计算机组成原理

0.System Verilog

锁存器

锁存器是一种在异步时序电路系统中，对输入信号电平敏感的单元，用来存储信息。锁存器在**数据未锁存**时，输出端的信号随输入信号变化，就像信号通过一个缓冲器；一旦锁存信号有效，则数据被锁存，输入信号不起作用。（与触发器关于时钟信号敏感不同）

锁存器的危害：对毛刺敏感，**不能异步复位**，且会使得静态时序分析变得非常复杂。（在编写`System Veriog`程序时，务必记得在`case`后加上`default`语句，在`if`判断后加上`else`）

注意事项

1. 通过命名来区分寄存器和信号。
2. 信号或寄存器应当**仅在一个块中赋值。**
3. 讲时序逻辑和组合逻辑写在不同的块中。
4. **每个寄存器应当只在一个时钟上升沿触发。**
5. 寄存器应该实现复位逻辑，并且复位到常量（如果编写的硬件逻辑面向Xilinx FPGA），建议采用**同步复位**的方法。
6. **组合逻辑需要保证每个分支下每个信号都有赋值。**为了防止遗忘，可以在分支开头设置一个**默认值。**
7. 如有必要可以对寄存器设置FPGA启动初始值。

```
logic some_reg;
initial some_reg = 1'b0;
```

8. 组合逻辑块中的赋值语句之间若有依赖，则需要保证赋值顺序。
9. **组合逻辑块中使用阻塞赋值`=`，时序逻辑块中使用非阻塞赋值`<=`。**
10. 异步复位中边沿触发要与`if`判断语句极性一致。即，异步复位时，如果复位信号是高有效，那么敏感信号应该写`posedge reset`，判断语句应该写`if (reset)`。反之同理。
11. 在编写需要通配符的`case`语句时，使用`casez`而不是`casex`，它们的区别在于二者遇到输入数据为`x`类型时，匹配的行为不一样。
12. `== For comparing bits (0 or 1) === For comparing all 4 states (0, 1, x, z)`

1.计算机的指令系统

1.1计算机程序及分类

程序的最小单元是指令，同时，指令也是计算机硬件执行程序的最小单位。

Von Neumann结构计算机：

1. 存储程序计算机：程序由指令构成，程序功能通过指令序列描述
2. 顺序执行指令：用PC指示当前被执行的指令，从存储器中读出指令执行

1.2指令系统

- 指令的功能分类
    1. **数据运算指令**：算术运算、逻辑运算
    2. **数据传输指令**：内存/寄存器，寄存器/寄存器
    3. **控制指令**：无条件跳转，条件跳转，子程序的支持（调用与返回）
    4. **输入输出指令**：与输入输出端口的数据传输
    5. 其他指令：停机、开/关中断、特权指令、设置条件码
- 指令格式：操作码、操作数地址的**二进制**分配方案

  指令字：完整的一条指令的二进制表示

  指令字长：指令字中二进制代码的位数

  > **机器字长**：计算机能够直接处理的二进制数据的位数

- 寻址方式：指**确定本条指令的操作数地址及下一条要执行的指令地址的方法。**

  在指令的操作数地址字段中，可能需要指出的包括：

    1. 运算器中的累加器的编号或专用寄存器名称
    2. 输入/输出指令中用到的I/O设备的入出端口地址
    3. 内存储器的一个存储单元的地址
- 评价计算机性能的指标：
    1. 吞吐量：单位时间内完成的任务数量
    2. 响应时间：完成任务的时间
- 指令系统分类
    1. **复杂指令集CISC**
    2. **精简指令集RISC**
    3. **超长指令字VLIW**
- RISC-V指令系统

  > 本课程采用ThinPAD RISC-V指令系统。

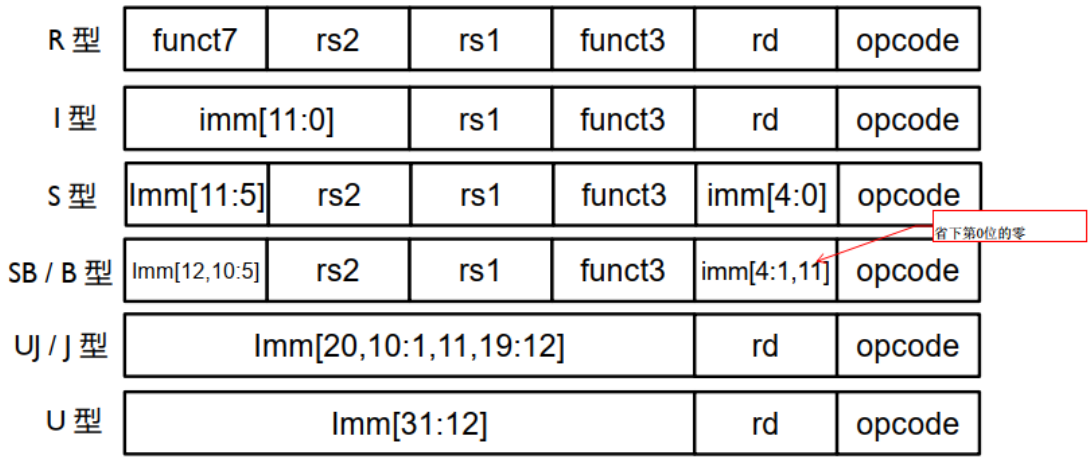
  指令格式：**32位固定字长**，操作码**位置**及**长度**固定，寻址方式简单

  共设计有19（基础指令）+6（支持优先级，支持中断与异常的指令）+1（支持虚拟地址的指令）条指令，并且可根据需要进行扩展。

  RISC-V指令格式：

RISC-V 指令格式（32位）

- 所有的指令都是32位字长，有 6 种指令格式：寄存器型，立即数型，存储型，分支指令、跳转指令和大立即数



22

1.3数据表示及检错纠错

码距：指任意两个合法码之间至少有几个二进制位不同。合理增大码距，能提高发现错误的能力，但表示一定数量的合法码所使用的二进制位数会变多。**码距和编码方案将决定某个码的检错纠错能力。**

常用检错纠错码

1. **奇偶校验码**：分为奇校验与偶校验，在k位数据码之外增加1位校验位，使k+1位码字中取值为1的位数总保持为偶数（偶校验）或奇数（奇校验）。
2. **海明校验码**：为k个数据位设立r个校验位，使k+r位的码字同时具有以下两个特性：能发现并改正k+r位中任何一位出错；能发现k+r位中任何两位同时出错，但已无法改正。

   原理：用k个数据中不同的数据位组合来形成每个校验位的值，使任何一个数据位出错时，讲影响r个校验位中不同的校验位组合。因此，位数r和k需要满足：$2^r \geq r+k+1$，即用2^r个编码分别表示k个数据位、r个校验位中哪一位出错，以及都不错；或$2^{r-1} \geq k+r$，用r-1位校验码位出错位编码，再单独设一位用以区分是1位还是2位同时出错，这种实现在实际中更实用。

1.4ALU

补码乘法规则（布斯算法）

$$
\begin{aligned}
[x \cdot y]_{补} &= [x]_{补} \cdot (-2^{n-1}y_{n-1} + \sum_{i=0}^{n-2} y_i 2^i) \\
&= [x]_{补} \cdot [2^{n-1}(y_{n-2} - y_{n-1}) + 2^{n-2}(y_{n-3} - y_{n-2}) + \cdots + 2^0(0 - y_0)] \\
&= [x]_{补} \cdot \sum_{i=0}^{n-1} 2^i (y_{i-1} - y_i)
\end{aligned}
$$

即根据乘数相邻两位的不同组合，确定是$+[x]_{补}$或是$-[x]_{补}$。

补码除法规则：

1. 被除数与除数同号时，用被除数减去除数；被除数与除数异号时，用被除数加上除数；

2. 余数和除数同号，商为1，余数和商同时左移一位，下次余数减去除数；余数和除数异号时，商为0，余数和商同时左移一位，下次余数加上除数；
3. 重复步骤2，共n+1步（n为除符号位外的位数）

1.5运算器部件组成及设计

运算器的基本功能：完成算数、逻辑运算，产生运算结果，并给出运算结果的状态信息（C，Z，V，S），暂存运算所用的操作数，暂存运算的中间结果，以及输出运算结果。

定点运算器的组成包括**算术逻辑单元ALU**、**通用寄存器组**、**乘商寄存器。**

18位控制指令=8位（A、B口地址各4位）+9位控制信号（3×3）+1位进位

2.控制器原理及设计

2.1 RISC-V指令系统

- 寄存器组主要包括**通用寄存器**和**控制状态寄存器**；**CSR寄存器**用于配置或记录一些运行的状态，其是处理器核内部的寄存器。

  在RISC-V中有32个寄存器，每个寄存器的长度均为32位；其中X0是一个特殊的寄存器，只用于全零（注意，在任意的指令中，如果使用X0作为目标寄存器，那么将没有任何效果，仍然保持0不变）
- **所有的指令都是32位字长**，有六种指令格式：寄存器型、立即数型、存储型、分支指令、跳转指令和大立即数。
- RISC-V的设计策略：在每条指令中，让**寄存器的位置保持不变**，如果有立即数，**让立即数的符号位处于指令的最高位。**

2.1.1算术、逻辑、移位指令

- 操作码最后一个字符为“i”的，会将第二个操作数认为是一个**立即数**，此时立即数是直接嵌在指令编码中的。
- RISC-V忽略溢出问题，其高位被截断，低位写入到目的寄存器中
- **逻辑右移**：在最高位添加0；**算术右移**：在最高位添加符号位

2.1.2数据传输指令（访存指令）

数据传输指令在寄存器和内存之间传输数据。而所有其他的RISC-V指令操作（即除load、store之外的指令）只会在寄存器之间进行操作。

格式：`memop reg, off(bAddr)`，`reg`为寄存器的名字，可以为源寄存器或者目标寄存器，访问的内存地址为`bAddr+off`。

- 传输一个字节数据时，可以使用字类型指令，并配合位的掩码来实现；或是使用字节传输指令`lb`与`sb`。

  > 大端机器：高位字节存储在低地址；
  >
  > 小端机器：低位字节存储在低地址。

- `store`指令中，如果操作位数不足32位，高位会被忽略；`load`指令中，如果操作位数不足32位，高位会做符号扩展。
- 字节传输指令`lb, sb`；半字传输指令`lh, sh`

2.1.3分支与跳转指令

- 比较指令
  1. `slt`（Set Less Than）`slt dst, reg1, reg2`

     如果`reg1`中存储的值小于`reg2`中存储的值，就将`dst`设置为1，否则设置为0。`sltu`用来处理无符号数的比较。
  2. `slti`（Set Less Than Immediate）`slti dst, reg1, imm`
- 条件跳转指令
  1. 通过标记`label`的方式来定义语句块起始（类似于C语言中的`goto`）
  2. `beq`（Branch If Equal）`beq reg1, reg2, label`

     如果`reg1`中的值等于`reg2`，跳转到`label`。

     `bne`（Branch If Not Equal）

     > 注意，这里`label`实际上是相对于PC的距离。MIPs中相对距离的计算依据的是下一条指令的PC；而RiscV中依据的就是本条跳转指令的PC。
- 无条件跳转指令`jal`，`jalr`，其将返回地址放到寄存器`ra`，然后跳转到指定地址。

2.1.4伪指令

伪指令可以给程序员更加直观的指令，而不是直接通过硬件来实现，之后再通过汇编器翻译为实际的一条或多条硬件指令。

例如`move dst, src`。然而并没有实际的数据移动指令，这条指令被翻译为下面的指令`addi dst, src, 0`或是`add dst, src, x0`。

2.1.5函数调用

- 过程：
  1. 将参数放置在函数可以访问到的地方
  2. 将控制流转到函数中
  3. 函数获取任何其所需要的存储资源
  4. 执行函数体，完成功能
  5. 函数放置返回值，清理函数调用信息
  6. 控制流返回给函数调用者
- `x1`为返回地址寄存器，用于返回到起始点。传递参数的时候，**顺序是有用的**，代表了程序中参数的顺序。如果寄存器空间不够存储参数，则需要借助于在内存中的栈。
- RISC-V中的函数调用

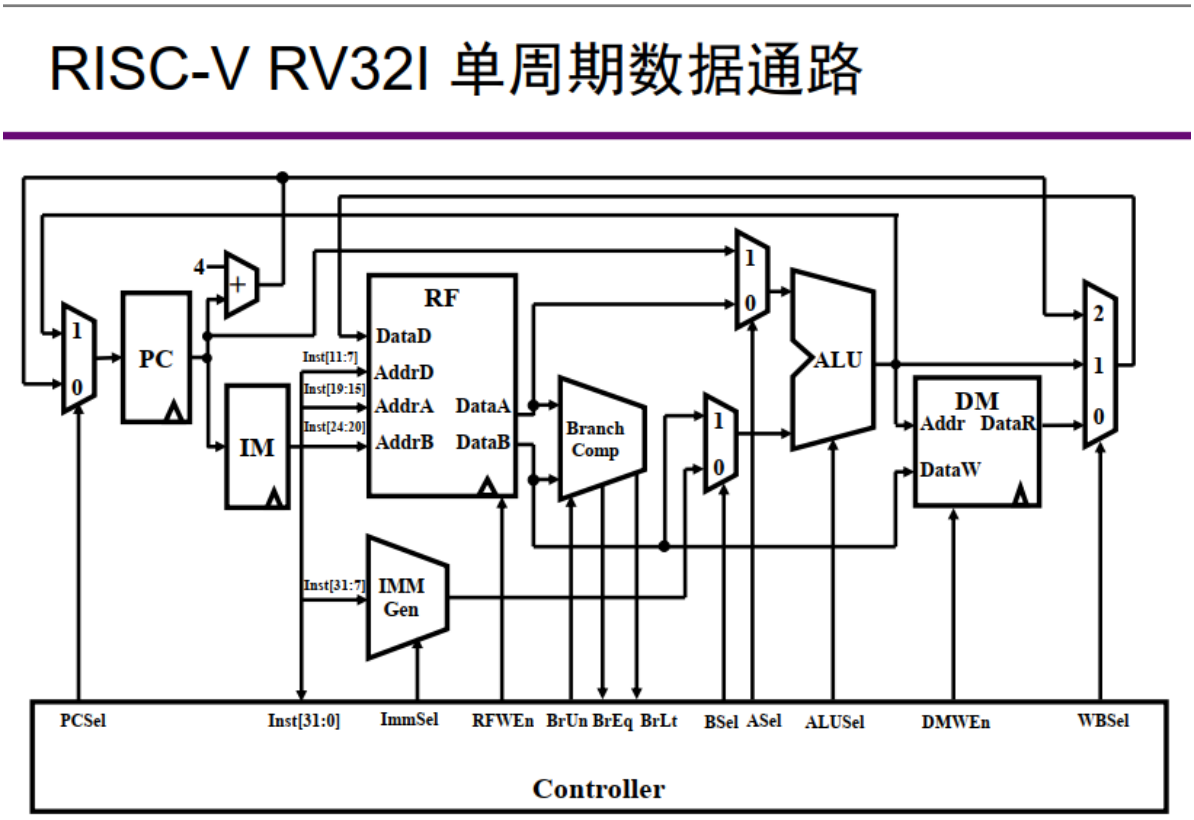
  `jal label`：跳转到某个`label`对应的地址。`jal`表示`Jump and Link`，`link`的意思指在调到对应函数内部之前，将下一条指令的地址放置在返回地址寄存器`x1`中。

  指令和数据都存放在同一个地址空间中，标记`label`会被翻译为一个指令地址。

  `jalr src`：当`label`的位数不足以存放跳转地址对应的立即数时，就可以考虑利用寄存器来读出这个跳转地址。
- RISC-V中的寄存器
  1. `x10-x17`用以传递参数和返回值
  2. `x1`：`ra`为返回地址寄存器
  3. `x2`：`sp`为栈指针
  4. 寄存器惯例：调用者保存的寄存器在函数调用前后可能会被改变；被调用者保存的寄存器在调用前后不会被改变。

- **程序计数器**（PC）：指向当前正在执行的指令的地址，其值在指令执行的第一个阶段就会被更新，通常被更新为下一条指令的地址PC+4。所有的分支指令、跳转指令都是通过更新PC来完成功能。

2.2控制器数据通路设计



2.2.1指令执行过程

1. 读取指令（IF）：从存储器读来指令并形成下条指令地址
2. 指令译码（ID）：译码，并读寄存器堆为ALU准备数据
3. 执行运算（EXE）：ALU执行数据运算或计算存储器地址
4. 存储器读写（MEM）：完成存储器的读操作或者写操作
5. 写回寄存器组（WB）：写ALU的结果或存储器读出数据到寄存器堆

冯·诺伊曼结构的计算机：从主存储器读来一条指令，分析指令，按指令的功能要求完成执行过程，本条指令完成后自动开始下一跳指令的执行过程，由硬件本身完成。

2.2.2单周期CPU

指令周期：计算机一条指令的执行时间

CPU周期：一个CPU时钟时间

CPI：执行每条指令平均使用的CPU周期个数

全部指令都用**一个CPU周期**完成的系统被成为**单周期CPU**，也即指令全部串行执行，前一条指令结束后才启动下条指令，每条指令都用5个步骤时间完成，**控制各部件运行的信号在整个指令周期不变化。**

2.2.3多周期CPU

依据不同指令各自的功能需求为其选择不等的执行步骤，**控制各部件运行的控制信号随着指令执行步骤改变。**相邻指令可以完全串行执行，或部分时间重叠。

2.2.4指令流水线CPU

全部指令都是选用5个步骤完成，执行时间相同，但相邻指令的执行时间有所重叠，每结束一个执行步骤就启动下一条指令。**尖峰速度每个CPU时钟执行一条指令。**

2.3多周期CPU的设计

将指令执行过程分解为多个步骤，每个步骤占用一个时钟周期，同时尽量平衡各步骤的延迟。需要注意的是，多周期CPU中要引入"新"的内部寄存器，在某一步骤结束后保存好下一步骤要用到的值。

两种不同类型的控制器：

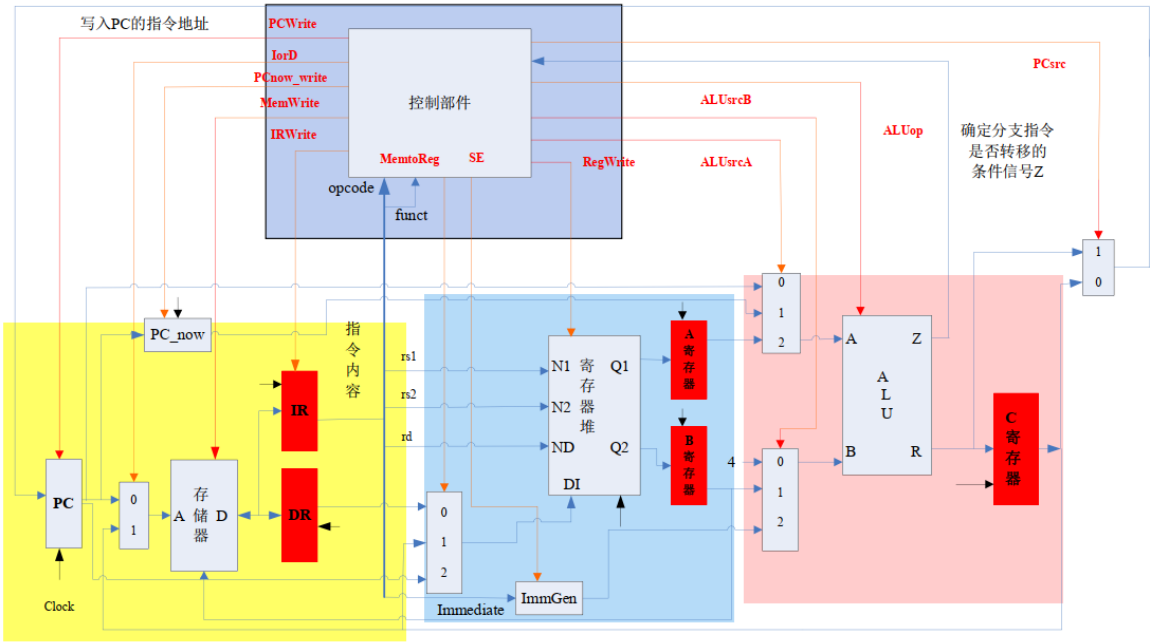
1. **硬连线控制器（组合逻辑控制器）**：采用组合逻辑线路、依据指令及其执行步骤直接产生控制信号

   硬连线控制器由**程序计数器PC**、**指令寄存器IR**、**节拍发生器Timer**和**控制信号产生部件**4部分组成。

   - PC用于提供待读出指令在主存储器中的地址
   - IR用于保存从主存储器中读出的指令内容
   - Timer用于给出并维护指令执行步骤的编码（**状态机**）
   - 控制信号产生部件用于依据指令内容和指令执行所处的操作步骤，用组合逻辑线路产生计算机本操作步骤中各个部件所需要的控制信号
2. **微程序控制器**：采用存储器电路把控制信号存储起来，依据指令执行的步骤读出要用到的信号组合

多周期CPU的数据通路与硬件系统组成：

多周期计算机硬件系统组成



3.层次存储系统

3.0层次存储器系统概述

存储器是计算机中用来存放程序和数据的部件，是Von Neumann结构计算机的重要组成。

用来进行存储的介质需要有两个稳定状态来表示二进制的“0”和“1”，并且两个状态可以方便地进行转换。几种常用的存储方式包括磁颗粒、半导体（电平/电容）、光

大容量存储器速度慢，而快速存储器容量小，为了实现存储器大容量、高速度、低成本、高可靠性的目的，**层次存储器系统**诞生了。

- 静态存储器速度高，可设置较小容量的高速缓冲存储器
- 动态存储器价格适中，速度适中，可作为主存储器
- 磁盘存储器价格低廉，可作为辅助存储器，暂存CPU访问频率不高的数据的程序

层次存储器利用程序的局部性原理，**以最低廉的价格提供尽可能大的存储空间，以最快速的技术实现高速存储访问。**

> **程序的局部性原理**：
>
> 1. 在一小段时间内，最近被访问过的程序和数据很可能再次被访问
> 2. 在空间上这些被访问的程序和数据往往集中在一小片存储区
> 3. 在访问顺序上，指令顺序执行比转移执行的可能性大

主存储器通过地址数据控制三类总线与CPU、其他部件连通

- **地址总线**用于选择主存储器的一个存储单元，**其位数决定了能够访问的存储单元的最大数目，称为最大可寻址空间。**
- **数据总线**用于在计算机各功能部件之间传送数据。数据总线的位数与总线时钟频率的乘积，与该总线所支持的最高数据吞吐能力成正比。
- **控制总线**用于指明总线的工作周期类型和本次入、出完成的时刻。

3.1动态存储器（DRAM）

动态存储器，使用**金属氧化物半导体**（**MOS**）的单个MOS管来存储一个二进制位信息。信息被存储在MOS管T的源极的**寄生电容CS**中，例如，用CS中存储有电荷表示1，无电荷表示0。

特性：**集成度高、容量大、能耗低、速度慢**

动态存储器的工作特点：

1. **破坏性读出**：读出时会被强制清零，读后需要将读出的数据再重新写回去，这一过程被称为**预充电延迟**
2. **需定期刷新**：在不进行读写操作时，DRAM存储器的各单元处于断路状态，由于**漏电**的存在，保存在电容C_S上的电荷会慢慢地漏掉，因此需要定时予以补充（即**刷新**）。

   刷新有两种常用方式，分别是**集中刷新**（停止内存读写操作，逐行将所有各行刷新一遍）；**分散刷新**（每次内存读写后，刷新一行，各行轮流进行）。

3.2静态存储器（SRAM）

利用D触发器来存储信息，非破坏性读出，没有放电的操作因此不需要刷新。

特性：**速度快、存储密度低、数据入/出共用管脚、能耗高、成本高**

3.3高速缓冲存储器Cache

设置于主存和CPU之间的存储器，用高速的静态存储器实现，缓存了CPU频繁访问的信息。

特性：高速（与CPU的运行速度基本匹配）；透明（完全硬件管理，对程序员透明）

相关概念

1. **块**：数据交换的最小单位 一般为4~128Bytes
2. **命中**（Hit）：在较高层次中发现要访问的内容

   命中率（Hit Rate，HR）：命中次数/访问次数

   命中时间：访问在较高层次中数据的时间
3. **缺失**（Miss）：需要在较低层次中访问块

   缺失损失：替换较高层次数据块的时间+将该块交付给处理器的时间

   命中时间<<缺失损失

3.3.1Cache的读取

- **全相联方式**

  有效位（1位）+标志位（与块的总个数相关）+数据

  > 缺点：标志位较长，如果主存空间有2^m块，则标志位要有m位。同时，如果Cache有n行，则需要有n个比较电路。

- **直接映射方式**

  主存的字块只可以和固定的Cache字块对应。标志位较短，比较电路的成本较低。如果主存空间有2^m块，Cache中字块有2^c块，则标志位只要有m-c位，且仅需要比较一次。全相联方式中标志位的最后c位被直接映射到Cache中的块的编号，这样节省了标志位的位数以及比较器的个数。

  > 缺点：利用率低，命中率低（可能会在Cache没有填满的情况下多次替换），效率较低

- **多路组相联**

  结合全相联与直接映射两种方式的优点，每个内存地址可以对应Cache中的多个位置。在映射失效时，如果需要腾出Cache空间，需要选择替换出哪一个Cache行。

3.3.2Cache的写操作

- **写直达**（Write Through）

  在Cache中命中时，同时修改Cache和对应的主存内容

  没有在Cache中命中时，存在写分配以及非写分配两种手段，前者写入主存之前，在Cache中给其分配位置，同时写入Cache中。
- **拖后写**（Write Back）

  弱一致性保证，在替换时再写主存。CPU通过监听总线上的访问操作来实现。

3.3.3提高存储访问的性能

平均访问时间=命中时间×命中率+缺失损失×缺失率

块大小的权衡：一般来说，数据块较大可以更好地利用空间局部性；但是数据块大意味着缺失损失的增大，因为需要花费更长的时间来装入数据块，且块大小相对Cache总容量来说太大的话，命中率将降低，因为Cache的块数相对太少。

Cache替换策略：

1. 最近最少使用LRU：有较高命中率，但硬件实现复杂
2. 先进先出FIFO：满足时间局部性，实现比较简单
3. 随即替换RAND：实现简单，命中率也不太低

3.4虚拟内存

3.5外存储器

3.5.1磁芯存储器

圆柱形陶瓷上涂磁粉，存储原理简单，工艺复杂，可靠性低，容量大，成本低，断电后还可以保存信息

通过磁颗粒的不同偏转方向来区分不同的状态

磁记录方式：指一种编码方式，将一串二进制信息，通过读写电路变换成磁层介质中的磁化翻转序列

1. 归零制（RZ）：正脉冲为1，负脉冲为0，两位信息位之间线圈电流为零
2. 不归零制（NRZ）：线圈中一直有正或负脉冲
    - 见1翻转的不归零制：只有见到1才改变电流方向
3. 调相制（PM）：用脉冲的边沿来表示0和1
4. 调频制（FM）：1：位周期中心和位与位之间都翻转；0：位周期中心不翻转，位与位之间翻转
5. 改进的调频制（MFM）：只有连续两个或以上的0时，才在位周期的起始位置翻转

3.5.2磁盘

特点：容量大、价格低廉、速度慢

使用旋转托盘上的表面磁颗粒来存储数据，使用可移动的读/写头来访问磁盘。

- 磁盘访问过程
    - **寻道**：将读写磁头移动到正确的磁道上（平均需要8~20ms）

      > **硬磁盘参数**
      >
      > 磁盘每面包含500~2000个**磁道**，每个磁道包含32~128个**扇区**；**扇区是磁盘访问的最小单位**（通常包含1KB数据）
      >
      > 为保证位密度不变，外磁道比内磁道的扇区数要多一些
      >
      > **柱面**：位于同一半径的磁道集合

    - **寻找扇区**：等待磁盘旋转到需要访问的扇区（旋转速度为3600~7200RPM，转每分钟，平均寻址时间为8ms到4ms）
    - **数据传输**：读写数据（1个或多个扇区）（2~15MB/sec)

  磁盘访问时间=寻道时间+旋转延迟+传输时间+磁盘控制器延迟（磁盘进行各种动作所带来的延迟）

  旋转延迟，或平均旋转延迟为**磁盘旋转半周的时间**

  注意：

  1. 额外开销在总开销中占比较大——一次传输大量数据比较有效
  2. 将页面存放在相邻扇区可以避免额外的寻道开销
  3. **对磁盘的访问总是由缺页引起的。**如果操作系统检查页表，发现该页不在内存中，需要从磁盘调入
  4. 磁盘每次写入数据的间隙，会计算校验码并写入扇区中

3.5.3RAID

RAID：廉价磁盘的冗余阵列（Redundant Arrays of Inexpensive Disks）

N个磁盘的容量，1/N的访问时间，采用冗余技术提高存储信息的可用性

- **RAID0**：将由RAID模拟的单个虚拟磁盘划分成**带**，每带k个扇区

这种在多个驱动器上分布数据的方式叫做**分带**。如果软件发出从带的边界开始读四个连续带的数据块的命令，RAID控制器将把这个命令分解成四个单独的读命令，四个驱动器每个一个，让它们并行执行。

> 没有冗余，可靠性差，并不能算真正的RAID

- **RAID1**：复制了所有磁盘，有四块主磁盘和四块辅助磁盘。每个对磁盘的写操作都要进行两次，而每次读磁盘则可以读任意一个备份，从而**把负载均衡分布到不同的驱动器上**，使得读磁盘的性能提高了两倍，容错性能也更好
- **RAID2**：7个驱动器，后三个驱动器作为前四个驱动器的海明码校验值，每个驱动器读取或写入的单位是1bit。要求驱动器必须**同步旋转**，同时驱动器个数要足够多
- **RAID3**：RAID2的一个简化版本，对每个字计算一个校验位，写到一个校验驱动器上。同样，驱动器之间必须**严格同步**。
- **RAID4**：和RAID0类似，不过将对**带**的校验位写在额外的驱动器上。例如，若带的长度为k个字节，将所有带异或到一起，产生一个k字节长的校验带。如果其中一块磁盘崩溃，它的内容可以从校验磁盘上重新计算出来。

  不需要驱动器同步，但纠错性能较差，且校验盘负载较重
- **RAID5**：为减少校验盘的负载，将校验位循环均匀分布到所有驱动器上
- **RAID6**：为防止两个磁盘同时崩溃，采用二维校验，即采用两种校验方法，得到两个校验值，分别采用循环均匀分布的方式存储到不同的磁盘上

3.5.4固态硬盘（SSD）

固态硬盘没有机械结构，没有可以移动的部分。其功耗低，性能高，但价格较高，**擦除次数有限。**随着擦除次数的增加，存储单元不能可靠地保持状态。

固态硬盘是用固态电子存储芯片阵列制成的硬盘，由**控制单元**和存储单元组成。

SSD主要由SSD控制器、FLASH存储阵列，板上DRAM，以及跟HOST接口组成。包括三个重要组成部分：**主存芯片**、**闪存芯片**、**固件算法**

存储介质的组织：package-die-plane-block-page，block是最小的擦除单位，页是最小的读写单位

- SSD的写入：不会写入到原来的page，写入之间需要进行擦除操作。由FTL来维护上层管理软件的逻辑地址和底层的物理地址之间的映射
- SSD的地址转换FTL（Flash Translation Layer）

  除了做地址转换，FTL还帮助完成**磨损均衡**：写入的时候需要挑选位于擦除次数最少的块中的页面，完成磨损均衡。
- SSD上的垃圾搜集：搜寻已经不再需要的页面，将其转移到一个block中，然后将其擦除
- SSD读数据的延迟大约在20~100us

  SSD写数据的延迟大约在200us

3.4 Riscv系统模式

- **M模式下的异常处理**

  8个控制状态寄存器（CSR）

  1. mtvec：发生异常时处理器需要跳转到的地址
  2. mepc：指向发生异常的指令；对于同步异常，mepc指向导致异常的指令；对于中断，它应指向中断处理后应该恢复执行的位置（这一行为由**软件**来处理）。
  3. mcause：发生异常的指令
  4. mie：处理器目前能处理和必须忽略的中断
  5. mip：正准备处理的中断

6. mtval：trap中的附加信息：例如地址异常中出错的地址、非法指令、异常的指令等
7. mscratch：暂时存放一个字的大小的数据
8. mstatus：机器的状态；处理器在M模式下运行时，只有在全局中断使能mstatus.MIE置为1的时候才会产生中断

4.2总线

计算机总线：共享的信息通道，用于连接计算机多个子系统

- 单总线计算机：简单、成本低，但速度慢，总线将成为系统瓶颈
- 双总线系统：分开主存与外设

  输入、输出总线通过**适配器**和处理器-主存总线相连
- 三总线系统

  三条总线：内存总线、主板总线，I/O总线

  内存总线连接主存和适配器（北桥芯片），主板芯片连接北桥芯片和南桥芯片，I/O总线连在南桥芯片上。

  大大减少了处理器-主存总线的负载，是现代PC基本采用的结构

总线事务：

包括两个部分：发起命令（和地址），传输数据

主设备是总线事务的发起者，从设备是总线事务的响应者。

总线仲裁器：优先权和公平性的平衡（优先级高的设备应该得到优先服务，最低优先级的设备也不能永远被排除在总线服务之外）

总线仲裁方式：集中仲裁（存在一个总的仲裁器）和分布仲裁（确认没有其他请求后再发起请求）

- **菊链仲裁**：**所有设备共用一个总线信号**

  总线授权信号逐级传递，缺点在于无法保证公平性，低优先级设备可能得不到总线使用权
- **集中平行仲裁**：**通过集中的仲裁器进行**

  所有设备都平行地将请求发给仲裁器，由仲裁器决定谁能向总线发起请求
- **同步总线**：控制线中包含有一根时钟信号线，总线上所有信号必须按照时钟频率工作；同时为了防止时钟扭曲，高速工作时，**总线距离必须足够短。**

  所有设备以同样的速度工作，协议简单
- **异步总线**：不使用统一时钟，使用**握手协议**，可适应设备的不同速度

增加总线带宽的方式：

1. 增加总线的宽度
2. 分别设置数据总线和地址总线
3. 采用成组传送方式

多主设备总线提高事务数量：

1. 仲裁重叠：在当前事务时，为下一总线事务进行仲裁
2. 总线占用：在没有其他主设备请求总线的情况下，某主设备一直占用总线，完成多个总线事务
3. 地址、数据传送重叠

DMA(Direct Memory Access)使用总线的方式

1. **独占使用**：当外设要求传送一批数据时，由DMA控制器发一个信号给CPU。DMA控制器获得总线控制权后，开始进行数据传输。一批数据传送完毕后，DMA控制器通知CPU可以使用内存，并把总线控制权交还给CPU

2. **周期挪用**：当I/O设备没有DMA请求时，CPU按程序要求访问内存；一旦I/O设备有DMA请求，则I/O设备挪用一个或几个周期
3. **DMA与CPU交替访问**：一个CPU周期分为2个周期，一个专供DMA控制器访存，另一个专供CPU访存，不需要总线使用权的申请、建立和归还过程。

4.3接口电路和外部设备

- **接口的基本功能**：
    1. 识别I/O设备
    2. 建立主机和设备之间的控制羽通信机制
    3. 提供主机和设备之间信息交换过程中的数据缓冲机制以及其他需求，包括屏蔽外部设备的差异

    > 串行接口芯片8251A：全双工，可通过指令指定接口的功能和运行控制参数，例如工作方式为同步还是异步，波特率因子为多少

- **USB接口**

    半双工模式，由四根线组成，包括电源、地和双数据线（虽然有双数据线，但两条线上实际上传递的是相同的信号）。

    USB接口工作原理：

    1. USB结构：根HUB、层次结构
    2. 设备检测：根HUB定时查询接口状态，若检测到有设备接入到接口上，则**为该设备赋地址（7位）**。设备初始地址为0，每个设备上应有ROM，保存设备参数；识别设备类型后，由设备驱动程序管理和使用设备
    3. 只有1个主设备，不需要仲裁，采用**轮询**方式
    4. **USB协议**：
        1. 每1ms，发出一个SOF包，进行时间同步

            > 协议包包括：**令牌包**（SOF，IN，OUT，SETUP）、**数据包**（Data）、**握手包**（ACK、NAK、STALL）、特别包

        2. 第1帧：根发出读命令（IN），包含有**地址（7位）**；设备返回数据包DATA，其中包括SYN、PID、载荷和16位校验码；ACK为根收到数据后返回给设备的确认包。（往设备写数据时类似）

- **外部设备**

    与接口进行连接，与主机进行通信，完成人机交互。

    - 键盘：把每个键在键盘上的位置对应为一个编码，还需要解决键的抖动、多个键同时被按下等问题。
    - 鼠标：

        机械式鼠标：鼠标内部有一个橡胶球，橡胶球紧贴着两个互相垂直的轴（X、Y轴），每个轴上有一个光栅轮，光栅轮两边对应着有发光二极管和光敏三极管。光敏三极管在接受发光二极管发出的光时被光栅轮间断地阻挡，从而产生脉冲信号，通过鼠标内部的芯片处理之后被CPU接受。
    - 阴极射线管（CRT）显示器

        成像原理：通过电子束撞击荧光板上的荧光粉，发光产生亮点。有光栅扫描（电子束从左到右，从上到下扫描整个屏幕）或随机扫描（只扫描需要显示的点）
    - 液晶显示器：通过液晶改变透射光的偏振性
    - 等离子显示器：利用惰性气体在一定电压作用下产生气体放电的特性，产生紫外线，用紫外线激发荧光粉发光。

通过在玻璃板之间隔开像素，每个像素点内有惰性气体和三色荧光粉，用电极进行控制。