Computer Organization and Design Review Notes

by 吴纵宇

chapter1 计算机概要与技术

1.1 引言

- 每当计算成本降低10倍，计算机的发展机遇就会增大10倍
- 计算机分类
  1. 个人计算机
  2. 服务器（高端服务器被称为超算 supercomputer）

     注：terabyte(TB)，原始定义为2^{40}字节，重新定义为10^{12}

     tebibyte(TiB)，为2^{40}字节
  3. 嵌入式计算机（数量最多的计算机）
- 代替PC的是个人移动设备PMD(Personal Mobile Device)
- **云计算**代替了传统的服务器，依赖于称为仓储规模计算机(Warehouse Scale Computer)
- 软件即服务(Software as a Service,SaaS)是软件工业的革命
- 程序性能的决定因素：
  - 算法
  - 建立程序并将其翻译成机器代码的软件系统
  - 计算机执行机器指令的有效性(可能包括I/O操作)
算法	**决定了源码语句级语句的数量和I/O操作的数量**
编程语言、编译器和体系结构	**决定了每条语句对应的计算机指令数量**
处理器和存储体系	**决定了指令的执行速度**
I/O系统（硬件和操作系统）	**决定了I/O操作可能的执行速度**

1.2 计算机系统结构的8个伟大思想

1. 面向摩尔定律的设计
2. 使用抽象简化设计
3. 加速大概率事件(common case fast)
4. 通过并行提高性能
5. 通过流水线提高性能
6. 通过预测提高性能
7. 存储器层次
8. 通过冗余提高可靠性

1.3 程序概念入门

- **应用软件**、**系统软件**和**硬件**(体现了**抽象**)
- 系统软件有很多种，但**操作系统**和**编译程序**是必须的两种。操作系统是用户程序和硬件之间的接口。
- 操作系统最为重要的作用：
  1. 处理基本的输入和输出操作
  2. 分配内存和外存
  3. 为多个应用程序提供共享计算机资源的服务
- 汇编程序、汇编语言和机器语言

1.4 硬件概念入门

- 任何一台计算机的基本功能：输入数据、输出数据、处理数据和存储数据
- 输入设备：为计算机提供信息的装置，如键盘
- 输出设备：将计算结果输出给用户（如显示器）
- 组成计算机的五个经典部件是：**输入设备**、**输出设备**、**存储器**、**数据通路**(运算器)和**控制器**，最后两个部件合称为**处理器**。

1.4.1 显示器

- [ ] LCD并非光源。大多数LCD显示器采用**动态矩阵显示**。
- [ ] 位图（bit map）
- [ ] 采用光栅刷新缓冲区(帧缓冲区)来保存位图

1.4.3 打开机箱

- **集成电路**(integrated circuit)俗**称芯片**(chip)

- 有时候**处理器**被称为**中央处理单元**
- **内存**(memory)是程序运行时的存储空间，也用于保存程序运行时使用的数据。内存由DRAM构成
- 处理器内部使用的是**缓存**(cache memory)，一般作为DRAM的缓冲，由SRAM(静态随机访问存储器)构成，比DRAM更加昂贵，集成度更低
- 最重要的抽象之一是硬件和底层软件之间的接口，被命名为计算机的**指令集体系结构(instruction ser architecture)**，简称**体系结构**
- 提供给程序员的**基本指令集**和**操作系统接口**称为**应用二进制接口**
- **实现**：遵循体系结抽象的硬件

1.4.4 数据安全

- 内存是**易失性存储器**(volatile memory)
- **非易失性存储器**(novolatile memory)
- 易失性存储器也被称为**主存**，非易失性存储器被称为二级存储器(磁盘在早期占主导地位，闪存在个人移动设备中代替了磁盘)

1.4.5 与其他计算机通信

- 最为普遍的网络类型是**以太网**。传输距离可达1000km，传输速度可达40Gbps
- 局域网(Local Area Network,LAN)和广域网

1.5 处理器和存储器制造技术

- 真空管---->晶体管---->集成电路---->超大规模集成电路---->甚大规模集成电路
- 晶体管是一种受电流控制的开关
- 硅是一种半导体，用特殊的方法对硅添加某些材料，可以把其细微的区域转变为以下三种类型之一：
  - 良好的导电体
  - 良好的绝缘体
  - 可控的导电体或绝缘体（类似开关）
- 集成电路的制造是从**硅锭(silicon crystal ingot)**开始的
- 合格芯片要装到I/O引脚上，这一过程被称为**封装**
- 芯片的成本取决于成品率以及芯片和晶圆的面积，与芯片面积之间的关系不是线性的

1.6 性能

1.6.1 性能的定义

- 个人计算机用户会对降低**响应时间**(也称为**执行时间**)更加感兴趣。
- **响应时间**：从一个任务开始到一个任务完成的时间，包括硬盘访问、内存活动、I\O活动、操作系统开销和CPU执行时间等
- 数据中心感兴趣的则是**吞吐率**，也叫**带宽**，表示单位时间内完成的任务数量
- 响应时间和吞吐率往往相互影响

1.6.2 性能的度量

- 程序的执行时间一般以秒为单位。对时间最直接的定义是墙上时钟时间，也叫响应时间，消逝时间等
- **CPU执行时间**，简称**CPU时间**，只表示在CPU上花费的时间而不包括等待I/O或运行其他程序的时间
- CPU时间还可以分为用户CPU时间和系统CPU时间
- 使用**系统性能**表示空载系统的响应时间，**CPU性能**表示**用户CPU时间**
- 时钟频率和时钟周期
- 一个程序的CPU执行时间 = 一个程序的CPU时钟周期数 x 时钟周期时间
- CPU时钟周期数 = 程序的指令数 x 每条指令需要的平均时钟周期数

- **每条指令需要的平均时钟周期数 = CPI**

- CPU时间 = 指令数 x CPI x 时钟周期时间
- 唯一能够被可靠测量的计算机性能指标是时间

1.7 功耗墙

- 占统治地位的集成电路技术是CMOS，主要的功耗来源是动态消耗，即在晶体管开关过程中产生的能耗。
- 能耗$\propto 1/2 * 负载电容 * 电压^2$
- 功耗$\propto 1/2 * 负载电容 * 电压^2 * 开关频率$
- 如果电压继续下降会使晶体管泄漏电流过大，就像水龙头不能被完全关闭
- 静态耗能也是存在的
- 功耗的极限迫使微处理器的设计发生了巨变
- 受到功耗、指令集并行程度和存储器长时间延迟时间的限制，单核处理器的性能增长放缓
- 微处理器(microprocessor)和处理器(processor,为了避免混淆，被称为core，核)

chapter2 指令：计算机的语言

2.1 引言

- 指令集：一个给定的计算机体系结构所包含的指令集合
- **存储程序概念**：多种类型的指令和数据均以数字形式存储于存储器中的概念，存储型计算机即源于此
- 设备简单性
- MIPS操作数有32个寄存器，2^{30}个存储器字。
- 寄存器：用于数据的快速存取。在MIPS中，只能对存放在寄存器中的数据执行算术操作
- MIPS使用字节编址，所以连续的字地址相差4
- 存储器字：存储器只能通过数据传输指令访问，用于保存数据结构、数组和溢出的寄存器
- 各种指令见教材
- 每条MIPS算术指令执行一个操作，并且有且仅有3个变量
- 硬件设计三大基本原则
  - 简单源于规整（每条指令有且仅有三个操作数）
  - 越小越快（寄存器的个数限制为32个，大量的寄存器可能会使时钟周期变长，因为电信号传输更远的距离必然花费更多的时间）
  - 优秀的设计需要适宜的折中方案

2.3 计算机硬件的操作数

- MIPS体系结构中寄存器大小为32位，称其为字
- 寄存器由硬件直接构成且数量有限
- MIPS约定用一个"$"加两个字符来代表一个寄存器
- 使用**$s0,$s1**表示C与Java程序中的变量所对应的寄存器，使用**$t0,$t1**等表示将程序编译为MIPS指令时所需的临时寄存器
- 数据结构（如数组和结构）是放在存储器中的
- **数据传输指令**：在存储器和寄存器之间移动数据的命令
- 为了访问存储器中的一个字，指令必须给出存储器**地址**
- 将数据从存储器复制到寄存器的数据传送指令通常叫做***取数***指令，**lw**
- ```
  g = h + A[8];
  lw $t0,32($s3)
  add $s1,$s2,$t0
  ```
- **基址**和**偏移量**
  ```

- 存放基址的寄存器称为*基址寄存器*
- 大多数体系结构按**字节编址**。因此，一个字的地址必须和它所包括的四个字节中的某个的地址相匹配
- **对齐限制**：数据地址与存储器的自然边界对齐的要求，可以加快数据传输
- **大端编址**和**小端编址**：大端为最左边字节，小端为最右边字节。
- MIPS指令集采用的是**大端编址（big-endian）**
- 将不常用的变量（或稍后使用）存回到存储器中的过程叫做**存储器溢出**
- **加立即数**（add immediate，addi

2.4 有符号数和无符号数

- **最低有效位**和**最高有效位**
- 符号和幅值表示法
- 原码、反码、补码解释https://blog.csdn.net/chenchao2017/article/details/79733278?utm_medium=distribute.pc_relevant_t0.none-task-blog-searchFromBaidu-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-searchFromBaidu-1.control
- 对取数指令来说有符号数和无符号数是有区别的。取回有符号数后需要使用符号位填充寄存器的所有剩余位，称为符号扩展。无符号数则是用0填充
- lb和lbu指令
- 符号扩展

2.5 计算机中指令的表示

- 机器指令分为若干**字段(field)**
- 指令的布局格式称为**指令格式**
- 所有MIPS指令都是32位
- 把指令的数字形式称为**机器语言**，这样的指令序列叫做**机器码**
- **R型指令**如下（用于寄存器）：

op（操作码）	rs	rt	rd	shamt（位移量）	funct（功能，指明op操作中的变体）
6位	5位	5位	5位	5位	6位

  - rs：第一个源操作数寄存器
  - rt：第二个源操作数寄存器
  - rd：用于存放操作结果的目的寄存器
  - shamt：位移量
- **I型指令**，立即数和数据传输指令用的就是这种格式：
op	rs	rt	constant or address
6	5	5	16

  - 在lw指令中，rt变成了接受取数结果的目的寄存器
- 定长指令的需求与设置尽可能多的寄存器的需求矛盾。寄存器数量的任何增长都需要在指令格式的各个寄存器字段至少增加1位。综合考虑这些限制和越小越快的设计原则，当今的大多数指令系统中有16个或32个通用寄存器

2.6 逻辑操作

- 为了保持三操作数的格式，MIPS的设计值引入了**或非NOR**指令吗，只要另一个操作数为0就相当于按位取反

2.7 决策指令

- 计算机与简单加速器的区别在于决策能力
- beq （如果相等则分支，branch if equal）
- beq reg1,reg2,L1
- bne（如果不相等则分支，branch if not equal）
- bne reg1,reg2,L1
- **beq**和**bne**称为**条件分支conditional branch**指令
- 在if语句的结尾处，需要引进另一种分支结构--无条件分支指令，命名为**jump**，简写成**j**

- ```
  if (i==j) f = g + h;
  else f = g - h;
  ```

  编译成汇编指令如下

  ```
  bne $s3,$s4,Else
  add $s0,$s1,s2
  j Exit
  Else:sub $s0,$s1,$s2
  Exit:
  ```

  以下为C语言的代码

  ```
  while (save[i] == k)
      i += 1;
  ```

  编译成汇编指令如下：

  ```
  Loop: sll $t1,$s3,2
  add $t1,$t1,$s6
  lw $t0,0($t1)
  bne $t0,$s5,Exit
  addi $s3,$s3,1

  j Loop
  Exit:
  ```

- 以分支指令结束的这类指令序列对编译非常重要，他们有专用的术语：**基本块**
- **基本块**：没有分支（末尾除外）并且**没有**分支目标（标签）的指令序列
- 小于则置位**slt**（set on less than）和**slti**

- ```
  slt $t0, $s3, $s4 # $t0 = 1 if $s3 < $s4
  slti $t0,$s2,10
  ```

- 比较指令应该具有分清**有符号数**和**无符号数**的能力
- sltu和sltiu用于处理无符号数
- 将有符号数作为无符号数来处理，是一种检验0<=x<y的低开销方法

- ```
  sltu $t0,$s1,$t2
  beq $t0, $zero, IndexOutOfBounds
  ```

- 实现switch语句的最简单方法是借助一系列的条件判断，将switch语句转换为一系列的if-esle-then嵌套，有时候更有效的另一种方法是将多个指令序列分支的地址编为一张表

- **转移地址表**或**转移表**，是一个由代码中标签所对应地址构成的数组。程序需要跳转的时候首先将表中适当的项加载到寄存器中，然后使用寄存器中的地址进行跳转，这种指令叫做**jr（jump register）**

2.8 计算机硬件对过程的支持

- **过程**或**函数**是程序员进行结构化编程的工具

- 过程是软件中实现**抽象**的一种方法

- 在过程运行中，程序必须遵循以下6个步骤：

  1. 将参数放在函数可以访问的位置
  2. 将控制转交给函数
  3. 获得函数所需的存储资源
  4. 执行需要的任务
  5. 将结果的值放在调用程序可以访问的位置
  6. 将控制返回初始点，因为一个函数可能由一个程序中的多个点调用

- MIPS在为函数调用分配32个寄存器时遵循以下约定：

  - $a0~$a3:用于传递参数的四个**参数寄存器**
  - $v0~$v1:用于返回值的两个**值寄存器**
  - $ra:用于返回起始点的**返回地址寄存器**

- 函数调用指令：跳转到某个地址的同时将下一条指令的地址保存在寄存器$ra中

- 这条**跳转和链接指令jal**的格式为：jal ProcedureAddress

- **jr指令**

- 调用者和被调用者

- 使用一个寄存器来保存当前运行的指令地址，为**指令地址寄存器**，由于历史原因称其为**程序计数器**，在MIPS指令中缩写为PC。jal指令实际上将PC+4保存在寄存器$ra中，从而将链接
  ```

指向下一条指令

- 如果编译器需要使用多于4个参数寄存器和2个返回值寄存器，即需要将寄存器换出到存储器的一个例子。
- 换出寄存器最理想的数据结构是**栈（stack)**
- 栈指针（sp）以字为单位进行调整，MIPS为栈指针准备了第**29**号寄存器并命名为$sp。压栈push和出栈pop
- 例子：编译一个不调用其他过程的C过程
- ```
  int leaf_example(int g,int h, int i, int j)
  {
      int f;
      f = (g+h) - (i+j);
  }
  ```
- 转换成汇编语言如下：
- ```
  addi $sp, $sp, -12
  sw $t1, 8($sp)
  sw $t0, 4($sp)
  sw $s0, 0($sp)
  add $t0,$a0,$a1
  add $t1,$a2,$a3
  sub $s0,$t0,$t1
  add $v0,$s0,$zero
  lw $s0,0($sp)
  lw $t0,4($sp)
  lw $t1,8($sp)
  addi $sp,$sp,12
  jr $ra
  ```
- $t0~$t9: 10个临时寄存器，在过程调中不必被调用者保存
- $s0~$s7: 8个保留寄存器，在过程中必须被保存（一旦被使用，由调用者保存或恢复）
- 不调用其他过程的过程称为**叶过程**

- ```
  int fact (int n) {
      if (n<1)
          return 1;
      else
          return n*fact(n-1)
  }
  ```
- 转换成相应的汇编代码为：

  ```
  fact:
      addi $sp, $sp, -8 # 为参数和返回值开辟空间
      sw $ra, 4($sp)    # 返回值地址
      sw $a0, 0($sp)    # 参数$a0
      slti $t0, $a0, 1
      beq $t0, $zero, L1
      addi $v0, $zero,1
      addi $sp, $sp, 8
      jr $ra
      L1: addi $a0, $a0, -1
          jal fact
      lw $a0, 0($sp)
      lw $ra, 4($sp)
      addi $sp, $sp, 8
      mul $v0, $a0, $v0
      jr $ra
  ```
- C语言中的一个变量通常对应存储中的一个位置，其解释取决于其类型和存储方式
- C语言有两种存储方式：动态的和静态的
- 为了简化静态数据的访问，MIPS保留率另一个寄存器，称为全局指针（global pointer），即$gp
保留	**不保留**
保存寄存器:$s0~$7	临时寄存器
栈指针寄存器:$sp	参数寄存器
返回地址寄存器:$ra	返回值寄存器
栈指针以上的栈	栈指针以下的栈
- 栈的最后一点复杂性是栈还需要存储过程的局部变量，但这些变量不适用于寄存器，例如局部的数组和结构体
  ```

- 栈中包含函数所保存的寄存器和局部变量的片段称为**过程帧**或**活动记录**
- **帧指针fp**：指向该帧的第一个字（一般是保存的参数寄存器）
- 虽然使用栈指针和少量的地址运算就能完成对变量的引用，但使用固定的帧指针引用变量会更为简单。如果在一个过程中栈没有局部变量，编译器将可以不设置和不恢复帧指针以节省时间
- 除了动态变量对函数是局部有效之外，C程序员还需要在内存中为静态变量和动态数据结构提供空间。栈的内存由高端开始并向下增长
- 内存低端的第一部分是保留的，之后是机器代码的第一部分，通常称为**代码段**，代码段之上的代码称为**静态数据段**，是存储常量和其他静态变量的空间。链表等数据结构对应的段习惯上称为**堆**。
- 栈和堆允许互相增长，从而能在两个段此消彼长的过程中达到内存的高效使用
- MIPS约定：称为$at的寄存器1被汇编器所保留，称为$k0~~$k1的寄存器26~~27倍操作系统所保留
- 如果参数多于4个，MIPS约定将额外的参数放在栈中帧指针的上方
- 尾递归用尾迭代替换

2.9 人机交互

- lb（load byte）指令从内部中读出一个字节，并将其放在一个寄存器最右边的8位。
- sb（store byte）指令把一个寄存器最右边的8位取出来然后写到内存中。
- 可按照下面的顺序复制一个字节：
- ```
  lb $t0,0($sp)
  sb $t0,0($gp)
  ```
- 表示一个字符串的方式有3种：
  - 保留字符串的第一个位置用于给出字符串的长度
  - 附加一个带有字符串长度的变量
  - 字符串最后的位置用一个字符来标识其结尾
- MIPS指令集包含显示的读取和存储16位半字的指令。**load half（lh）**
- 更加常用的是无符号读取半字
  ```

2.10 MIPS指令中32位立即数和寻址

- 虽然保持MIPS指令为32位长简化了硬件，但有时使用32位常量或32位地址更加方便。
- 尽管常数往往比较短而且适合16位字段，但有时它们会更大。
- 读取立即数高位指令**（lui）**专门用于设置寄存器中常数的高16位，低16位用0填充。允许后续指令设置常数的低16位
- **lui $t0, 255** #$t0 is register 8对应的机器语言：
-

001111	**00000**	**01000**	**0000000011111111**

- 执行完该条指令后寄存器$t0的值
-

0000000011111111	**0000000000000000**

- 编译器或汇编程序必须把大的常数分解为若干小的常数，然后再合并到一个寄存器中。立即数字段大小的限制，无论在取/存数指令中对存储器的地址还是在立即数指令中对常数都可能带来问题。汇编程序必须有一个可用的临时寄存器来创建长整数值。这是给汇编程序保留$at寄存器的一个原因
- **ori指令**将低16位写入常数中
- MIPS跳转指令寻址采用最简单的寻址方式。它们使用最后一种MIPS指令格式，称为J型。J型除了6位操作码之外，其余26位都是地址字段。
- 和跳转指令不同，条件分支除了规定分支地址之外还必须指定两个操作数。因此bne被汇编为下面的指令，只保留了16位用于指定分支地址
-

5	**16**	**17**	**Exit**
6位	5位	5位	16位

- 如果让程序地址适应该16位字段，则地址太小，不太现实
- 另一个方法：指定一个总是加到分支地址上的寄存器，这样分支地址的地址可按如下方式计算：
- 程序计数器 = 寄存器 + 分支地址
- 问题：使用哪一个寄存器呢？答案：取决于条件分支是如何使用的。

- 一半条件分支的跳转距离小于16条指令。因为PC包含当前指令的地址，如果我们使用PC来作为增加地址的寄存器，则PC是一个理想的选择。这种称为**PC相对寻址**。
- 跳转链接指令并非总是靠近调用者的过程，所以它们经常使用其他寻址方式。
- PC相对寻址时所加的地址被设计为字地址而不是字节地址。
- pc是32位。跳转指令仅仅替代PC的低28位，而高4位保持不变。装载器和链接器必须十分小心以避免超过256MB的寻址界限
- MIPS指令采用字节寻址
- 分支指令的距离大于16位时，需要插入一个无条件跳转
- 多种不同的寻址形式一般统称为**寻址模式**
- 寻址模式
  - 立即数寻址（操作数是位于指令自身中的常数）

op	**rs**	**rt**	**立即数**

  - 寄存器寻址（操作数是寄存器）

op	**rs**	**rt**	**rd**	**...**	**funct**

  rs中的地址

  - 基址寻址（操作数在内存中，其地址是指令中基址寄存器和常数的和）

op	**rs**	**rt**	**形式地址**

  - PC相对寻址（地址是PC和指令中常数的和）

op	**rs**	**rt**	**形式地址**

  - 伪直接寻址（地址由指令中26位字段和PC高位相连而成）

op	**形式地址**

- 有时候必须通过逆向工程将机器语言恢复到最初的汇编语言，比如检查“**核心转储**”时。

2.11 并行与指令：同步

- 任务之间需要同步，否则就有发生**数据竞争**的危险
- 加锁、解锁可以创造一个仅允许单个处理器操作的区域，叫做**互斥区**

- 在多处理器中实现同步需要一组**硬件原语**，提供对存储单元进行原子读和原子写的能力，使得在进行存储器原子读或原子写时任何其他操作都不得插入
- 原子交换原语
- 用交换原语实现同步的关键是操作的原子性：交换操作是不可分割的，并且由硬件对两个同时执行的交换操作进行排序。有可能两个处理器尝试同时置位同步变量，但这两个处理器认为它们同时成功设置了同步变量是不可能的。
- 一种可行的方法是采取**指令对**：其中第二条指令返回一个表明这对指令是否原子执行的标志值。
- MIPS处理器中这一对指令包括一条叫做**链接取数**的特殊取数指令和一条叫做**条件存数**的特殊存数指令

2.12 翻译并执行程序

编译器、汇编器、链接器和加载器

chapter3 计算机的算术运算

- 无符号数通常用于表示内存地址，这种情况下溢出可以忽略，因此计算机设计者必须提供一种方法能够在某些情况下忽略溢出。
- add、addi、sub这三条指令在溢出时产生异常。
- addu、addiu、subu这三条指令在溢出时不会发生异常
- 异常和中断
- 注意：addiu。u代表着不会产生溢出异常，但是16位立即数字段要符号扩展成32位的，。因此即使操作是“无符号”的，立即数字段也是有符号的
- 异常程序计数器EPC，来保存导致异常的指令地址。指令mfc0用来将EPC存入一个通用寄存器，从而使MIPS可以通过寄存器跳转回异常指令处

chapter4 处理器

4.1 引言

- 计算机的性能由三个关键因素决定：CPI、时钟周期长度和指令数
- 每条指令的前两步是一样的
  1. PC指向指令所在的存储单元，并从中取出指令
  2. 通过指令字段内容，选择读取一个或两个寄存器。对于取字指令，只需读取一个寄存器，而其他大多数指令要求读取两个寄存器。
- 除跳转指令外的所有指令在读取寄存器后，都要调用算数逻辑单元ALU。
- 连接功能单元的**粗线**表示总线

- **控制单元**：以指令为输入，决定功能单元和两个多选器的控制信号
- 时钟周期必须设置为足够容纳执行时间最长的指令

4.2 逻辑设计的一般方法

- 组合单元和状态单元
- 组合单元：一个**操作单元**。处理数据值，输出只取决于当前的输入。如与门与ALU。
- 状态单元：一个**存储单元**。如果一个单元内部带有内部存储功能，它就包含状态，称之为状态单元。如指令存储器、数据存储器和寄存器都是状态单元
- 逻辑状态上最简单的一种状态单元是D触发器
- 包含状态的逻辑单元又被称为时序的
- **时钟方法**规定了信号可以读出和写入的时间。
- 为简单起见，若某状态单元**在每个有效的时钟边沿都进行写入操作**，则可忽略**写控制信号**。
- **有效**表示信号为逻辑高，**无效**表示信号为逻辑低

4.3 建立数据通路

- 设计数据通路比较合理的方法是首先分析每种MIPS指令执行时主要需要哪些部件
- 数据通路部件：一个用来操作或保存处理器中数据的单元。在MIPS实现中，数据通路部件包括**指令存储器**、**数据存储器**、**寄存器堆**、**ALU**和**加法器**
- 因为指令存储器是只读的，所以我们将其视为组合逻辑。
- PC是32位的寄存器，在**每个**时钟周期末都会被写入，所以不需要控制信号
- 处理器的32个通用寄存器位于一个叫**寄存器堆**的结构中
- 寄存器堆：包含一系列寄存器的状态单元，可以通过提供寄存器号进行读写
- R型指令（算术逻辑指令）
  - 每条指令都要从寄存器堆读出两个数据字，再写入一个数据字
  - 为读出一个数据字，寄存器堆需要两个输入：一个要读的**寄存器号**和一个从寄存器堆读出结果的**输出指示**
  - 实现R型指令的ALU操作所需要的两个单元：寄存器堆和ALU
- 指令集规定计算分支地址时使用的基地址，是分支指令的**下一条**指令的地址
- 分支数据通路一般需要两个操作：计算分支目标地址和比较操作数
- **存取指令**需要部件：寄存器堆、ALU、数据存储单元和符号拓展单元
- **数据存储单元**：是一个**状态单元**，两个输入为地址和所写数据
- 在实际的MIPS指令中，分支指令是“延迟的”，即无论分支条件是否满足，它之后的那条指令总是被执行。（设计原因：减轻流水线对分支指令的影响）

4.4 一个简单的实现机制

- 对于R型指令，根据指令低6位的funct字段，ALU执行5种操作中的一种
- 使用一个小的控制单元即可生成4位的ALU控制信号，其输入为指令的funct字段和2位的ALUop字段。
- 多级译码的方法（主控制单元生成ALUOp作为ALU控制单元的输入，再由ALU控制单元生产真值的控制ALU的信号）是一种常用的实现方式。使用多级译码可以减少主控制单元的规模。
- 9位控制信号（7个1位状态和2位的ALUop）
- 除PCSrc控制信号外，所有控制信号都可以只根据指令的操作码（高6位）来确定
- ALUop加上指令的低6位funct即可确定ALU执行的操作
- 单周期实现：一个时钟周期执行一条指令的实现机制。虽然容易理解，但是太慢而不实用

4.5 流水线

- 流水线是一种实现多条指令重叠执行的技术
- 流水线实际上是改进了吞吐率
- MIPS指令的一些特点：
  1. 所有MIPS指令的长度都是相同的。这一限制简化了流水线的第一级取指与第二级译码
  2. MIPS只有很少的几种指令格式，并且每一条指令中的源寄存器字段都是相同的
  3. MIPS中存储器操作数仅出现在存取指令中

4.5.2 流水线冒险

- 在下一个时钟周期中下一条指令不能执行，这种情况称为冒险（hazard）

  1. **结构冒险**：因缺乏硬件支持而导致指令不能在约定的时钟周期内执行的情况
  2. **数据冒险**：也称为流水线数据冒险：无法提供指令执行所需要的数据而导致指令不能在预订的时钟周期内执行的情况。

  从内部资源直接提前得到缺少的运算项的过程称为**前推**或者**旁路**（一种解决数据冒险的方式，具体做法是从内部寄存器而非程序员可见的寄存器或存储器中提前取出数据）。

- IF：取指阶段。ID：指令的译码或寄存器堆的读取阶段。EX：指令的执行阶段。MEM：存储器的访问阶段。WB：写回阶段

- 右半边阴影表示它们在此步骤被读取，左半边的阴影表示它们在此步骤被写入。

- **取数-使用型数据冒险**：流水线不得不阻塞一个周期。

- **流水线阻塞**，也经常被称为**气泡**

- **3.控制冒险**：决策依赖于下一条指令的结果，而其他指令正在执行中。控制冒险也称为分支冒险。因为取到的指令并不是需要的，或者说指令地址的变化并不是流水线所预期的而导致指令不能在预定的时钟周期内执行。
- 有两种方法可以解决控制冒险：阻塞或者预测
- 计算机采取预测的方法来处理分支结构
- 动态预测方法。一种比较普遍的实现方法是保存每次分支的历史记录，然后利用这个历史记录来预测
- 除了存储系统以外，流水线的有效运作是决定处理器CPI乃至其性能最重要的因素。
- 结构冒险一般出现在浮点单元附近，浮点单元是一个几乎不可能完全流水的地方。
- 控制冒险一般出现在整数程序中
- 数据冒险在整数和浮点程序中都可能成为性能瓶颈
- 流水线并不能减少单一指令的执行时间，也称为***延迟***

4.6 流水线数据通路及其控制

- 1. IF：取指令
  2. ID：指令译码
  3. EX：执行或计算地址
  4. MEM：数据存储器访问
  5. WB：写回
-