期末复习要点：

1. 题型及分数分布

\* 客观题（判断题（10分），单选题（10分），多选题（10分），填空题（10分））。

\* 主观题（简答题（10分），计算题（15分），理解/设计题（15分），综合题（20分））。

2. 复习重点：

\* 简答题：共出2道题，每题5分，下面4题作为复习参考题目：

# • 试分别举例说明程序和数据的局部性（包括空间局部性和时间局部性）。

## 空间局部性（Spatial Locality）：

空间局部性是指程序在访问存储器时，倾向于访问那些与最近访问过的数据空间上相邻的数据。

1. **数组处理**：在C语言中，对数组的访问通常具有高度的空间局部性。例如，以下代码片段：

c

int arr[1000];

for (int i = 0; i < 1000; i++) {

printf("%d ", arr[i]);

}

在这个例子中，程序会连续访问数组arr中的元素，这些元素在内存中是连续存储的。

1. **循环结构**：循环内部的代码通常会重复访问相同的变量或数据结构。例如：

c

int sum = 0;

for (int i = 0; i < 1000; i++) {

sum += i;

}

在这里，变量sum和i在循环的每次迭代中都会被访问，它们在内存中的位置是固定的。

## 时间局部性（Temporal Locality）：

时间局部性是指程序在访问存储器时，倾向于重复访问最近访问过的数据。

1. **循环结构**：循环中的指令和数据会被重复执行和访问。继续上面的例子，变量sum和i在循环的每次迭代中都会被访问，体现了时间局部性。
2. **函数调用**：在递归函数中，函数会重复调用自身。例如：

c

int factorial(int n) {

if (n <= 1) return 1;

return n \* factorial(n - 1);

}

在这个递归函数中，每次调用factorial都会重复访问相同的代码和数据。

1. **条件语句**：在包含条件语句的循环中，某些代码路径可能会被频繁执行。例如：

c

for (int i = 0; i < 1000; i++) {

if (i % 2 == 0) {

printf("Even number: %d\n", i);

}

}

在这个例子中，如果i是偶数，那么打印语句会被执行，这在循环中是频繁发生的。

1. **数据缓存**：操作系统或应用程序可能会将频繁访问的数据缓存起来，以便快速访问。例如，浏览器缓存网页内容，使得用户再次访问相同网页时可以更快地加载。

# • 针对复杂指令集(CISC)和精简指令集(RISC)计算机，试分别出列举它们相对立的5个特点。\*

## CISC特点：

1. **指令功能强大**：**指令集功能强大复杂庞大**，可以直接执行复杂操作，减少了程序的指令数量。
2. **//向后兼容性好**：由于CISC处理器支持丰富的指令集，能够运行大量历史遗留的软件和应用程序。
3. **指令长度不固定**：CISC的指令长度不固定，
4. 指令格式种类多，
5. 寻址方式种类多。
6. **存储器访问不受限制**：可以访存的指令不受限制，而RISC只有取数/存数指令访问存储器。
7. **微程序控制**

## RISC特点：

1. **指令集简单**：RISC**指令集简单**，指令数量少，
2. 每条指令执行时间固定，大部分在一个时钟周期内完成。
3. **指令长度固定**：RISC的指令长度固定，
4. 指令格式种类少，寻址方式种类少。
5. **存储器访问限制：**只有Load/Store指令访存，其余指令的操作都在寄存器之间进行。
6. **硬布线控制**
7. **流水线技术**：RISC一定采用流水线技术，大部分指令在一个时钟周期内完成。
8. **多通用寄存器**：CPU中有多个通用寄存器，比CISC的多。



# • 简述精简指令集计算机(RISC)的概念及主要特点。

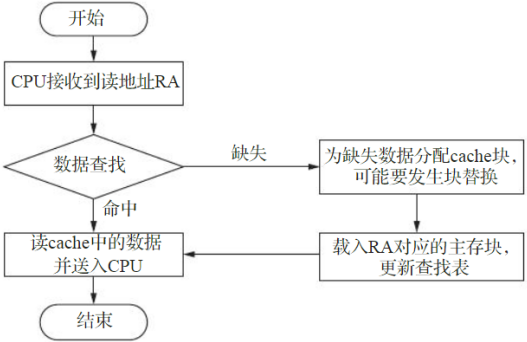
精简指令集计算机（Reduced Instruction Set Computer，简称RISC）是一种计算机架构设计理念，其核心思想是简化处理器的指令集，减少每条指令的复杂性，以提高处理器的性能和效率。

RISC架构的主要特点如下：

1. **指令集的精简**：RISC优先选取使用频率最高的简单指令和一些有用但不复杂的指令，避免使用复杂指令，以提高指令执行的效率。
2. **效率高**：大多数RISC指令在一个时钟周期内完成，这有助于简化处理器的设计，并提高指令的吞吐率。
3. **LOAD/STORE结构**：RISC采用LOAD/STORE结构，减少对主存的访问，只允许LOAD（取数）和STORE（存数）两种指令访问主存，其余指令只能对寄存器操作数进行处理，以减少访问主存的时间消耗。
4. **简单的指令格式和寻址方式**：RISC采用简单的指令格式和寻址方式，指令长度固定，这有助于简化指令的译码逻辑。
5. **固定的指令格式**：RISC的指令长度和格式固定，这可以简化指令的译码逻辑，提高流水线的执行效率，并且便于编译器优化。
6. **面向寄存器的结构**：为了减少对主存的访问，RISC处理器内设有大量的通用寄存器，这样可以减少数据在处理器和主存之间的传输时间。
7. **硬布线控制逻辑**：由于指令系统的精简，RISC的控制部件可以由组合逻辑实现，减少或不使用微程序控制，从而提高控制部件的速度。
8. **编译优化**：RISC架构注重编译器的优化，力求有效地支持高级语言程序，使得高级语言编写的程序能够更高效地转换为RISC处理器可以执行的机器代码。

RISC架构的设计理念是为了提高处理器的性能和效率，通过简化指令集和优化硬件设计来实现。这种架构在现代计算机系统中非常流行，尤其是在高性能计算和移动设备领域。

# • 试简要概述cache的读流程。

****

1. **CPU接收到读地址RA：**CPU接收到一个需要读取的内存地址RA（Read Address）。
2. **数据查找：**CPU在cache中查找与地址RA对应的数据。
3. **判断命中或缺失：**
   * **命中：**如果数据在cache中找到（命中），则流程进入下一步。
   * **缺失：**如果数据不在cache中（缺失），则需要进行缺失处理。
4. **命中处理：**
   * 从cache中读取对应的数据。
   * 将数据送入CPU。
5. **缺失处理：**
   * 为缺失数据分配cache块，这可能涉及到块替换，//根据替换算法选择一个cache行进行替换。
   * 从主存中载入RA对应的主存块到cache的指定位置。
   * 更新查找表，以便下次访问时能够快速找到该数据。
   * 返回数据给CPU：将请求的数据返回给CPU，以完成读操作。
6. **结束：完成数据的读取后，流程结束。**

OR:

1. **CPU发起读请求**：当CPU需要读取数据时，它首先生成一个包含主存地址的请求，并将其发送给cache。
2. **地址分解** ：CPU发送的地址被分解为主存块地址（tag）和块内偏移（offset）两部分。块内偏移用于在cache数据块内定位具体的数据，而主存块地址用于在cache中查找对应的数据块。
3. **查找cache**：cache使用主存块地址（tag）来检查是否存储有请求的数据块。这一步骤涉及在cache中查找与请求地址相匹配的数据。
4. **判断命中或缺失**：
   * **命中（Hit）**：如果cache中存在请求的数据（即数据命中），cache会根据块内偏移（offset）提供相应的数据给CPU。
   * **缺失（Miss）**：如果cache中不存在请求的数据（即数据缺失），则需要从主存中加载数据块到cache中。这通常涉及到替换cache中已有的数据块，特别是当cache已满时。
5. **数据传送**：
   * 在命中的情况下，cache直接将数据传送给CPU。
   * 在缺失的情况下，cache从主存中读取数据块，并更新cache中的内容，然后将请求的数据传送给CPU。
6. **更新cache**：在数据缺失的情况下，除了传送数据给CPU外，cache还需要更新其内容，可能包括替换策略的实施和脏位的更新。
7. **返回数据给CPU**：无论是命中还是缺失后的处理，最终cache都会将请求的数据返回给CPU，以完成读操作。

这个流程体现了cache在提高数据访问速度和减少CPU等待时间方面的关键作用，同时也展示了cache如何处理数据存储和替换的复杂性

# \* 其他（在掌握全局知识结构的前提下，指定的教材页为复习重点）：

• 4，9，11-12，14，19-24，30， 43-44，60-62，83-85，97-100，117，119-120，122，126-130，132-136，140，153，159-160，163，167-169，194，196-199，221-222，225-227，229-233，235-240，241-242，289，322。