计算机学院课程

计算机组成原理

支持I/O

高小鹏

北京航空航天大学计算机学院
系统结构研究所

容量与地址线

- 总容量=容量/单元×单元总量
- 地址线：决定单元的总量
  - 8位地址：256个单元
  - 20位地址：1M个单元
  - 32位地址：4G个单元
- 示例：4GB，仅仅是存储总量的概念
  - 在没有明确单元容量时，无法判断地址线宽度；反之亦然
  - 1B×4G；4B×1G；8B×512M

CPU地址空间

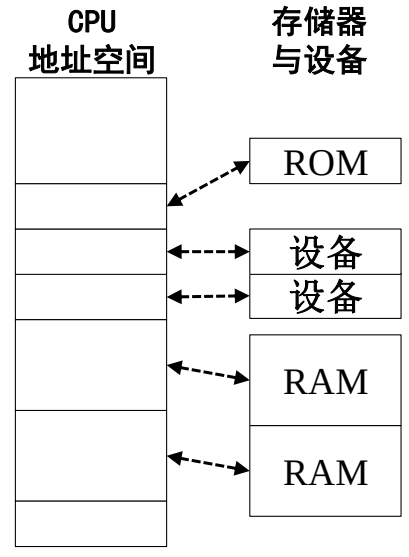
- 地址空间：数学概念，CPU能访问的空间
  - MIPS：CPU地址空间为4GB = 1B/单元 ×4G单元
    - ALU计算的地址宽度为32位，对应4G个地址单元
    - 每个地址单元宽度为byte
- 存储器：实体，关键属性是容量
  - 存储器属性：容量、数据线宽度、地址线宽度、性能、读写方式等
  - 容量 = 数据线宽度/8 × $\log_2^{\text{地址线宽度}}$
    - 例如SRAM芯片：32位数据，20位地址，则容量为4MB

存储器与地址空间

- 地址映射：将某段地址空间分配给某个存储器
- 例如：把1GB DRAM映射在4GB地址空间可以有N种方案
  - 方案1：位于0x0000_0000～0x3FFF_FFFF
  - 方案2：位于0x1000_0000～0x4FFF_FFFF
  - 方案3：位于0x0000_0003～0x4000_0002

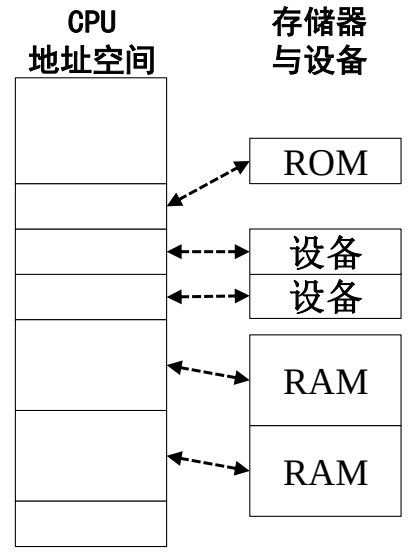
地址空间与存储器/设备

- CPU地址空间根据存储器/设备的容量需求被划分为若干区域
  - 每个区域对应一个存储器或设备



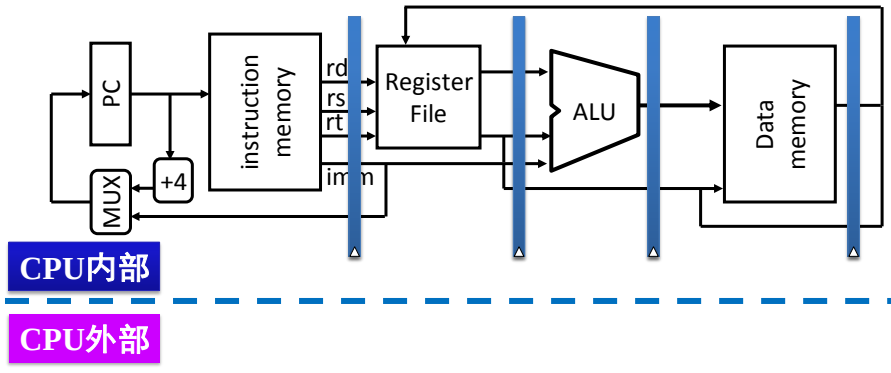
软件：读写存储器/设备

- LW/STORE指令
- 本质：地址对应于某个存储单元或设备寄存器
  - 设备寄存器被映射到地址空间
- 从软件角度看，读写设备寄存器与读写存储器无差别



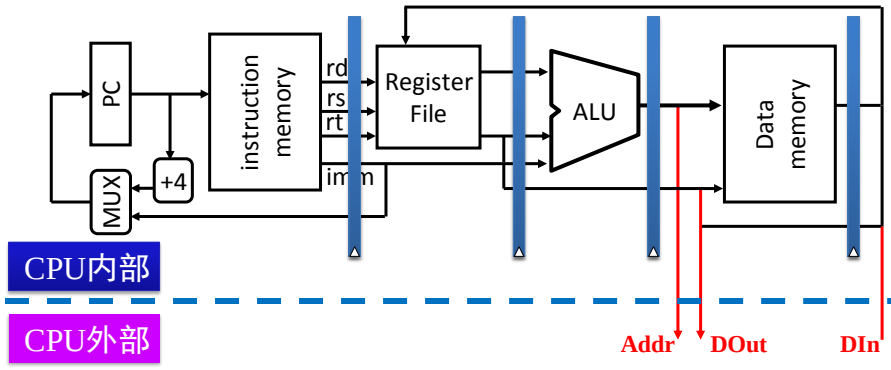
数据通路：增加信号

- 增加必要的地址、数据



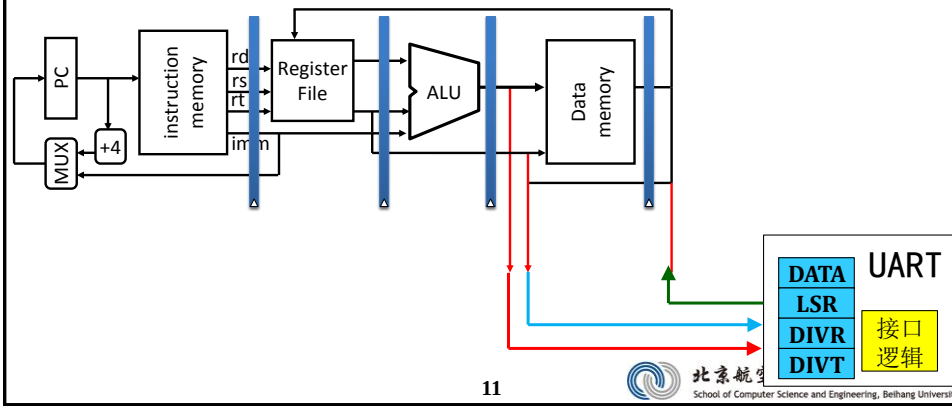
数据通路：增加信号

- 增加必要的地址、数据
  - Addr：ALU计算的存储器地址
  - DOut：CPU写数据
  - DIn：CPU读入的数据



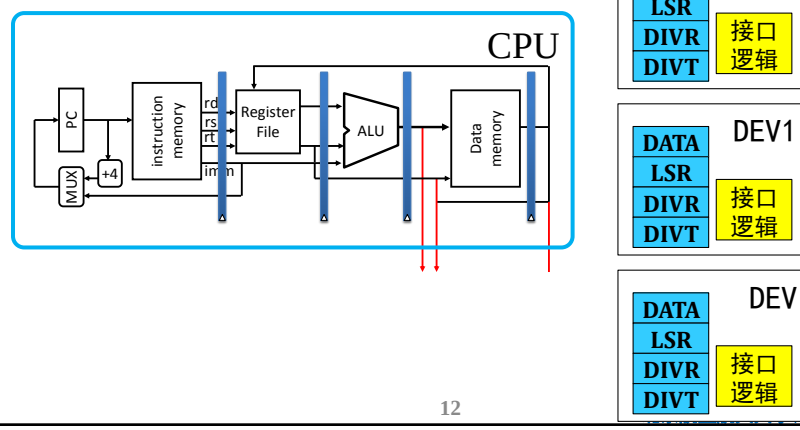
支持对I/O的访问：与设备对接

- 每个设备都有自己的$addr_{dev}$、din、dout
  - $Addr_{dev}$：选择设备内部的寄存器。其本质是offset
  - 设备内部的寄存器数量少，因此$Addr_{dev}$位数少
- Q：$Addr_{CPU}$位数多，怎么处理？
  - A：只保留必要的低位地址即可



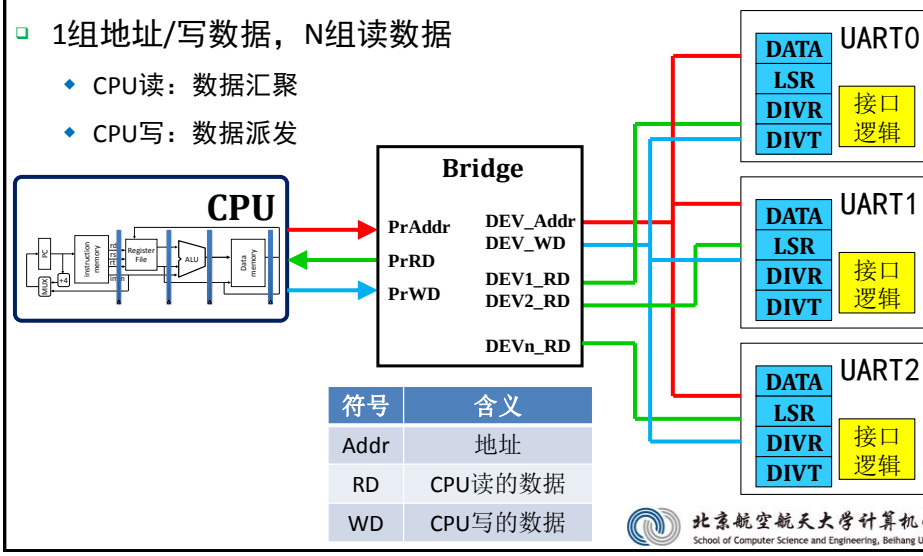
多个设备怎么办？

- CPU不能为每个设备都提供一套地址/数据
  - 否则会导致CPU设计变得复杂



增加新模块：Bridge

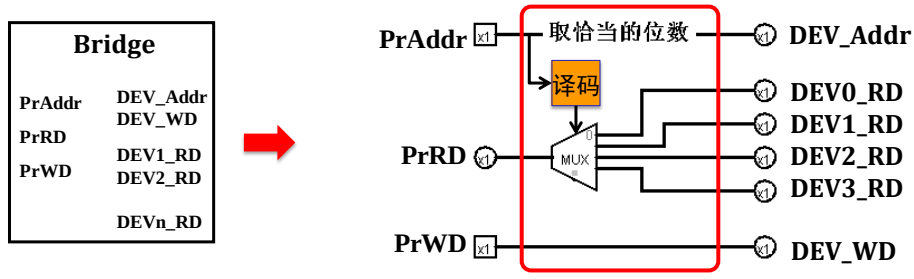
- Bridge：类似与网络switch
  - CPU侧：1组接口。设备侧：N组接口
- 1组地址/写数据，N组读数据
  - CPU读：数据汇聚
  - CPU写：数据派发



符号	含义
Addr	地址
RD	CPU读的数据
WD	CPU写的数据

Bridge功能及内部结构

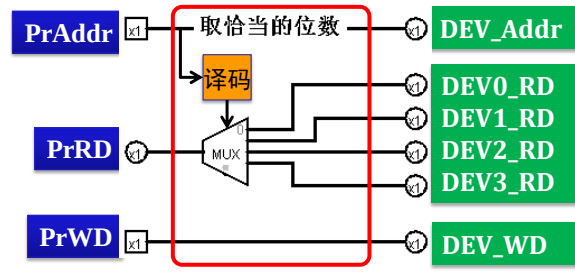
- 完成地址、数据转换，控制信号的产生
  - 地址
  - 读数据
  - 写数据



地址图

- 地址图：所有设备在地址空间的分布区域
  - CPU读写设备(其实是程序员)必须知道设备地址
  - Bridge也必须知道设备，否则无法完成译码
  - 示例：假设设备0～3均需要256B的地址空间需求

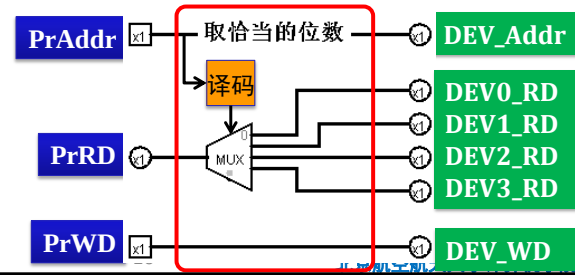
设备	MIPS地址范围	占用空间
DEV0	$A0000000_H$~ $A00000FF_H$	256字节
DEV1	$A0000100_H$~ $A00001FF_H$	256字节
DEV2	$A0000200_H$~ $A00002FF_H$	256字节
DEV3	$A0000300_H$~ $A00003FF_H$	256字节



Bridge功能(1)：输出地址1/2

- DEV_Addr地址：将PrAddr[X:2]直接输出即可
  - X：由N个设备中地址空间需求最大者决定
  - 所有设备都只接入各自必要的地址
- 示例：由于DEV3的地址空间为1MB，因此X为19

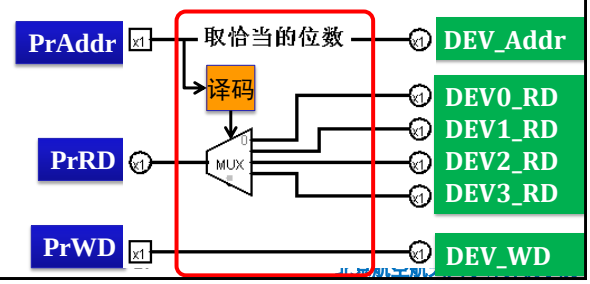
设备	MIPS地址范围	占用空间
DEV0	$A0000000_H$~ $A00000FF_H$	256字节
DEV1	$A0000100_H$~ $A00001FF_H$	256字节
DEV2	$A0000200_H$~ $A00002FF_H$	256字节
DEV3	$A0100000_H$~ $A01FFFFF_H$	1MB字节



Bridge功能(1)：输出地址2/2

- DEV0~2：引入DEV_Addr[7:2]即可
  - DEV0~2地址空间需求：256B
- DEV3：必须引入DEV_Addr[19:2]
  - DEV3地址空间需求：1MB

设备	MIPS地址范围	占用空间
DEV0	$A0000000_H$~ $A00000FF_H$	256字节
DEV1	$A0000100_H$~ $A00001FF_H$	256字节
DEV2	$A0000200_H$~ $A00002FF_H$	256字节
DEV3	$A0100000_H$~ $A01FFFFF_H$	1MB字节



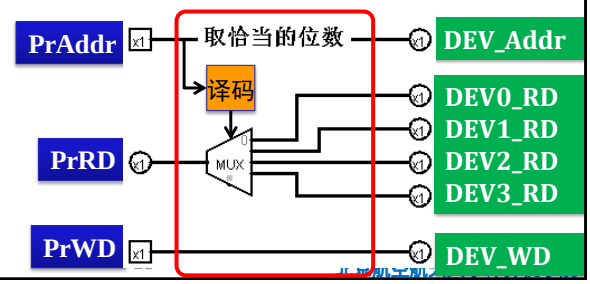
Bridge功能(2)：地址匹配

- 设备地址译码
  - 设备基地址：分为高位和低位
    - 基地址低位：位数由设备占用空间大小决定，也就是偏移地址的位数
    - 基地址高位：Bridge用于译码选择设备

31	X+1	X	0
基地址高位		基地址低位	

设备0 $A0000000_H$
设备1 $A0000100_H$
设备2 $A0000200_H$
设备3 $A0100000_H$

设备	MIPS地址范围	占用空间
DEV0	$A0000000_H$~ $A00000FF_H$	256字节
DEV1	$A0000100_H$~ $A00001FF_H$	256字节
DEV2	$A0000200_H$~ $A00002FF_H$	256字节
DEV3	$A0100000_H$~ $A01FFFFF_H$	1MB字节



Bridge功能(2)：地址匹配

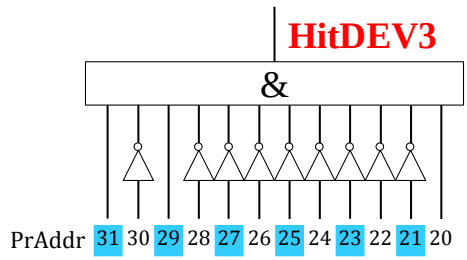
- 设备地址译码
  - 为每个设备产生一个译码信号

设备0 $A0000000_H$
设备1 $A0000100_H$
设备2 $A0000200_H$
设备3 $A0100000_H$

Verilog样例

```
assign HitDEV0 = (PrAddr[31:8] == 'hA00000) ;
…
assign HitDEV3 = (PrAddr[31:20] == 'hA01) ;
```

HitDEV3电路模型



Bridge功能(3)：CPU读数据

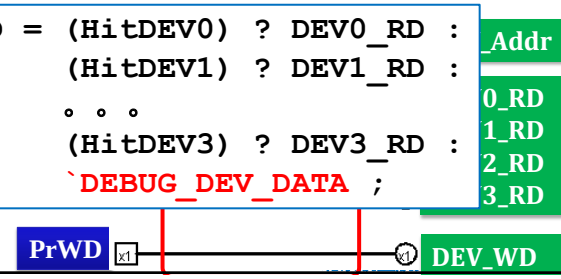
- 所有设备的数据输出汇聚至CPU的数据输入
- MUX的控制由PrAddr中某些位译码决定

常规写法

```
assign PrRD = (HitDEV0) ? DEV0_RD :
              (HitDEV1) ? DEV1_RD :
              。。。
              DEV3_RD ;
```

支持Debug写法

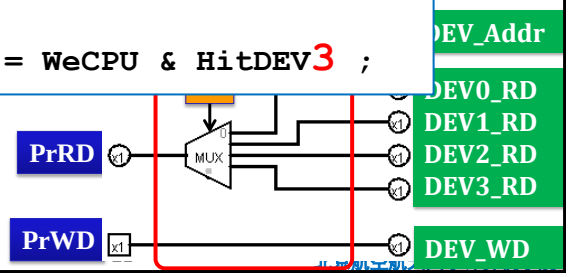
```
assign PrRD = (HitDEV0) ? DEV0_RD :
              (HitDEV1) ? DEV1_RD :
              。。。
              (HitDEV3) ? DEV3_RD :
              `DEBUG_DEV_DATA ;
```



Bridge功能(4)：CPU写数据

- CPU写数据：连接至所有设备的输入
  - 直通输出，不需要再转换
- 控制信号：We
  - 有多少个设备，就需要多少个We
  - Verilog样例代码：

```
assign WeDEV3 = WeCPU & HitDEV3 ;
```



问题：谁来区分存储器和设备？

- 假设$s0为0xA000_0000
- lw $t0, 0($s0)：是否同时会读存储器和设备？

设备	MIPS地址范围
DEV0	$A0000000_H$~ $A00000FF_H$
DEV1	$A0000100_H$~ $A00001FF_H$
DEV2	$A0000200_H$~ $A00002FF_H$
DEV3	$A0000300_H$~ $A00003FF_H$