第三讲 下址逻辑

翟象平

电邮: blueicezhaixp@nuaa.edu.cn

个人主页: http://cyber.nuaa.edu.cn

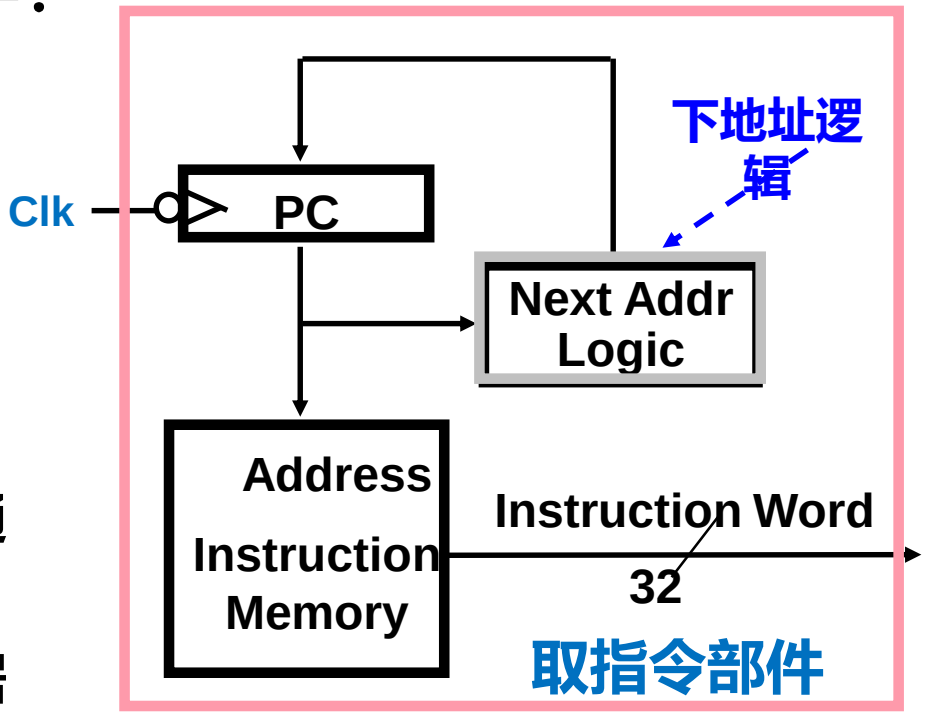
取指令部件(Instruction Fetch Unit)

- 每条指令都有的公共操作：
  - 取指令： M[PC]
  - 更新PC：PC ← PC + 4

转移（Branch and Jump）时，PC内容再次被更新为“转移目标地址”

取指后，各指令功能不同，数据通路中信息流动过程也不同

下面分别对每条指令进行相应数据通路的设计



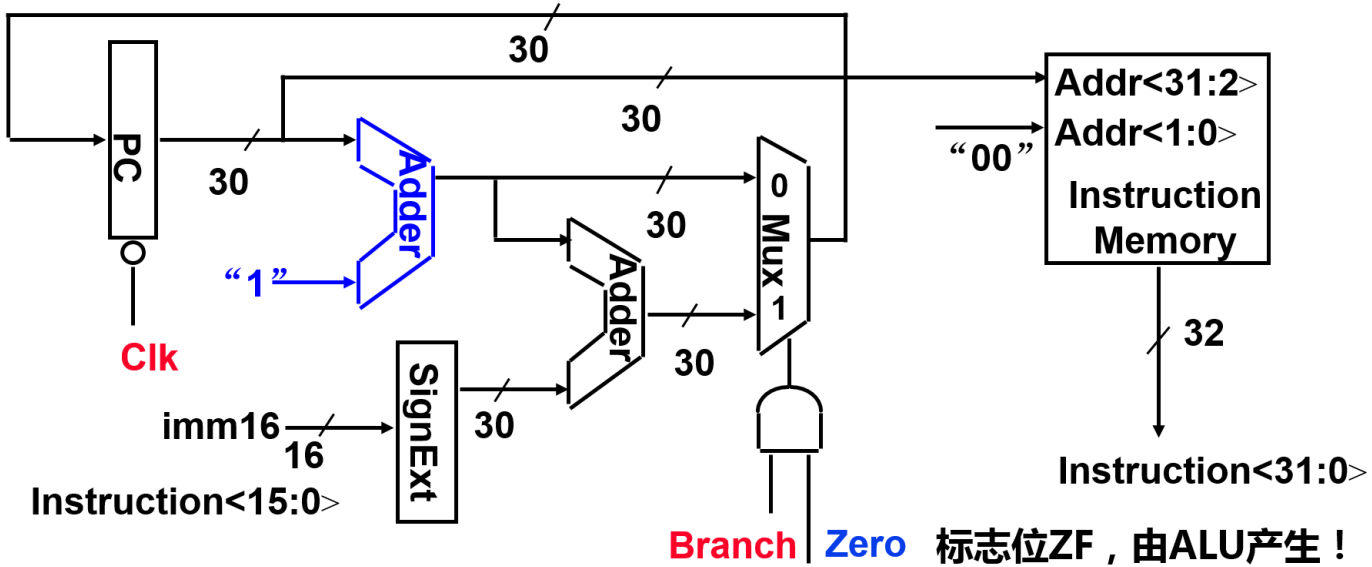
下址逻辑计算

- PC是一个32位地址:
  - 顺序执行时: PC<31:0> = PC<31:0> + 4
  - 转移执行时: PC<31:0> = PC<31:0> + 4 + SignExt[Imm16] × 4

- MIPS按字节编址，每条指令为32位，占4个字节，故PC的值总是4的倍数，即后两位为00，因此，PC只需要30位即可：
  - 顺序执行时: PC<31:2> = PC<31:2> + 1
  - 转移执行时: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - 取指令时: 指令地址 = PC<31:2> 串接 “00”

下址逻辑设计

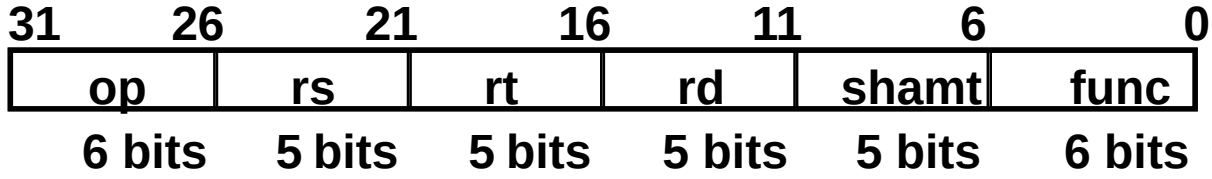
- Using a 30-bit PC：
  - 顺序执行时: PC<31:2> = PC<31:2> + 1
  - 转移执行时: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - 取指令时: 指令地址 = PC<31:2> 串接 “00”

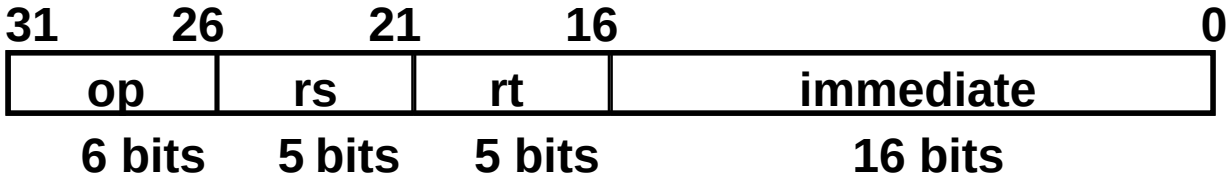


“ALU”和“Adder”有什么差别?

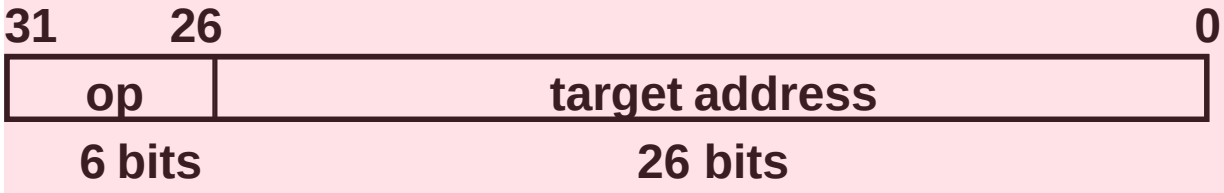
无条件转移指令

- 实现目标（7条指令）：
- ADD and SUBSTRACT
  - add rd, rs, rt
  - sub rd, rs, rt
- OR Immediate
  - ori rt, rs, imm16
- LOAD and STORE
  - lw rt, rs, imm16
  - sw rt, rs, imm16
- BRANCH
  - beq rs, rt, imm16
- JUMP
  - j target

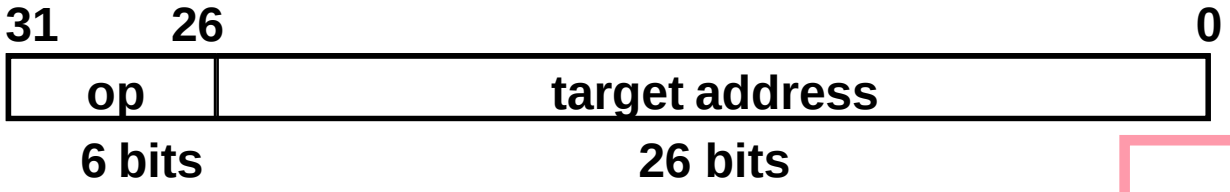




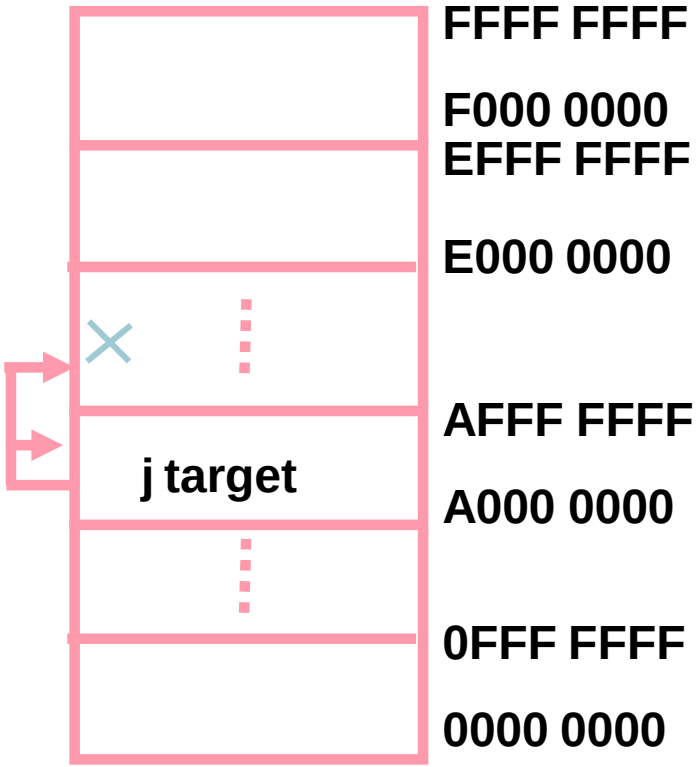
考虑Jump指令（无条件转移指令的代表）



无条件跳转指令（JUMP）



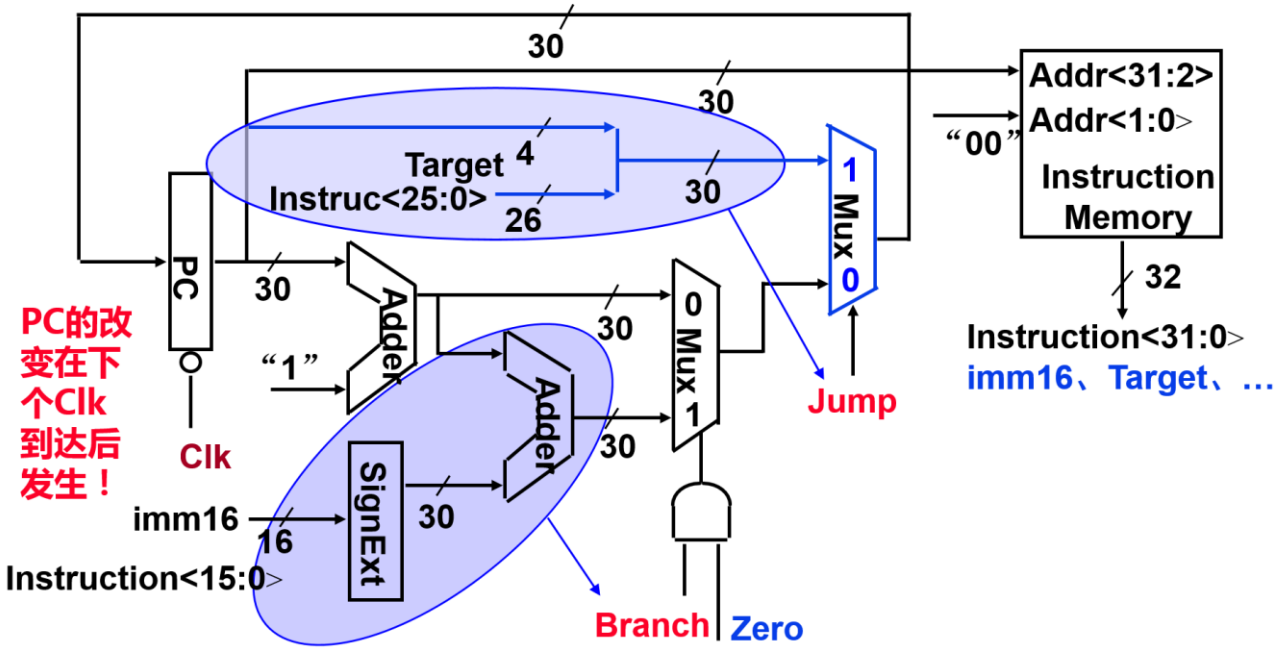
- j target
  - M[PC]　　取指令（公共操作，取指部件完成）
  - PC<31:2> ← PC<31:28> 串接 target<25:0>　　计算目标地址



- 不是相对寻址，而是绝对寻址

取址部件的数据通路

- j target
  - PC<31:2> ← PC<31:28> concat target<25:0>



RegDst=x

ExtOp=x

ALUSrc=x

MemtoReg=x

ALUctr=x,

RegWr=0,

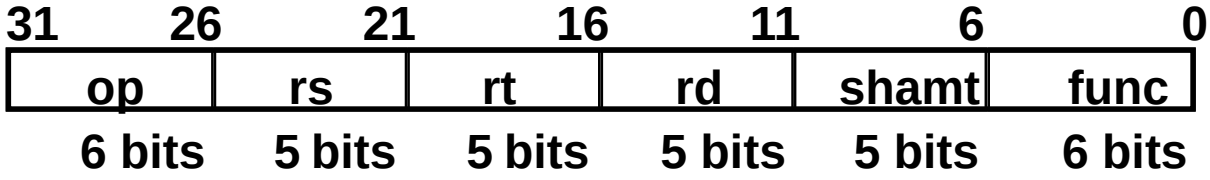
MemWr=0,

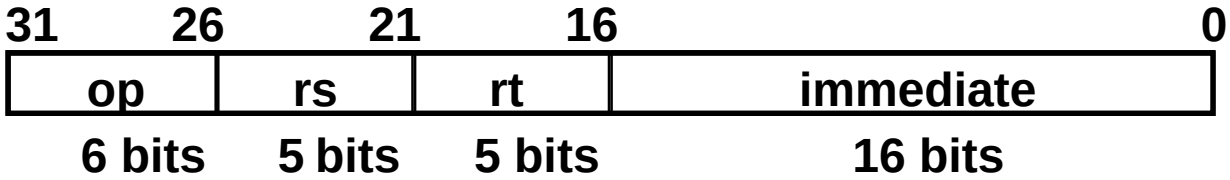
Branch=0,

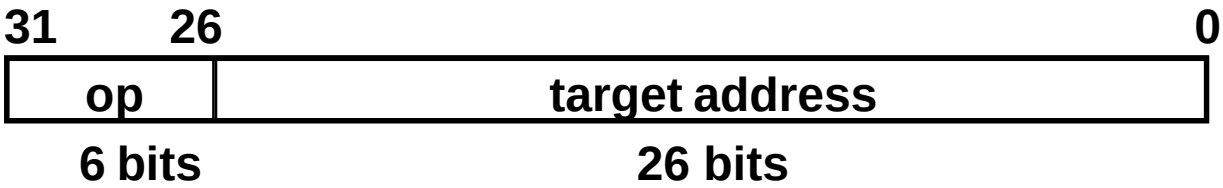
Jump=1

MIPS子集（数据通路）

- ADD and SUBSTRACT
  - add rd, rs, rt
  - sub rd, rs, rt
- OR Immediate
  - ori rt, rs, imm16
- LOAD and STORE
  - lw rt, rs, imm16
  - sw rt, rs, imm16
- BRANCH
  - beq rs, rt, imm16
- JUMP
  - j target

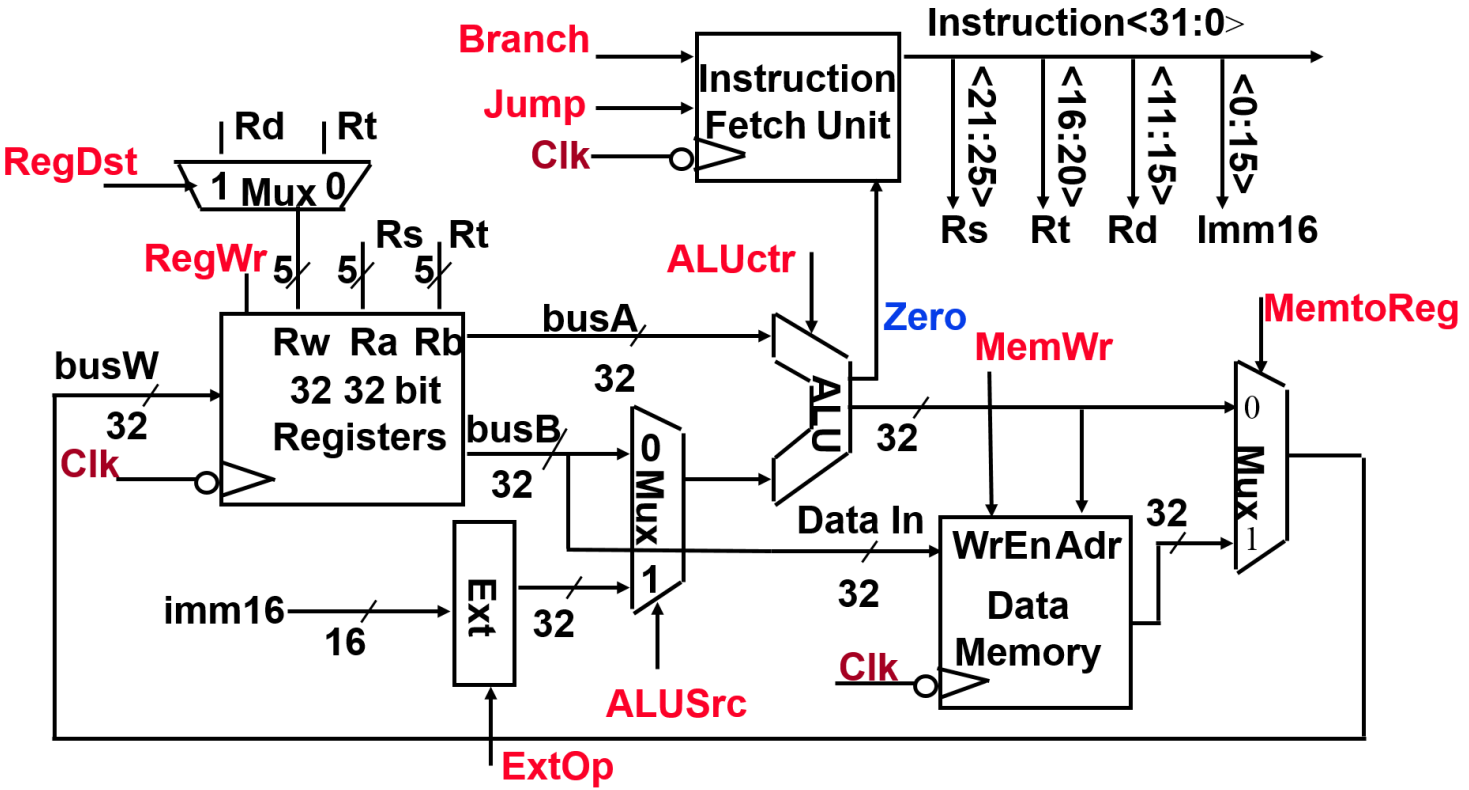






所有指令的数据通路都已设计好，合起来的数据通路是什么样的?

CPI举例



- 指令执行结果总是在下个时钟到来时开始保存在 寄存器 或 存储器 或 PC 中

Q & A

THANKS