名词解析

- SIMD: (Single Instruction Multipe Data, 单指令流多数据流并行机,P3) 是一种并行机体系结构，用同一控制器同时控制多个数据单元流动，来提高空间上的并行性。
- SMP：(Symmetric Multiprocessor, 对称多处理机, P4,P215)是一种共享存储的可扩放并行计算机，通过高速总线连向共享存储器。
- UMA：(Uniform Memory Access,均匀存储访问模型,P36)是并行计算机的一种主要访存模型，物理存储器被所有存储器均匀共享，所有处理器访问任何存储字取相同的时间。
- DSM：(Distributed Shared Memory, 分布式共享存储多处理机,P4,P359)是一种在物理上分布存储的系统中，逻辑地实现共享存储的并行机模型。
- MTTF：(Mean Time To Fail,平均无故障时间, P107,P302)是指系统发生故障前平均正常运行的时间，是一种系统可靠性的表示方式。
    - MTTR(Mean time To Repalr, 平均修复时间)指系统失效后修理恢复正常的工作时间。
    - 可用性计算公式： $Availability = MTTF/(MTTF + MTTR)$
- TFLOPS：(Trillion Floating Points Operation Per Second,P98)表示每秒万亿次浮点运算，常用来评价向量计算机的性能。
- SSI：(Single System Image,单一系统映像,P108,P306)是机群的一个重要特征，可以使机群在使用、控制和维护上更像一个工作站。
- SAF：(Service Availability Forum, 服务可用性论坛，)一个致力于定义一组用于电信设备和其他商业设备管理公共接口的组织。主要成果有HPI、AIS等。
- MIN：(Multistage Interconnection Network, 多级互联网络, P163)由单级交叉开关级联起来形成的一种互联网络，被用于MIMD和SIMD计算机设计中。
- COW：(Cluster of workstations,工作站机群, P4,P18) 是一种并行机体系结构，将一群工作站或高档微机使用某种结构的互联网络互联起来，充分利用各工作站的资源，统一调度、协调处理，以实现高效并行计算。
- MPP：(Massively Prarallel Processor,大规模并行处理机,P4,P268) 由成百上千甚至近万处理器所组成的大规模并行计算机系统。(异步MIMD处理模式)
- PVP：(Parallel Vector Processor,并行向量处理机，P32) 包含少量的为高性能专门设计的向量处理器VP，使用专门设计的高带宽的交叉开关网络，将VP连接到共享存储器模块中(注意：PVP不使用高速缓存，而使用大量的向量寄存器和指令缓冲器)。
- GFLOPS：(Giga Floating Points Operation Per Second,P98)表示每秒十亿次浮点运算，常用来评价向量计算机的性能。
- Hypercuba：(超立方，P157) 一个n-立方是指包含了$N = 2^n$个顶点，节点和网络直径都是n，对剖析度为$N/2$的高维网络结构。(超立方上有很多优秀算法，且很多低维网络都可以嵌入到超立方中)
- Cut-Through：(CT,切通, P189)并行计算机互联网络中的一种选路方式，切通网络会将信包进一步分成数据片和包头进行传输，代表说虫蚀选路。
    - 选路：就是Routing，消息从源到目的地所选取的走法。

简答题

1.请列举主要的并行计算机访存模型。P37

解：

（1）UMA，均匀存储访问模型。所有处理器访问任何存储字取相同时间。

（2）NUMA，非均匀存储访问模型。处理器访问不同存储器的时间不同。

（3）COMA，全高速缓存存储访问模型。是NUMA的一种特例，各处理器节点没有存储层次结构，全部高速缓存组成了全局地址空间。

（4）CC-NUMA，高速缓存一致性非均匀存储访问模型。实际上是一个分布共享存储的DSM多处理机系统。

（5）NORMA，非远程存储访问模型。所有存储器都是私有的，仅能由其自己的处理器访问。

（6）NCC-NUMA，高速缓存不一致非均匀存储访问模型。

2.请比较Amdahl定律和Gustafson定律。P113

解：二者都是评价加速比性能的定律，Amdahl定律适用于固定计算负载，Gustafson适用可扩放问题。

若我们令P表示处理器数目，S表示加速比，W表示问题规模，其中可并行的部分为W_p，串行的部分W_s，又令f表示串行部分的占比。

(1) Amdahl定律：出发点是在计算负载固定不变的前提下，对其中可并行的部分通过增加处理器的数目来提高计算速度。公式为：

$$S=\frac{W_s+W_p}{W_s+W_p/P}=\frac{P}{1+f(P-1)}$$

它意味着随着处理器数目的无限增大，并行系统所能达到的加速比上限为1/f，此结论在历史上曾对并行系统的发展起到了悲观的作用。

(2) Gustafson定律：出发点是在实际应用中，没有必要固定负载，增加处理器必须相应增大问题规模，才有实际意义。公式为：

$$S=\frac{W_s+PW_p}{W_s+W_p}=P-f(P-1)$$

它意味着随着处理器数目的增多，加速比几乎与处理器数成比例的线性增加，串行比例f不再是程序的瓶颈，这对并行系统的发展是个非常乐观的结论。

二者看似矛盾，实则是在不同问题假设下的统一，Gustafson中的问题规模增大，实际上是相对于Amdahl负载不变来说，增加了并行负载部分的比例，降低了f，因此并不矛盾。

3. 请比较描述可扩放性评价中的几种评价标准。P128

解： 可扩放性标准主要包含：

（1）等效率度量标准。即在保持效率$E = \frac{S}{P}$不变的前提下，研究问题规模W如何随处理器数目P而变化。即为了维持一定效率(介于0和1之间)，需要在P增大时，相应增加问题规模W的值，W增加的小，则可扩放性良好。

（2）等速度度量标准。是在保持处理器平均速度$\overline{v} = \frac{W}{PT}$不变的前提下，研究处理器数目P增多时应该相应增加多少工作量W。对于一个并行算法，当处理器数目增大时，若增大一定工作量能维持平均速度不变，则称该算法是可扩放的。

（3）平均延迟度量标准。是在效率E不变的前提下，用处理器数目及问题规模变化前后的平均延迟之比，来表征算法的可扩放性。对于一个并行算法，比值越大，可扩放性越好。

以上三种评判可扩放性的标准的基本出发点，都是抓住了影响算法可扩放性的基本参数T_o，只是等效率标准采用解析计算的方法得到T_o；等速度标准是将T_o隐含在所测量的执行时间中；而平均延迟标准则是保持效率恒定时，通过调节W与P来测量并行和串行的执行时间，最终通过平均延迟反应T_o。事实上，三种度量可扩放性的标准是彼此等效的。

4.请举例描述并行程序设计中的任务划分方法。P139

解：（1）域分解：也叫数据划分，所要划分的对象为数据，这些数据可以是算法的输入、计算的输出或者中间结果。域分解的步骤为：首先分解与问题相关的数据，在可能的情况下使得这些小的数据片尽可能大致相等；再将每个计算关联到它所操作的数据上。这样划分产生一系列任务，每个任务包括一些数据及其上的操作。

（2）功能分解：也被成为任务分解或者计算划分。首先关注被执行的计算；然后如果所做的计算划分是成功的，再继续研究计算所需的数据，若数据基本不相交则意味着划分成功；若数据具有相当的重叠，意味着必然有大量的通信，应当考虑数据分解。

5.请比较描述SMP中不同的锁机制。P253

解：(1)简单的软件锁算法通过处理器指令集中支持的某种原子指令来实现的。最为典型的就是原子交换指令。例如使用Test&Set指令、Swap指令、Fetch&Op指令等来对一个特定的锁变量进行修改从而实现。现代的微处理器还支持LL和SC从而实现对一个变量的原子IO操作。

(2)简单的锁算法存在性能问题，故有了Test&Set lock with Backoff，降低Test&Set的发出频率，通过在两条Test&Set指令之间插入一个延迟时间进行实现或者通过Test-and-Test&Set实现，在忙等待的时候只使用Load读取锁变量的值，只有在读取到锁释放的时候才使用Test&Set去获取锁。

(3)更加先进的锁算法包括票锁，即每个锁进程持有一个票号，忙等待一个全局的Now-serving，只有Now-serving与自己的票号相同才能得到锁。且没有一个进程释放锁，Now-serveing才会+1。基于数组的锁通过Fetch&Increment指令去获得忙等待的一个位置，当锁释放的时候下一个位置上在忙等待的处理器将会通过读缺失知道自己获得了锁。

(4)全硬件的方式同样可以实现锁。

6.简单比较基于侦听的高速缓存一致性协议和基于目录的高速缓存一致性协议。P228(侦听)P365(目录)

解：(1)侦听一致性协议利用了总线的两个特性：所有总线上的事务对所有的高速缓存控制器都是可见的；它们对所有的控制器都以相同的次序可见。该协议利用了在单机体系结构中已经存在的两个基本因素：总线事务和高速缓存块相关的状态转换图。该协议不适用于大型多处理机系统中，因为在多级互联网络中实习广播的代价很大。

(2)基于目录的高速缓存一致性协议通过维护一个全局目录，每次需要通过一致性命令来维护高速缓存一致性时利用全局目录只把一致性命令发送给特定的节点从而实现一致性协议。该协议可以通过将目录分布到各个节点上防止其成为瓶颈。

7.请回答为何并行检查点操作中会产生多米诺效应？又该如何解决？P305

解：产生多米诺效应的情形：进程P回卷后依赖于进程Q在上一个检查点的消息，从而使得Q必须回卷到上一检查；回卷后的Q又依赖于P的上一个检查点的消息或行为。从而使得P和Q都需要回卷到最初状态，这种情况下检查点毫无作用。其直接原因是检查点无法保存进程执行中的消息状态。

为了解决多米诺效应，独立检查点操作需要增加一个消息日志。其思想是：在检查点操作时间，各个进程不仅分别独立地保存他们的局部检查点，而且还将消息保存一个日志中。

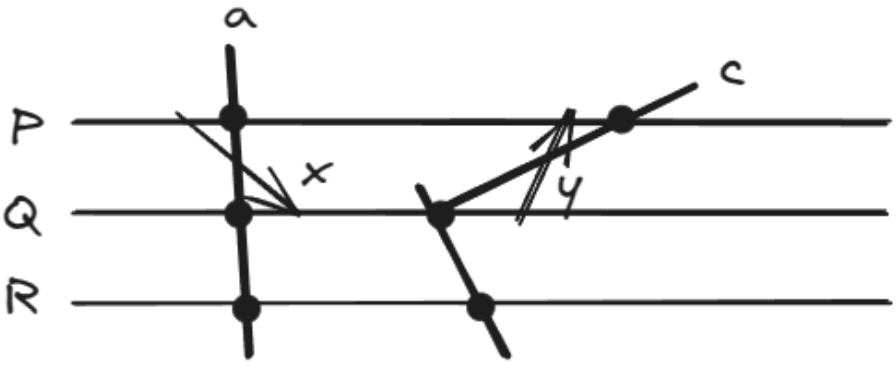
8.请描述基本的故障恢复策略，并举例说明何为非一致的全局检查点。P302 P305

解：基本的故障恢复策略包括向后恢复和向前恢复。

(1) 向后恢复：进程周期性地将一个一致的状态(检查点)保存到稳定的外存中。在发生失效后，系统重配置，以隔离失效组件；恢复至前一个检查点，以继续正常操作，这也称为回卷(Rollback)。

(2) 向前恢复：系统不回卷至前一个检查点，而是利用失效诊断信息重建一个有效的系统状态。

如果进程之间不存在一个进程的检查点已经收到了消息，而另一个进程的检查点还未发出消息的消息传递，则称这些检查点是全局一致的。而非一致的全局检查点显然是违背这一描述的情况，如下图所示：快照a中的检查点是全局一致的，而c中的检查点是非一致的。



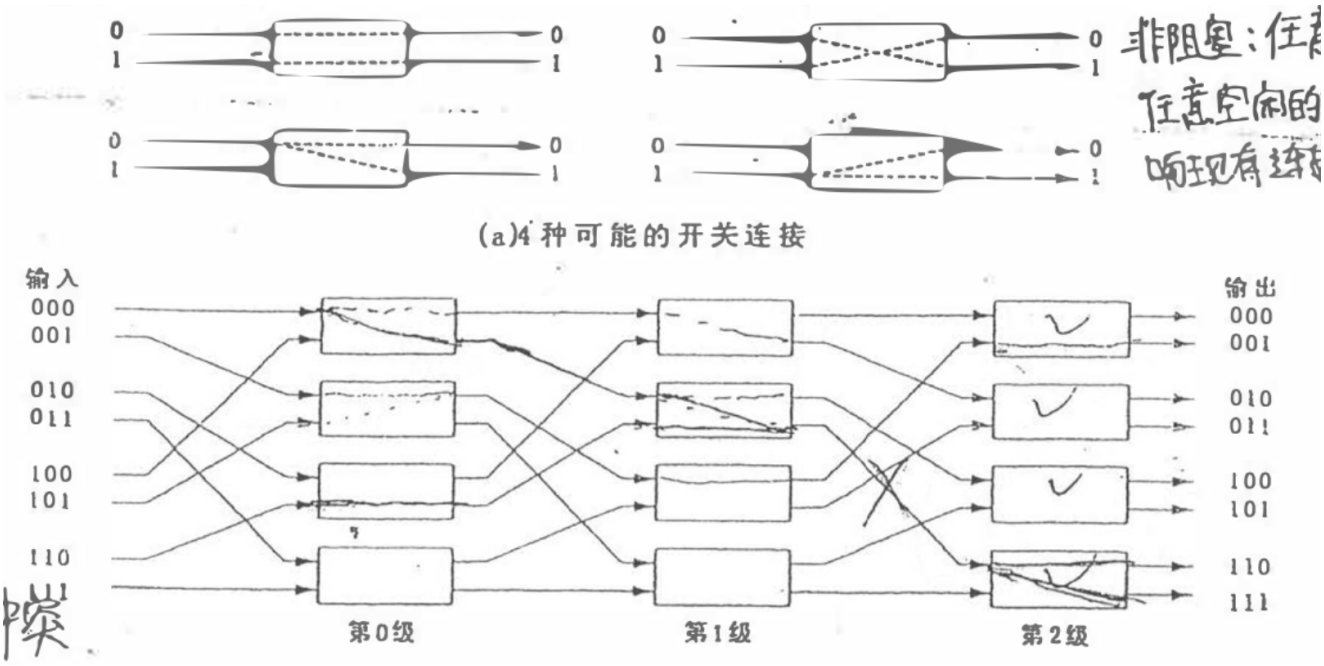
9.请回答何为机群中的单一系统映像以及它主要包括哪些服务？P306

解：单一系统映像是集群的一个重要特征，使用它使得机群在使用、控制、管理和维护上更像一个工作站，它包含了单一系统、单一控制、对称性以及位置透明的多层含义。用户可以把整个集群视为单一系统来使用，逻辑上最终用户使用的服务都来自只有唯一接口的同一地方，对所有节点

它主要包含如下服务：

（1）单一入口点。（2）单一文件层次。（3）单一输入/输出。（4）但一个管理和控制点。（5）单一网络。（6）单一存储空间。（7）单一作业管理系统。（8）单一用户界面。（9）单一进程空间。

10.举例说明为何下图所示的网络不是非阻塞网络。



> *本题的解法只需要找到一对冲突的输入/输出即可*

在本题中，存在冲突的输入输出对，为$000 \rightarrow 110$和$110 \rightarrow 111$。因此该网络不算非阻塞网络。

11. 描述立方网络中的自路由算法

解：对于$N = 2^n$个节点的n维立方体，令源节点的二进制编码为：$S = S_{n-1}S_{n-2}\dots S_1S_0$，令目的节点为$D = d_{n-1}d_{n-2}\dots d_1d_0$。

将n维表示成$i = 1, 2, 3, \dots, n$。其中第i维对应节点地址的第$i-1$位，设$v = v_{n-1}v_{n-2}\dots v_1v_0$是路径中任一中间节点。

将算法可描述如下：

输入：待选路的信包在源处理器中。

输出：将源处理器中信包送至目的地。

过程：

```
for i=1 to n do
    r_i = s_{i-1} ⊕ d_{i-1};
end
i=1,v=s;
while i <= n do
    if  r_i = 1 then
        从当前节点v 选路到节点
    endif
    i=i+1;
end
```

超立方体路由算法(超立方体维序路由算法或E-cube算法)

```
输入：  当前节点的地址Current和目的节点地址Dest
输出：  选择输出通道Channel
过程：
offset = Current ⊕ Dest;
if offset == 0 then
    Channel = Internal;
else
    Channel =FirstOne(offset); //返回第一个值为1的位
endif
```

超立方体最小P-Cube算法(立方网络中的自路由算法) 选这个

```
输入：  当前节点的地址Current和目的节点地址Dest
输出：  选的择输出通道Channel
过程：
E0 ={},E1={};
for i=0 to n-1 do
   if digit(Current,i)==0 and digit(Dest,i)==1 then
        E0 = E0 + {i};
    endif
    if digit(Current,i)=1 and digit(Dest,i)==0 then
        E1 = E1 + {i};
    endif
end

if E0 != {} then
    Channel = Select(E0);
endif
if E0=={} and E1!={} then
    Channel = Select(E1);
endif
if E0=={} and E1=={} then
    Channel = Internal;
endif
```

分析题(问答题)

> *问答题主要集中在第三章(网络路由 拓扑结构) 和 第四章 (高速缓存一致性 存储一致性等) 参考作业题目和往年例题。*

【题型一】 MESI Dragon分析(高速缓存一致性 P288)

> *MESI是一个四态写回(写)无效协议，Dragon是一个四态写回(写)更新协议。*
>
> *写回：当高速缓存块中的内容被替换时，才被写回主存。*
>
> *写无效/写更新：当一个高速缓存块被更新时，是无效其他块，还是广播更新其他块。*

给定一个序列，已知一系列操作的时钟周期，来比较执行代价(计算时钟周期)和性能差异(从一致性协议角度出发)。

我们常常将读写命中、缺失引起的总线事务、缺失引起的高速缓存块传输分别记为：H、B、T。

例题 往往假设：所有的存取操作都针对同一个内存位置，r/w代表读/写，数字代表发出该操作的处理器。假设所有高速缓存在开始时是空的，使用写分配策略，并且使用下面的性能模型：

- 读/写高速缓存命中，代价1个时钟周期；
- 缺失引起简单的总线事务（如BusUpgr，BusUpd），60个时钟周期；
- 缺失引起整个高速缓存块传输，90时钟周期。

对MESI(M修改 E互斥干净 S共享最新 I无效)而言：

当读发生时，若当前块无效时**(I)**，则引发BusRd信号并进行传输(从内存到高缓 或者高缓到高缓) 即**BTH**；

当写发生时，若当前块无效时**(I)**，则引发BusRdX信号并进行传输，同时其他块变I，即**BTH**;

若当前块是存在多个拷贝的最新状态(**S**)，则引发BusUpgr升级信号，置其他块为I，即**BH**。

当块处于E|S|M(最新态)时，读写只触发H。

确保写发生时无效其他块。

对于Dragon(E互斥干净 SC共享干净 SM共享最新 M单块最新 特殊无效状态) 而言：

当读发生时，若当前块Miss，引发BusRd信号，并进行传输(从内存到高缓 或高缓到高缓) 即**BTH**；

当写发生时，若当前块Miss，引发BusRd信号，并进行传输(从内存到高缓 或高缓到高缓)，写后引发更新信号(BusUpd)，即**BTBH**。

若当前块SC或SM，只引发BusUpd信号，不涉及块传输，只有BH。

若当前块是E或M，只有H。

(注意：更新信号只广播修改的部分而不是整个块 因此不算T只算B)

当块不是特殊无效的情况下，读都只有H；当块不是特殊无效的情况下，SX只有BH，E/M只有H。

序列如下：

- 序列1：r1 w1 r1 w1 r2 w2 r2 w2 r3 w3 r3 w3；(作业)
- 序列2：r1 r2 r3 w1 w2 w3 r1 r2 r3 w3 w1；(作业)
- 序列3：r1 r2 r3 r3 w1 w1 w1 w1 w2 w3；(作业)
- 序列4：r1,r1,r1,r2,r2,r2,r3,r3,r3,w1,w1,w1,w2,w1,w1,w3,r3; (2017年考题)
- 序列5：r1,r1,r2,r2,r3,r3,r4,r4,w1,w1,w1,w2,w2,w3,w3,w4,w4; (2018考题)

解：我们将读写命中、缺失引起的总线事务、缺失引起的高速缓存块传输分别记为：H、B、T。

> *当然可以选择在一个读或取中overlap掉小的时钟周期，比如BTH只记T，BH只记B。*

对于MESI：

序列1： BTH H H H BTH BH H H BTH BH H H； 5B+3T+12H = 582.

序列2： BTH BTH BTH BH BTH BTH BTH BTH H BH BTH; 10B+8T+11H = 1331

序列3：BTH BTH BTH H BH H H H BTH BTH; 6B+5T+10H = 820

序列4：BTH H H BTH H H BTH H H BH H H BTH BTH H BTH H; 7B+6T +17H = 977

序列5：BTH H BTH H BTH H BTH H BH H H BTH H BTH H BTH H; 8B + 6T + 17H = 1037

对于Dragon:

序列1：BTH H H H BTH BH H BH BTH BH H BH; 7B+3T + 12H = 702

序列2：BTH BTH BTH BH BH BH H H H BH BH; 8B + 3T +11H = 761

序列3：BTH BTH BTH H BH BH BH BH BH BH; 9B+ 3T + 10H = 820

序列4：BTH H H BTH H H BTH H H BH BH BH BH BH BH BH H; 10B+3T + 17H = 887

序列5：BTH H BTH H BTH H BTH H BH BH BH BH BH BH BH BH BH; 13B+4T+17H = 1157

分析的话：因为Dragon总是更新所有处理器中的缓存，因此当其他处理器经常使用更新后的数据时，Dragon协议命中次数更多，且用更多的总线事务来取代代价更高缓存块传输动作，以此降低延迟；此外的情况下，MESI更好。

【题型二】顺序一致性存存储模型下的进程输出（存储一致性 P224）

题目中往往会给定前提是顺序一致性存储模型，给出多个线程的代码(赋值 修改 输出等)，让给出所有非法的输出或分析合法的输出。

要点：(1) 同一个程序内的执行必须按照顺序。(2)不同程序间的执行可以乱序，但对内存的操作是全局性修改。

【题目一】 在顺序一致性存储模型下，有三个并行执行的进程：

P1 A=1; Print(B,C);

P2 B=1; Print(A,C);

P3 C=1; Print(A,B);

试问001110是否是合法的输出，并给出所有的非法输出，加以解释。

解：根据三个程序的特征，可以确定，在输出最后两位的时候，至少有两个变量会被设置为非0值，且最后一个print所打印两个变量必然非0，故可以得到三种非法输出＊＊＊＊00, ＊＊＊＊0＊, ＊＊＊＊＊0
当完成前两位输出后，此时至少会有一个变量被设置为非0，且该变量一定会在后续两次输出中被输出，故可以知道中间两位必然全不为0，得到一种非法输出
＊＊00＊＊

根据上面的分析，可能的合法输出有
$000111, 001011, 001111, 010111, 011011, 011111, 100111, 101011, 101111, 110111, 111011, 111111$

当前两位均为0时，分析可知此时三个处理机必然有一个已经执行结束，而另外两个还未执行，分析中间两位，对于剩余两个未执行的处理机，中间两位为$01, 10, 11$均合法

当前两位为01时，此时可以知道在执行print的处理机之外还有某个处理机执行了赋值语句，此时可能的执行序列为`... P3.1 ... P1.2 ...`且在`P1.2`之前`P2.1`不能被执行，故中间两位可能的输出包括$10, 11$，同理假设其它可能的执行顺序，可以得到所有的可能输出包括$01, 10, 11$

同理，对于前两位为10的情况进行分析，可知中间两位的可能输出为$01, 10, 11$

当前两位为11时，此时可以知道三个赋值语句必然都被执行了，后续只存在一种合法输出，即111111

故最终所有的非法输出为$****00, ****0*, *****0, **00**, 110111, 111011$
其中$*$为通配符，可以为0或1。

【题目二】 在顺序一致性存储模型下，有三个并行执行的进程：

P1 1: A=1; 2: C=1; 3: u=B;

P2 4:B=1; 5:C=2; 6:v=A;

P3 7:B=2; 8:A=2; 9: w=C;

试问哪些(u,v,w)不是一个合法的输出，并加以解释。

解： 我们对上述语句进行编号。

(1)假设 u=0时，即4和7 都在3之后执行。

若v=0, 非法，1必定在6之前执行。 (0,0,*)非法。

若v=1，则说明 8在6之后执行，此时w只能是2，也就是(0,1,0)和(0,1,1)非法。

若v=2，则说明6在8之后-执行，无其他约束，w能取2或1，也就是(0,2,0)非法。

(2)假设u=1时，即3在4之后执行，(7可能在4之前 或者在3之后)

若v=0，说明1和8在6之后执行，此时w显然不能取0 ，w可以取1或2。也就是(1,0,0)非法。

若v=1，6在1之后执行，8在6之后或者1之前执行，w可取任意值。

若v=2，6在8之后执行，w可取任何值。

(3)u=2时，3在7之后执行，4可能在3之后或者在7之前。

若v=0，则1和8在6之后执行，w可以取1和2，但不能取0，(2,0,0)非法。

若v=1，则6在1之后执行，8在6之后或者在1之前执行，w可取任何值。

若v=2，则6在8之后执行，1在6之后或者8之前执行，w可取任何值。

也就是 (0,0,0) (0,0,1) (0,0,2) (0,1,0) (0,1,1) (0,2,0) (1,0,0) (2,0,0)。

【题型三】预取策略考察

P334抄书是11周期，考题出现过12周期，两个对比下。

【题型四】双路径/多路径 多播路由

- 二维网格节点标记法：

  网络中的每个节点u都被指定一个标记l(u)，第一个节点标记为0，最后一个节点标记为n-1。
  l(x,y)=yn+x　　y为偶数
  $l(x,y)$=yn+n-x-1 y为奇数

  偶数行(比如0、2行) 从左到右 小到大， 奇数行(比如1、3行) 从右到左 小到大。

- 通道网络
  - 高通道子网包括所有从低标记节点指向高标记节点的通道。（D_H 从低到高）

    也就是向高方向移动被允许：上、(偶行)右、(奇行)左
  - 低通道子网包括所有从高标记节点指向低标记节点的通道。(D_L 从高到底)

    也就是向低方向移动被允许，下、(偶行)左、(奇行)右

双路径-多播

(1) 比源点大的点组成高通道子网DH，比源点小的点组成低通道子网DL。 将目标点按照处于高通道、低通道进行分组。
(2) 源点分别在高通道和低通道网络中，无环路地访问各自子网内的目标节点(注意不同子网内移动方向限制不同)

解题思路： (1)先画出二维网络的全图标好坐标，圆形画源点，带弧的矩形画目标点，其他暂不画外框。(注意X列标 Y行标 从0开始)

(2) 为每个节点完成节点标记，将目标节点划分出DH和DL。

(3)同时发出两条路径， 只沿小标记方向访问DL的目标点，沿大标记方向访问DH的目标点，注意恰当位置拐弯。

(4) 最后补上其他点的长方形外框，注意检查无环以及是否符号子网内的移动限制。

多路径-多播

(1) 把高低通道DH和DL进一步细分，例如：

- DL按照小于源的x坐标分一组，大于等于源的x坐标分一组(DL1,DL2)；(不相交的两块)
- DH按照小于等于源的x坐标分一组，大于源的x坐标分一组(DH1,DH2)。(不相交的两块)

(2)源点分别在四个子网中，无环地访问各个子网中的目标点，注意子网的方向限制。

解题思路： (1)先画出二维网络的全图标好坐标，圆形画源点，带弧的矩形画目标点，其他暂不画外框。(注意X列标 Y行标 从0开始)

(2) 为每个节点完成节点标记，用铅笔划分出DH1 DH2 DL1 DL2的区域范围。

(3)同时发出四条路径， 只沿小标记方向访问DL*的目标点，沿大标记方向访问DH*的目标点，注意恰当位置拐弯。

(4) 最后补上其他点的长方形外框，擦去区域范围， 注意检查无环以及是否符号子网内的移动限制。

【题型五】 二维网络 路由算法

1. 给出二维网格中的最小西向优先算法。

   输入：当前节点坐标(Xcurrent, Ycurrent)和目的节点坐标(Xdest, Ydest)

   输出：选择输出通道Channel

   过程：

```
Xoffset = Xdest - Xcurrent
Yoffset = Ydest - Ycurrent

if Xoffset < 0 then
Channel = X-;
endif

if Xoffset > 0 and Yoffset < 0 then
Channel = Select(X+,Y-);
endif

if Xoffset > 0 and Yoffset > 0 then
Channel = Select(X+,Y+);
endif

if Xoffset > 0 and Yoffset == 0 then
Channel = X+;
endif

if Xoffset == 0 and Yoffset <0 then
Channel = Y-;
endif

if Xoffset == 0 and Yoffset >0 then
Channel = Y+;
endif

if Xoffset == 0 and Yoffset == 0 then
Channel = Internal  ;
endif
```

2. 给出二维网络种的最小北向后算法。

输入：当前节点坐标(Xcurrent, Ycurrent)和目的节点坐标(Xdest, Ydest)

输出：选择输出通道Channel

过程：

```
Xoffset = Xdest - Xcurrent
Yoffset = Ydest - Ycurrent

if Xoffset > 0 and Yoffset < 0 then
Channel = Select(X+,Y-);
endif

if Xoffset < 0 and Yoffset < 0 then
Channel = Select(X-,Y-);
endif

if Xoffset == 0 and Yoffset < 0 then
Channel = Y-;
endif

if Xoffset < 0 and Yoffset >= 0 then
Channel = X-;
endif

if Xoffset == 0 and Yoffset > 0 then
Channel = Y+;
endif


if Xoffset > 0 and Yoffset >= 0 then
Channel = X+;
endif

if Xoffset == 0 and Yoffset == 0 then
Channel = Internal;
endif
```

3. 给出二维网格中的最小负优先路由算法。

输入：当前节点坐标(Xcurrent, Ycurrent)和目的节点坐标(Xdest, Ydest)

输出：选择输出通道Channel

过程：

```
Xoffset = Xdest - Xcurrent
Yoffset = Ydest - Ycurrent

if Xoffset < 0 and Yoffset < 0 then
Channel = Select(X-,Y-);
endif
```

```
if Xoffset < 0 and Yoffset >= 0 then
Channel = X-;
endif

if Xoffset >= 0 and Yoffset < 0 then
Channel = Y-;
endif

if Xoffset > 0 and Yoffset > 0 then
Channel = Select(X+,Y+);
endif

if Xoffset > 0 and Yoffset == 0 then
Channel = X+;
endif

if Xoffset == 0 and Yoffset > 0 then
Channel = Y+;
endif

if Xoffset == 0 and Yoffset == 0 then
Channel = Internal;
endif
```

单向二维环绕王的维序算法

输入：当前节点坐标(Xcurrent, Ycurrent)和目的节点坐标(Xdest, Ydest)

输出：选择输出通道Channel

过程：

```
Xoffset = Xdest - Xcurrent;
Yoffset = Ydest - Ycurrent;

if Xoffset <0 then
    Channel = c_{00};
endif
if Xoffset >0 then
    Channel = c_{01};
endif
if Xoffset ==0 and Yoffset <0 then
    Channel = c_{10};
endif
if Xoffset==0 and Yoffset > 0 then
    Channel == c_{11};
endif
if Xoffset==0 and Yoffset ==0 then
    Channel = Internal;
endif
```

二维网络中的XY路由算法

输入：当前节点坐标(Xcurrent, Ycurrent)和目的节点坐标(Xdest, Ydest)

输出：选择输出通道Channel

过程：

```
Xoffset = Xdest - Xcurrent;
Yoffset = Ydest - Ycurrent;
if Xoffset < 0 then
    Channel=X-;
endif

if Xoffset > 0 then
    Channel = X+;
endif

if Xoffset ==0 and Yoffset < 0 then
    Channel = Y-;
endif

if Xoffset == 0 and Yoffset > 0 then
    Channel = Y+;
endif

if Xoffset == 0 and Yoffset ==0 then
    Channel = Internal;
endif
```

【题型六】LL-SC(锁住读出-条件写入)实现锁

题目：给定ll和sc原子指令，给出实现xx锁的指令序列。

ll指令是Load-Locked，用来从指定内存中取得数据。

sc指令是Store-Conditional，用来检查从ll执行开始，内存位置的值是否发生改变，若无则新值写入该位置，反之则失败，不进行写入。

Test & Set Lock(作业题)

```
/*默认锁定状态时locatin存1，解锁态存0*/
lock : ll reg1, location  /*将内存位置的值加载到reg1*/
       bnz reg1, lock     /*如果锁定 则重试*/
       mov reg2, 1
       sc  location, reg2 /*将1写入 表明获得锁*/
       beqz lock          /*sc标记位是0 写入失败 则重试*/
       ret
unlock: st location, #0  /*写入0 释放锁*/
        ret
```

Ticket Lock(20年考题)

```
/*location是共享计数器的内存位置*/
ticket: ll reg1, location  /*获取共享计数器当前值*/
        inc reg1           /*共享计数器加1*/
        sc location,reg1 /*条件写入共享计数器*/
        beqz ticket      /*标记位是0 写入失败 则重试*/
        ret
/*now_serving存放了全局信号*/
lock: ld reg2, now_serving
      cmp reg2, reg1     /*比较全局和当前进程票据*/
      bnz lock           /*不相等则继续盲等待*/
      ret
unlock: inc reg2   /*now-serving加1*/
        st location, reg2  /*重新写入now-serving*/
        ret
```

Array-based Lock(07年考题)

```
/*array是锁数组, 初始均置0,为2表示已获取，为1表示锁空闲，下标从1开始*/
position: mov reg1,#0            /*用reg1表示获取的位置*/
   start: inc reg1              /*位置加1*/
          ll reg2, array[reg1] /*先尝试获取0位置*/
          cmp reg2,#2
          beqz start            /*该位置被占用则尝试下一个*/
          sc location[reg1],#2 /*未占用则尝试占用该位置*/
          beqz start           /*若sc失败则向下一个位置尝试*/
          ret
lock: ld reg2, location[reg1] /*不断在该位置忙等*/
      cmp reg2,#1
      bnz lock                /*锁不空闲则继续尝试*/
      ret
unlock: inc reg1
        st array[reg1],1     /*将锁空闲信号写入下个位置*/
```

compare && swap

```
(来自资料)
try: ll reg1, location
bnz reg1, try
sub reg3, reg2,reg1
bnz reg3,try
sc location,reg4
beqz reg4,try
ret
```