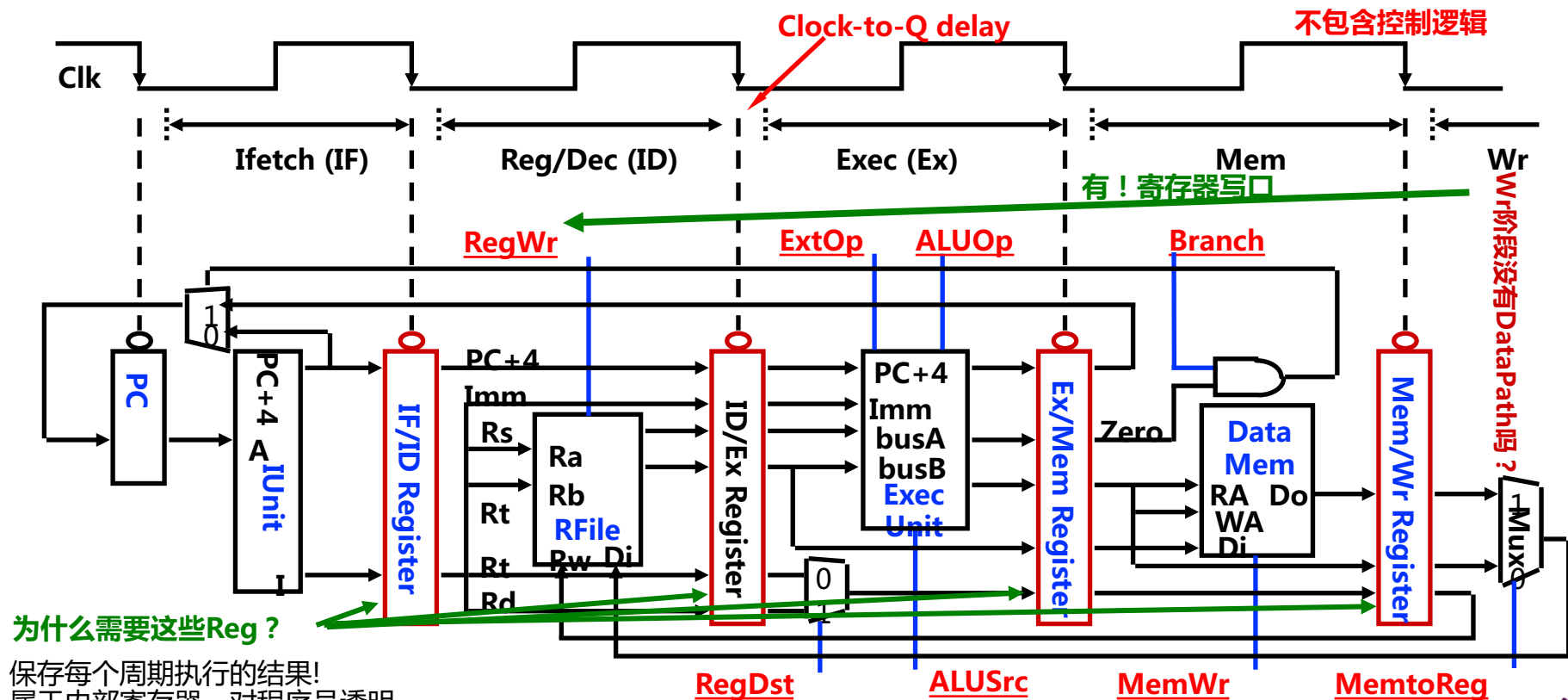
## 插入“空”段时：

1. 每个功能部件每条指令只能用一次（如寄存器写口不能用两次或以上）；
2. 每个功能部件必须在相同的阶段被使用（如寄存器写口总是在第5阶段被使用）。





# 5段流水线数据通路



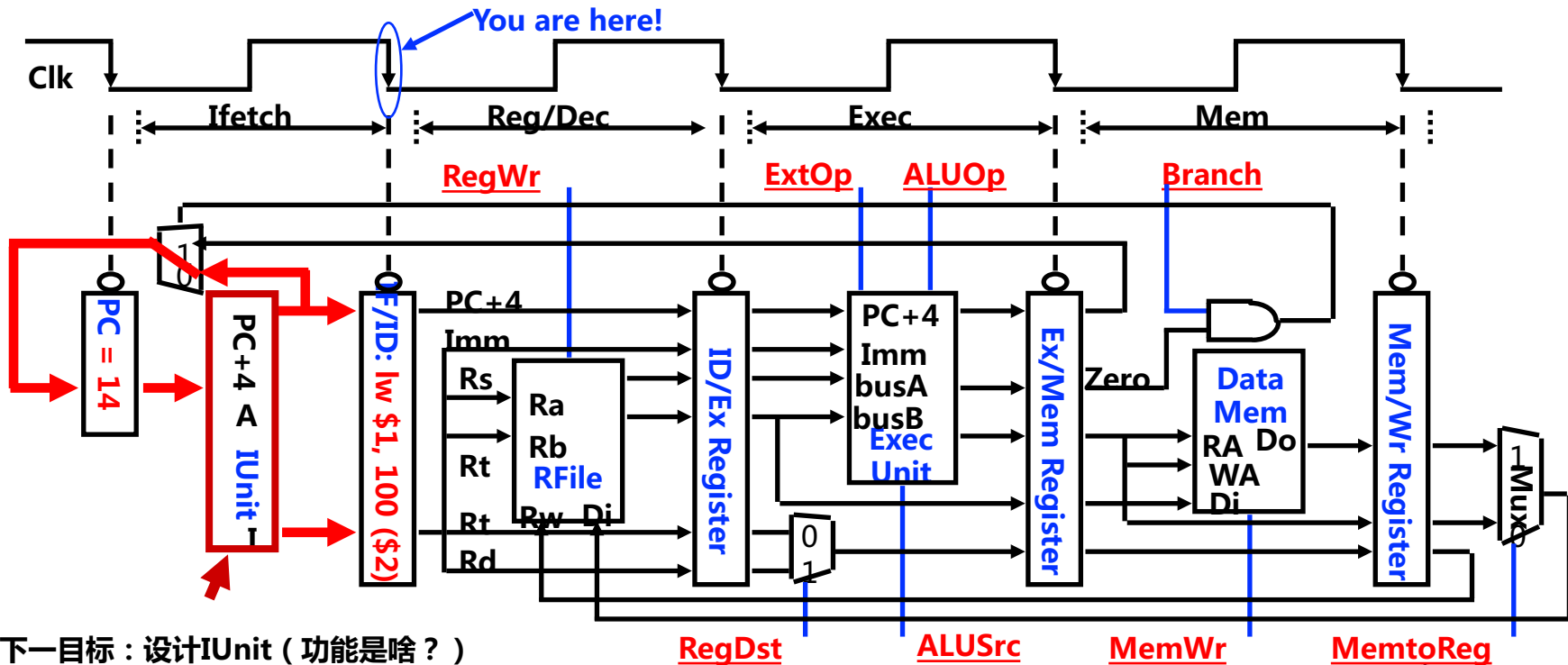
保存每个周期执行的结果!  
属于内部寄存器, 对程序员透明,  
不需作为现场保存。

下面看每条指令在流水线通路中的执行过程



# 取指令(Ifetch)阶段

- 第12单元指令: `lw $1, 0x100($2)`      功能:  $\$1 \leftarrow \text{Mem} [(\$2) + 0x100]$



下一目标：设计IUnit ( 功能是啥？ )

MIPS指令的地址可能是12吗？可以！

先猜一下IUnit中有哪些功能部件？





# 指令部件IUnit的设计

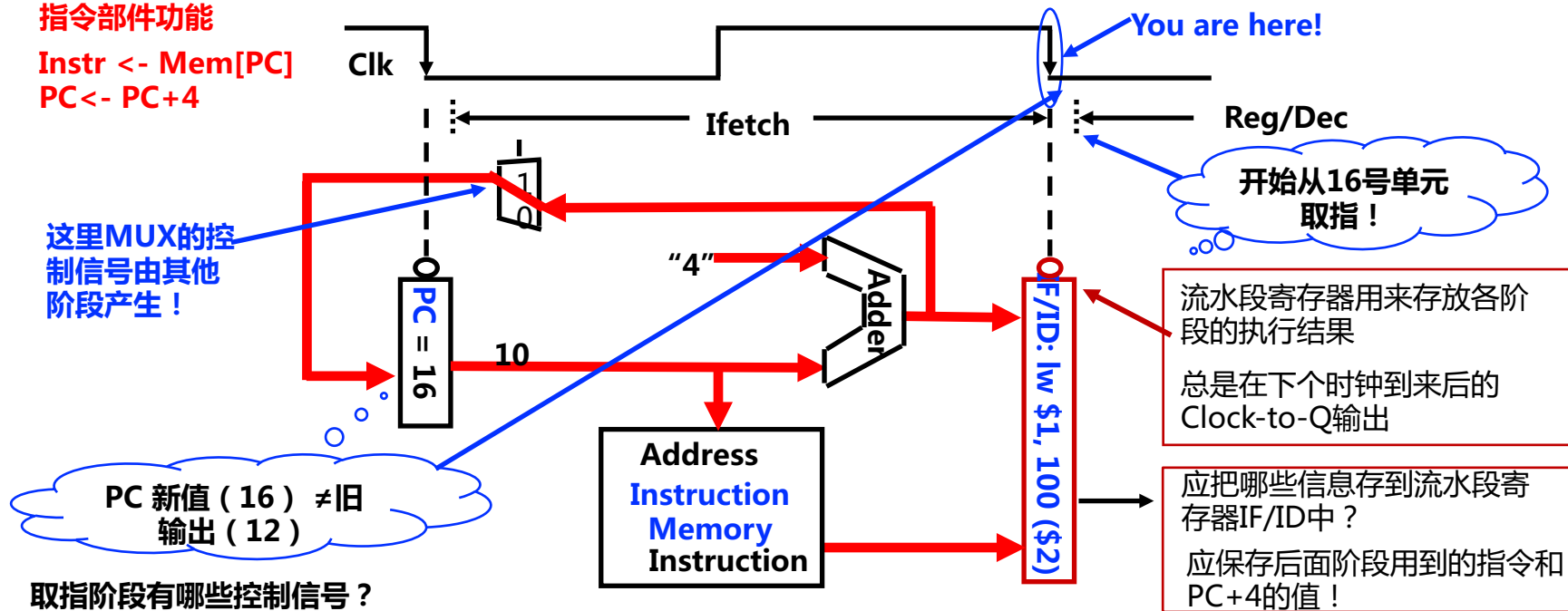
- 第12单元指令: `lw $1, 0x100($2)`

随后的指令在16号单元中！

指令部件功能

$\text{Instr} \leftarrow \text{Mem}[\text{PC}]$

$\text{PC} \leftarrow \text{PC} + 4$



取指阶段有哪些控制信号？

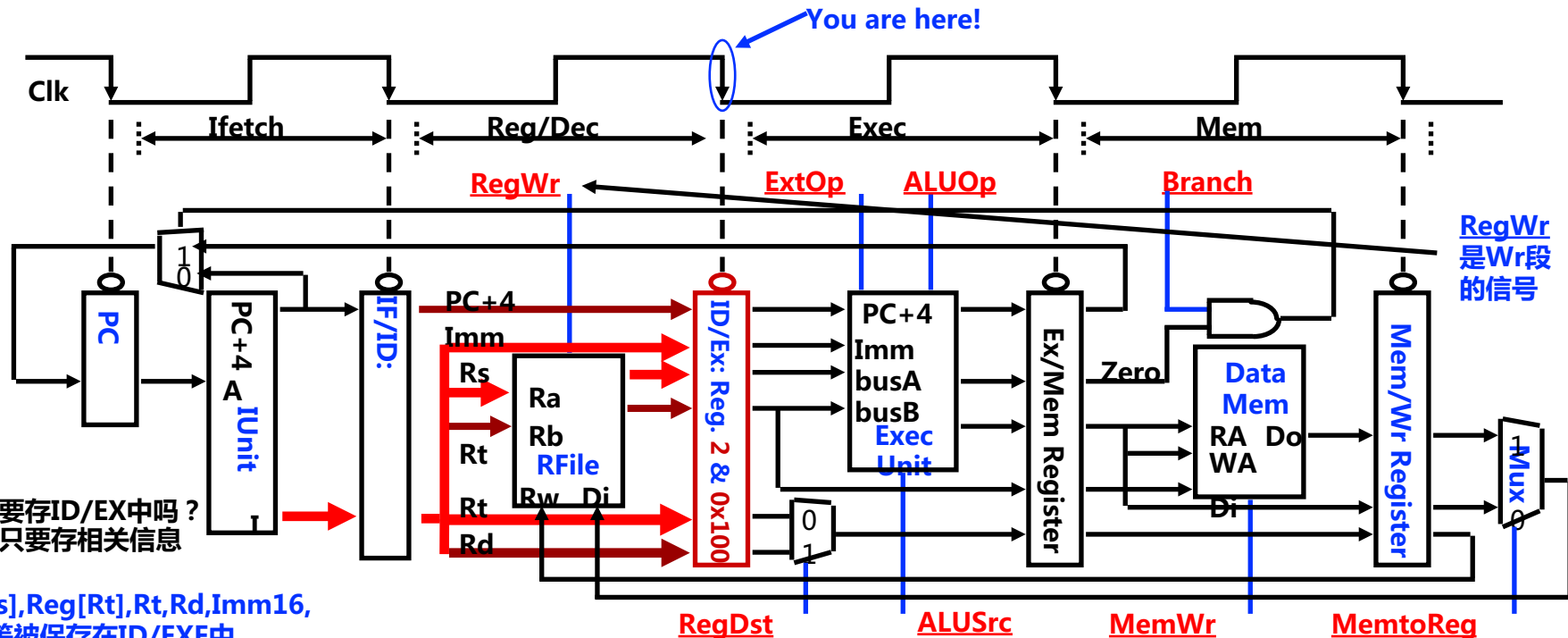
不需控制信号，因为每条指令执行功能一样，是确定的，无需根据指令的不同来控制执行不同的操作！

指令在随后阶段被送出译码！  
PC+4用来计算转移目标地址



# 译码/取数(Reg/Dec)阶段

- Location 12: lw \$1, 0x100(\$2)      功能 :  $\$1 \leftarrow \text{Mem}[(\$2) + 0x100]$



RegWr  
是Wr段  
的信号

指令还要存ID/EX中吗？  
不要，只要存相关信息

Reg[Rs], Reg[Rt], Rt, Rd, Imm16,  
PC+4等被保存在ID/EXE中

该阶段有哪些控制信号？ 没有！因是所有指令的公共操作，故无控制信号！

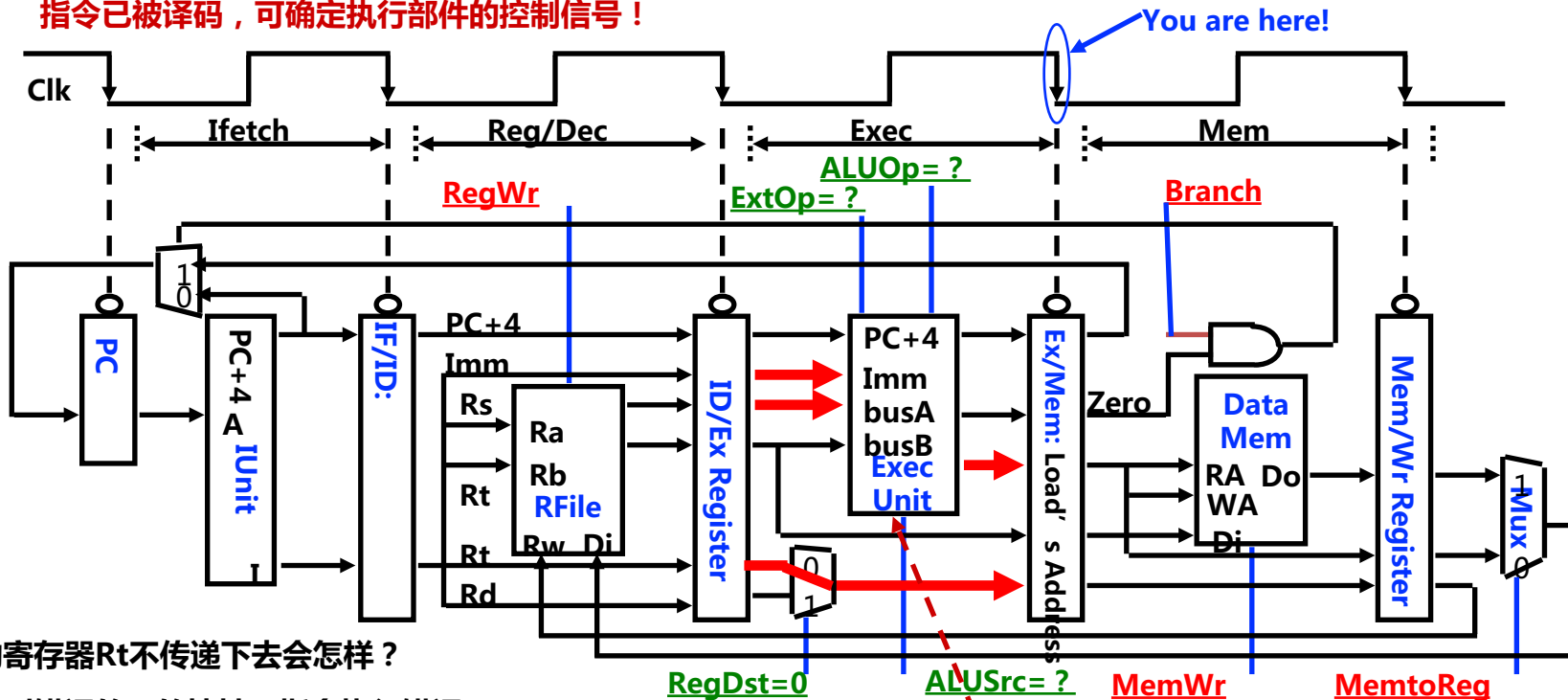




# 执行(Exec)阶段：Load指令的地址计算阶段

- Location 12: lw \$1, 0x100(\$2)      功能：\$1 <- Mem[(\$2) + 0x100]

指令已被译码，可确定执行部件的控制信号！



目的寄存器Rt不传递下去会怎样？

连接到错误的目的地址，指令执行错误！

下一目标：设计执行部件(Exec Unit) 猜有哪些部件？





# 执行部件(Exec Unit)的设计

## 执行部件功能？

- 计算内存地址
- 计算转移目标地址
- 一般ALU运算

## Load指令的各控制信号取值？

RegDes=0, ALUSrc=1  
ALUOp=addu, Extop=1

## Store指令呢？

RegDes=x, ALUSrc=1  
ALUOp=addu, Extop=1

## Branch指令呢？

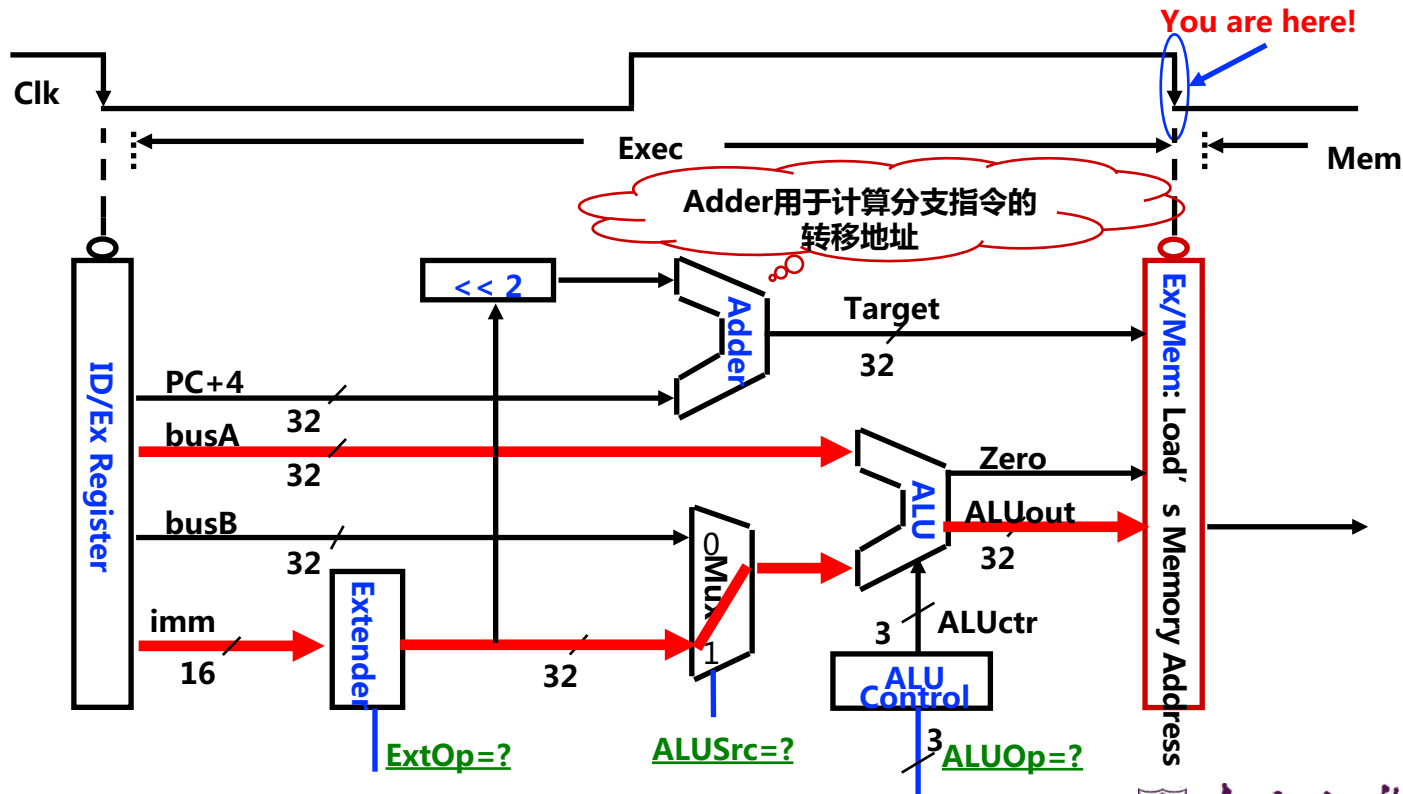
RegDes=x, ALUSrc=0  
ALUOp=subu, Extop=1

## Ori指令呢？

RegDes=0, ALUSrc=1  
ALUOp=or, Extop=0

## R型指令呢？

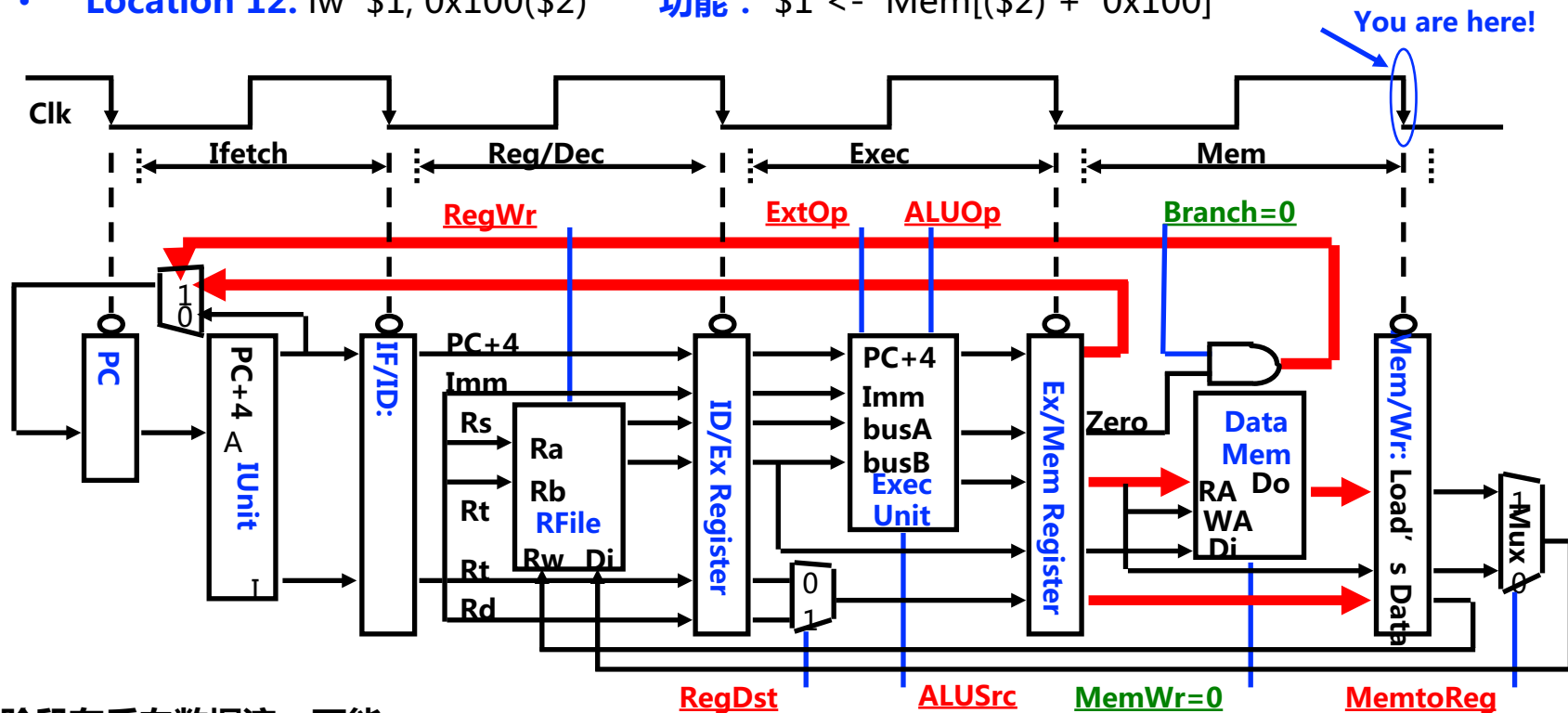
RegDes=1, ALUSrc=0  
ALUOp='func', Extop=x





# Mem阶段：Load指令的存储器读(Mem)周期

- Location 12: `lw $1, 0x100($2)`      功能:  $\$1 \leftarrow \text{Mem}[(\$2) + 0x100]$



该阶段有反向数据流，可能会引起冒险！以后介绍。

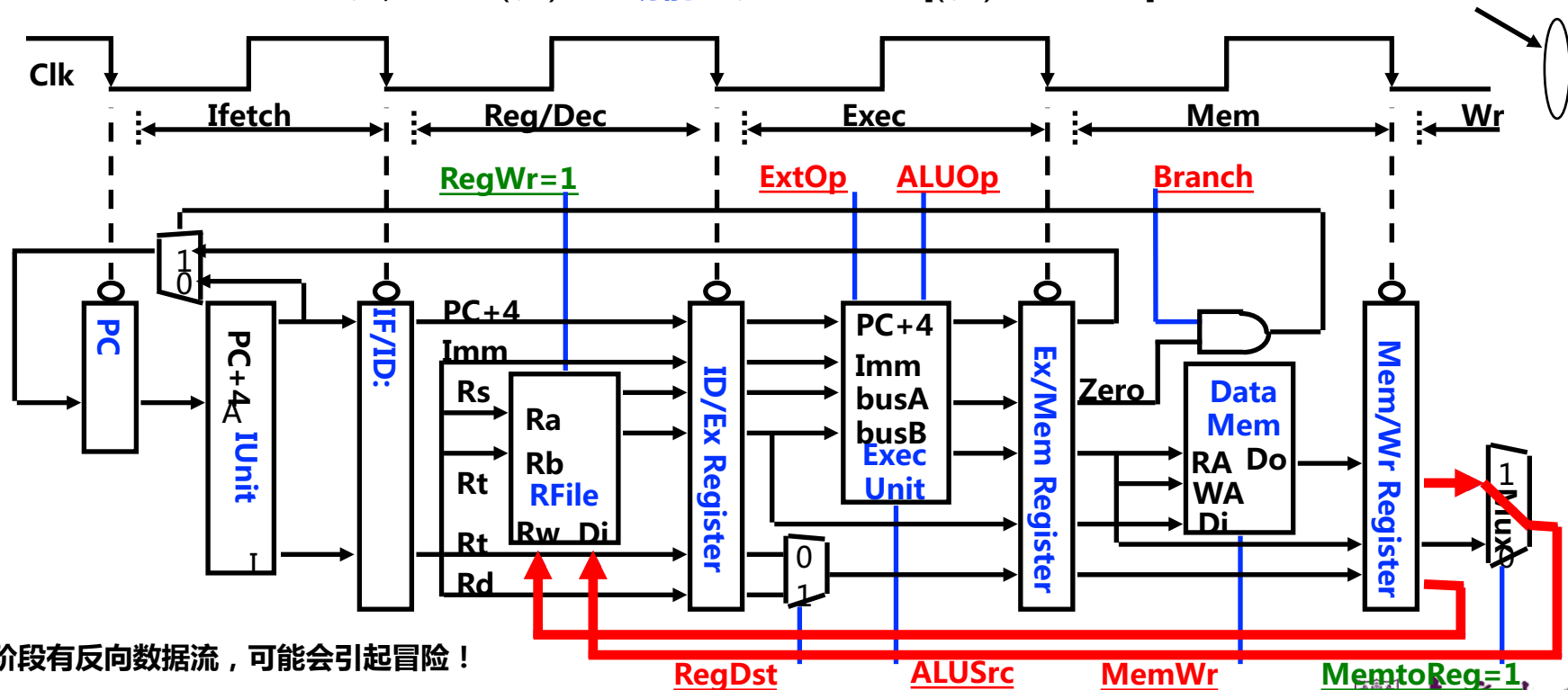
周期以最长操作为准设计  $\text{Cycle} > T_{\text{read}}$





# Wr段：Load指令的回写(Write Back)阶段

- Location 12: lw \$1, 0x100(\$2)      功能：\$1 <- Mem[(\$2) + 0x100]



该阶段有反向数据流，可能会引起冒险！

各阶段所经DataPath已有，控制信号如何得到？

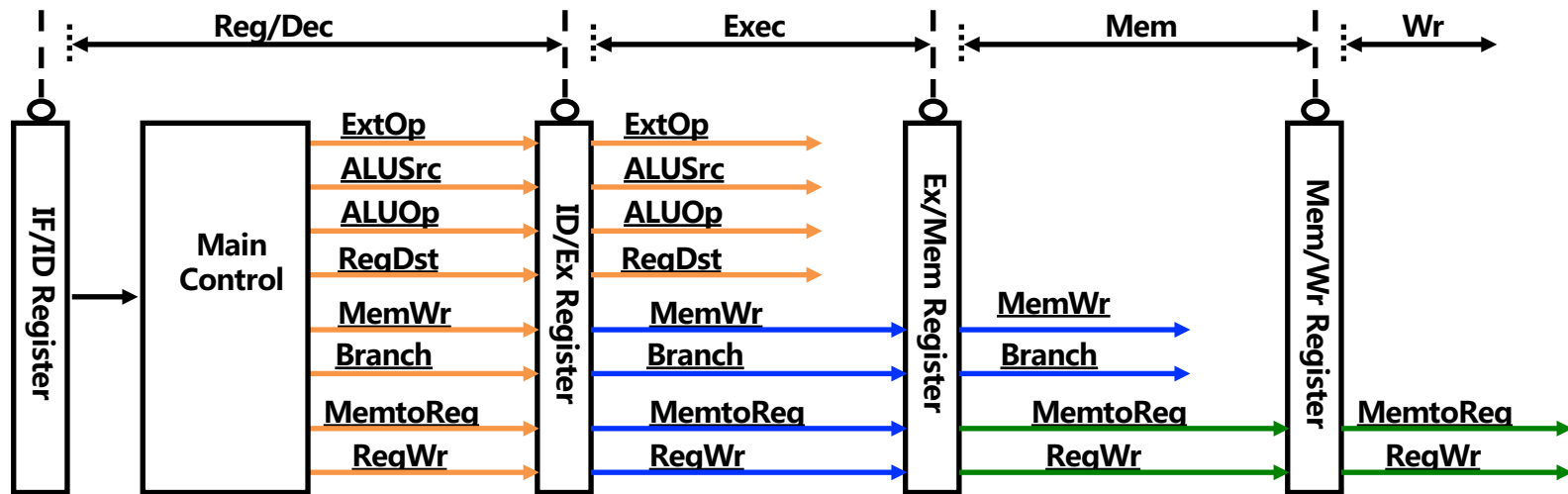


# 流水线中的控制信号

- 在**取数/译码 (Reg/Dec)** 阶段产生本指令每个阶段的所有控制信号 **所以，控制信号也要保存在流水段寄存器中！**

- **Exec信号 (ExtOp, ALUSrc, ...)** 在1个周期后使用
- **Mem信号 (MemWr, Branch)** 在2个周期后使用
- **Wr信号 (MemtoReg, RegWr)** 在3个周期后使用

为什么 第1、2阶段没有控制信号？  
IF和ID阶段的功能对每条指令来说都一样



各流水段部件在一个时钟内完成**某条指令的某个阶段**的工作！

在下个时钟到达时，把**执行结果以及前面传递来的后面各阶段要用到的所有数据**（如：指令、立即数、目的寄存器等）**和控制信号**保存到流水线寄存器中！



# 流水线中的控制信号

- 通过对前面流水线数据通路的分析，得知：
  - PC需要写使能吗？ 每个时钟都会改变PC，故不需要!
  - 流水段寄存器需要写使能吗？ 每个时钟都会改变流水段寄存器，故不需要!
  - Ifetch阶段和Dec/Reg阶段都没有控制信号
  - Exec阶段的控制信号有四个
    - ExtOp (扩展器操作)：1- 符号扩展；0- 零扩展
    - ALUSrc (ALU的B口来源)：1- 来源于扩展器；0- 来源于BusB
    - ALUOp (主控制器输出，用于辅助局部ALU控制逻辑来决定ALUCtrl)
    - RegDst (指定目的寄存器)：1- Rd；0- Rt
  - Mem阶段的控制信号有两个
    - MemWr (DM的写信号)：Store指令时为1，其他指令为0
    - Branch (是否为分支指令)：分支指令时为1，其他指令为0
  - Wr阶段的控制信号有两个
    - MemtoReg (寄存器的写入源)：1- DM输出；0- ALU输出
    - RegWr (寄存器堆写信号)：结果写寄存器的指令都为1，其他指令为0







# 提问

## Q & A

殷亚凤

智能软件与工程学院

苏州校区南雍楼东区225

yafeng@nju.edu.cn , <https://yafengnju.github.io/>



南京大學  
NANJING UNIVERSITY