期末复习

习题课

作业1

考虑通过添加专用硬件来增强处理器的加密/解密能力。加密操作在专用硬件上运行的速度是普通硬件上的 20 倍，而解密操作是快 10 倍。假设，在普通硬件上花费在加密操作上的时间百分比是 e，而在解密操作上时间百分比是 d.

1. 绘图表示使用专用硬件所获得的总体加速比（y 轴）与 e（x 轴）之间的关系。

2. 如果总体加速比为 2，那么使用专门硬件后花费在解密操作上的时间百分比是多少?

3. 我们能够得到的最大总体加速比是多少？哪种类型的工作会给我们带来这样的加速比？获得最大加速比时，e 和 d 分别是多少?

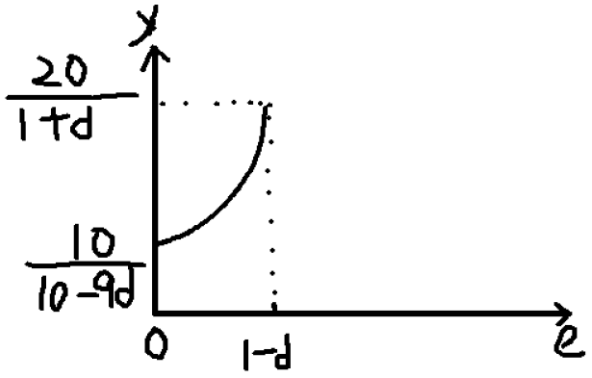
作业1

1. 绘图表示使用专用硬件所获得的总体加速比（y 轴）与 e（x 轴）之间的关系。

由Amdal定律，假设d为定值，只改变e，总体加速比为：

$$y = 1/(1-e-d + e/20 + d/10)$$

其中，e∈(0, 1−d)，大致图像如下：



作业1

2. 如果总体加速比为 2，那么使用专门硬件后花费在解密操作上的时间百分比是多少?

由 1/(1-e-d + e/20 + d/10) = 2 推出 d = (10-19e)/18；加速后解密操作的时间变为d/10，因此，解密操作的占比为：

$$\frac{d/10}{1-e-d+e/20+d/10}=\frac{10-19e}{90}$$

另一个角度：不用解e和d的关系，设原始（优化前）总用时为t0，则原始解密时间为dt0，优化后总时间为0.5t0（总加速比为2），解密时间为0.1dt0（时间变为原来的0.1），则占比为二者之比，即0.2d

作业1

3. 我们能够得到的最大总体加速比是多少？哪种类型的工作会给我们带来这样的加速比？获得最大加速比时，e 和 d 分别是多少

当e=1，d=0时，可以得到20倍的加速比，此时全部为加密操作，没有解密操作。

作业2

给定某内存系统，你的任务是确定其cache属性，包括块大小、相联度、cache大小、替换策略。已知这些cache属性值的范围是：

块大小：8, 16, 32, 64, or 128 Bytes

相联度：2-, 4-, or 8-way

Cache大小：4K or 8KB

替换策略：LRU or FIFO

在这个系统上，你唯一可以收集的统计数据是每次执行一系列内存访问后缓存命中率。以下是你观察到的情况：

序列	访问的内存地址(先->后)	命中率
1	0 16 24 25 1024 255 1100 305	2/8
2	31 65536 65537 131072 262144 8 305 1060	3/8
3	262145 65536 4	2/3

假设最开始时cache为空，3个序列连续访问（即序列1结束后不清空cache，马上开始序列2）。

推测cache的块大小、相联度、cache总大小、替换策略都是什么？并解释推断过程。

作业2

块大小：8, 16, 32, 64, or 128 Bytes
相联度：2-, 4-, or 8-way
Cache大小：4K or 8KB
替换策略：LRU or FIFO

0-> 00000
16->10000
24->11000
25->11001
1024->10000000000
255->11111111
1100->10001001100
305-> 100110001
31->11111
65536->10000000000000000
65537->10000000000000001
131072->100000000000000000
262144->1000000000000000000
8->1000
1060->10000100100
262145->10000000000000000001
4->100

序列	访问的内存地址(先->后)	命中率
1	0 16 24 25 1024 255 1100 305	2/8
2	31 65536 65537 131072 262144 8 305 1060	3/8
3	262145 65536 4	2/3

Cache Block Size： 16B
序列1中的缓存命中率是2/8。这意味着有2次命中。根据缓存块大小，我们可以推断出Block Size为16时，24,25命中，其他块大小都不是2次命中。

作业2

块大小：8, 16, 32, 64, or 128 Bytes
相联度：2-, 4-, or 8-way
Cache大小：4K or 8KB
替换策略：LRU or FIFO

0-> 00000
16->10000
24->11000
25->11001
1024->10000000000
255->11111111
1100->10001001100
305-> 100110001
31->11111
65536->10000000000000000
65537->10000000000000001
131072->100000000000000000
262144->1000000000000000000
8->1000
1060->10000100100
262145->10000000000000000001
4->100

序列	访问的内存地址(先->后)	命中率
1	0 16 24 25 1024 255 1100 305	2/8
2	31 65536 65537 131072 262144 8 305 1060	3/8
3	262145 65536 4	2/3

Associativity：2
序列2中的缓存命中率是3/8，这意味着有3次命中。
我们已经知道缓存块大小是16B，因此有4位偏移。
序列2中访问地址31会命中(和16在同一个block)，因为缓存块不会被替换。
序列2中访问地址305会命中，因为缓存块不会被替换。
序列2中访问地址65537会命中，因为缓存块不会被替换。
因此，其他所有访问应该都会失效。
序列2中访问地址65536、131072和262144会失效，因为这些地址没有属于任何之前访问过的缓存块。
地址65536、131072和262144的index和0相同 (index最多9bits)，会被映射到set 0中。
地址8未命中，表明其block被替换，因为它的缓存块映射到set 0，所以说明set 0大小必然小于4，因此关联度一定是2。

作业2

块大小：8, 16, 32, 64, or 128 Bytes
相联度：2-, 4-, or 8-way
Cache大小：4K or 8KB
替换策略：LRU or FIFO

0-> 00000
16->10000
24->11000
25->11001
1024->10000000000
255->11111111
1100->10001001100
305-> 100110001
31->11111
65536->10000000000000000
65537->10000000000000001
131072->100000000000000000
262144->1000000000000000000
8->1000
1060->10000100100
262145->10000000000000000001
4->100

序列	访问的内存地址(先->后)	命中率
1	0 16 24 25 1024 255 1100 305	2/8
2	31 65536 65537 131072 262144 8 305 1060	3/8
3	262145 65536 4	2/3

Cache Size：无法判断

作业2

块大小：8, 16, 32, 64, or 128 Bytes
相联度：2-, 4-, or 8-way
Cache大小：4K or 8KB
替换策略：LRU or FIFO

0-> 00000
16->10000
24->11000
25->11001
1024->10000000000
255->11111111
1100->10001001100
305-> 100110001
31->11111
65536->10000000000000000
65537->10000000000000001
131072->100000000000000000
262144->1000000000000000000
8->1000
1060->10000100100
262145->10000000000000000001
4->100

序列	访问的内存地址(先->后)	命中率
1	0 16 24 25 1024 255 1100 305	2/8
2	31 65536 65537 131072 262144 8 305 1060	3/8
3	262145 65536 4	2/3

替换策略：FIFO

- 缓存块大小是16B
- 缓存是2路关联的

序列3中的缓存命中率是2/3，这意味着有2次命中。
使用LRU策略时，只有序列3中访问地址262145会命中。使用FIFO策略时，序列3中访问地址262145和4会命中。
因此，缓存采用了FIFO策略。

作业3

某指令集支持8-bit虚拟内存地址，物理内存一共128 bytes，每个物理页16 bytes。页表使用一级结构，全部放在内存中。初始的时候内存布局如下：

物理页	存储内容
0	Empty
1	Virtual Page 13
2	Virtual Page 5
3	Virtual Page 2
4	Empty
5	Virtual Page 0
6	Empty
7	Page Table

采用容量为3个entry的TLB对地址转换结构进行缓存，TLB采用LRU替换策略（按照物理页访问的热度实现LRU）。初始时，TLB的内容为virtual page 0, 2和13对应的物理页号。对于下面的访问序列（virtual pages）：

0，13，5，2，14，14，13，6，6，13，15，14，15，13，4，3

（a）TLB的命中率为多少？

（b）全部访问结束后，TBL里面的内容是什么？

（c）全部访问结束后，页表的内容是什么？

作业3

某指令集支持8-bit虚拟内存地址，物理内存一共128 bytes，每个物理页16 bytes。页表使用一级结构，全部放在内存中。初始的时候内存布局如下：

物理页	存储内容
0	Empty
1	Virtual Page 13
2	Virtual Page 5
3	Virtual Page 2
4	Empty
5	Virtual Page 0
6	Empty
7	Page Table

采用容量为3个entry的TLB对地址转换结构进行缓存，TLB采用LRU替换策略（按照物理页访问的热度实现LRU）。初始时，TLB的内容为virtual page 0, 2和13对应的物理页号。对于下面的访问序列（virtual pages）：

0，13，5，2，14，14，13，6，6，13，15，14，15，13，4，3

（a）TLB的命中率为多少?

0(hit)，13(hit)，5(miss), 2 (miss), 14(miss, 分配page 0), 14(hit), 13(miss), 6(miss, 分配page 4), 6(hit), 13(hit), 15(miss, 分配page 6), 14(miss), 15(hit), 13(hit), 4(miss，分配page 5), 3(miss, 分配 page 2)

所以命中率为7/16

作业3

某指令集支持8-bit虚拟内存地址，物理内存一共128 bytes，每个物理页16 bytes。页表使用一级结构，全部放在内存中。初始的时候内存布局如下：

物理页	存储内容
0	Empty
1	Virtual Page 13
2	Virtual Page 5
3	Virtual Page 2
4	Empty
5	Virtual Page 0
6	Empty
7	Page Table

采用容量为3个entry的TLB对地址转换结构进行缓存，TLB采用LRU替换策略（按照物理页访问的热度实现LRU）。初始时，TLB的内容为virtual page 0, 2和13对应的物理页号。对于下面的访问序列（virtual pages）：

0，13，5，2，14，14，13，6，6，13，15，14，15，13，4，3

（b）全部访问结束后，TBL里面的内容是什么？

4, 13, 3

作业3

某指令集支持8-bit虚拟内存地址，物理内存一共128 bytes，每个物理页16 bytes。页表使用一级结构，全部放在内存中。初始的时候内存布局如下：

物理页	存储内容
0	Empty
1	Virtual Page 13
2	Virtual Page 5
3	Virtual Page 2
4	Empty
5	Virtual Page 0
6	Empty
7	Page Table

采用容量为3个entry的TLB对地址转换结构进行缓存，TLB采用LRU替换策略（按照物理页访问的热度实现LRU）。初始时，TLB的内容为virtual page 0, 2和13对应的物理页号。对于下面的访问序列（virtual pages）：

0，13，5，2，14，14，13，6，6，13，15，14，15，13，4，3

（c）全部访问结束后，页表的内容是什么？

物理页0-7对应的虚拟页分别是 14, 13, 3, 2, 6, 4, 15, Page table

作业5

假设某处理器有一个2-bits 的Global History Register（GHR），由所有的分支语句共享，其初始值为00（表示Not Taken）。每个Pattern History Table Entry（PHTE）包括一个2-bits的饱和计数器，其含义如下：

00 - Strongly Not Taken

01 - Weakly Not Taken

10 - Weakly Taken

11 - Strongly Taken

假设下面的代码运行在该处理器上。该代码含有两个分支语句（B1和B2）。

```
for (int i = 0; i < 1000000; i++) { /* B1 */
                                    /* TAKEN PATH for B1 */
    if (i % 3 == 0) {               /* B2 */
        j[i] = k[i] -1;             /* TAKEN PATH for B2 */
    }
}
```

作业5

```
for (int i = 0; i < 1000000; i++) { /* B1 */
                                    /* TAKEN PATH for B1 */
    if (i % 3 == 0) {               /* B2 */
        j[i] = k[i] -1;             /* TAKEN PATH for B2 */
    }
}
```

（a）有没有可能发生前5次循环所有分支预测全部错误？如果可能，列出每个PHTE可能的初始值（Not Taken用N表示，Taken用T表示）。

PHT Entry	Value
TT	01
TN	00
NT	01
NN	00或01

可能发生，前5次循环分支实际跳转的情况是：TT TN TN TT TN
给上面的跳转情况填加编号 T1 T2 T3 N4 T5 N6 T7 T8 T9 N10
对于GHR=NN，只在初始的时候为NN，其余都不是，因此，只有T1观察到了NN，所以，NN对应的PHTE初始值只能是00或者01，这样才会将T1预测为N；
对于GHR=TT，T3, N4, T9和N10都观察到了该pattern。如果将TT对应的PHTE设置为01，那么可以保证T3，N4，T9和N10都预测错误；
对于GHR=TN，T5和T7观测到了该pattern。如果要保证T5和T7都是预测错误，那么TN对应的PHTE必须为00；
对于GHR=NT，T2，N6和T8观测到了该pattern。为了保证这几个都预测错误，NT对应的PHTE为01即可；

作业5

```
for (int i = 0; i < 1000000; i++) { /* B1 */
                                    /* TAKEN PATH for B1 */
    if (i % 3 == 0) {               /* B2 */
        j[i] = k[i] -1;             /* TAKEN PATH for B2 */
    }
}
```

（b）当系统达到稳定状态之后（很多次循环之后），该分支预测器的准确率是否可以达到100%？如果可以达到，对PHT的初始值的设置是否有特殊要求？

PHT Entry	Value
TT	01
TN	00
NT	01
NN	00或01

稳态之后分支的真实模式为TTTNTN，可以推算，无论如何设置PHT的初始值，都不可能达到100%准确率。
TTTN连续2次出现，如果前面1次预测正确，后面一次必然也预测成T，不会预测成N！

作业6

考虑一个系统包含4个字节寻址的处理器。每个处理器有一个私有的L1 Cache，大小为256 bytes，采用直接映射（direct-mapped）和写回策略（write-back），block size是64 bytes。该系统采用MESI协议保证cache coherence。
内存访问的地址范围是0x50000000 - 0x5FFFFFFF。假设所有的内存访问地址在对应的cache block中的offset都是0，多处理器之间采用广播的方式进行通信。
假设宇宙射线导致该系统的某个cache line中的MESI协议状态发生了变化，导致了cache的不一致。下图给出了系统初始状态（已经发生了宇宙射线导致的变化）：

Cache 0		
	Tag	MESI State
Set 0	0x5FFFFF	M
Set 1	0x5FFFFF	E
Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	I

Cache 1		
	Tag	MESI State
Set 0	0x522222	I
Set 1	0x510000	S
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 2		
	Tag	MESI State
Set 0	0x5F111F	M
Set 1	0x511100	E
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 3		
	Tag	MESI State
Set 0	0x5FF000	E
Set 1	0x511100	S
Set 2	0x5FFFF0	I
Set 3	0x533333	I

作业6

Cache 0		
	Tag	MESI State
Set 0	0x5FFFFF	M
Set 1	0x5FFFFF	E
Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	I

Cache 1		
	Tag	MESI State
Set 0	0x522222	I
Set 1	0x510000	S
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 2		
	Tag	MESI State
Set 0	0x5F111F	M
Set 1	0x511100	E
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 3		
	Tag	MESI State
Set 0	0x5FF000	E
Set 1	0x511100	S
Set 2	0x5FFFF0	I
Set 3	0x533333	I

（a）哪个cache line的状态因为宇宙射线导致了变化，产生了不一致?

Cache 2, Set 1应该是S状态或者I状态。或者 Cache 3, Set 1应该是I状态

block size为64，所以地址的低6bits为block内部的offset，7-8bits为index，映射关系如下：00->set 0, 01->set 1, 10->set2, 11->set3。所以，表中只给出了高24bits，因为低6bits默认为0，7-8bits已知。

作业6

Cache 0		
	Tag	MESI State
Set 0	0x5FFFFF	M
Set 1	0x5FFFFF	E
Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	I

Cache 1		
	Tag	MESI State
Set 0	0x522222	I
Set 1	0x510000	S
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 2		
	Tag	MESI State
Set 0	0x5F111F	M
Set 1	0x511100	E
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 3		
	Tag	MESI State
Set 0	0x5FF000	E
Set 1	0x511100	S
Set 2	0x5FFFF0	I
Set 3	0x533333	I

（b）在初始状态基础上，下面的程序执行顺序，是否会导致错误的执行结果（即读到的数据不正确）？

order	Processor 0	Processor 1	Processor 2	Processor 3
1			ld 0x51110040	
2	st 0x5FFFFF40			
3				st 0x51110040
4		ld 0x5FFFFF80		
5		ld 0x51110040		
6		ld 0x5FFFFF40		

Processor 2读到错误数据！

第3条指令会访问processor 3的set 1，将其变为M状态，而processor 2的set1会变为I状态。这样就回到了一致状态。

作业6

Cache 0		
	Tag	MESI State
Set 0	0x5FFFFF	M
Set 1	0x5FFFFF	E
Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	I

Cache 1		
	Tag	MESI State
Set 0	0x522222	I
Set 1	0x510000	S
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 2		
	Tag	MESI State
Set 0	0x5F111F	M
Set 1	0x511100	E
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 3		
	Tag	MESI State
Set 0	0x5FF000	E
Set 1	0x511100	S
Set 2	0x5FFFF0	I
Set 3	0x533333	I

(c) 在初始状态基础上，下面的程序执行顺序，是否会导致错误的执行结果(即读到的数据不正确)？

order	Processor 0	Processor 1	Processor 2	Processor 3
1				ld 0x51110040
2	ld 0x5FFFFF00			
3			ld 0x51234540	
4	st 0x5FFFFF40			
5				ld 0x51234540
6	ld 0x5FFFFF00			

会，当宇宙射线导致了Cache 3的set 1从I变为S，第1条指令会读到错误数据。

作业6

Cache 0		
	Tag	MESI State
Set 0	0x5FFFFF	M
Set 1	0x5FFFFF	E
Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	E <- I

Cache 1		
	Tag <- 0x522222	MESI State
Set 0	0x5FF000	I
Set 1	0x510000	S
Set 2	0x5FFFFF	S
Set 3	0x533333	I <- S

Cache 2		
	Tag	MESI State
Set 0	0x5F111F	M
Set 1	0x511100	E
Set 2	0x5FFFFF	S
Set 3	0x533333	I <- S

Cache 3		
	Tag	MESI State
Set 0	0x5FF000	M <- E
Set 1	0x511100	S
Set 2	0x5FFFF0	I
Set 3	0x533333	I

（d）在初始状态基础上，怎么通过最少的内存访问指令，使得系统最终状态变成上图所示（假设宇宙射线不会再发生了），请写出内存访问顺序，具体操作（ld/st），访问的内存地址以及由哪个处理器发起。

最小的指令序列为：

（1）st 0x533333C0 //processor 0 ->映射到set3

（2）ld 0x5FFFFFC0 //processor 0 ->映射到set3

（3）ld 0x5FF00000 //processor 1

（4）st 0x5FF00000 //processor 3 ->映射到哦set0

只要保证（1）（2）之间的顺序，（3）（4）之间的顺序，其他指令顺序也可以。

作业7

一个程序员，写了如下C语言代码。他想在多核处理器上运行该程序，用两个线程，每个线程占一个核。处理器采用顺序一致性协议，变量X和flag存储在内存中，变量a和b存储在寄存器中。所有内存初始化为0。假设每一行C代码代表一条指令。

Thread T0	Thread T1
指令T0.0 X[0] = 1;	指令T1.0 X[0] = 0;
指令T0.1 X[0] += 1;	指令T1.1 flag[0] = 1;
指令T0.2 while(flag [0]] == 0);	指令T1.2 b = X[0];
指令T0.3 a = X[0];	
指令T0.4 X[0] = a*2;	

1. 变量a的最终值可能是多少？ 解释原因

2. 变量X[0]的最终值可能是多少？解释原因。

3. 变量b的最终值可能是多少？解释原因。

4. 假设该程序员想让变量a和b的取值在程序执行完毕时是相同的，那么在保留T1.1和T0.2两条指令的前提下，需要对原始程序如何做最小的改动（填写下表）？（提示：可以利用更多的flags）

作业7

一个程序员，写了如下C语言代码。他想在多核处理器上运行该程序，用两个线程，每个线程占一个核。处理器采用顺序一致性协议，变量X和flag存储在内存中，变量a和b存储在寄存器中。所有内存初始化为0。假设每一行C代码代表一条指令。

Thread T0	Thread T1
指令T0.0 X[0] = 1;	指令T1.0 X[0] = 0;
指令T0.1 X[0] += 1;	指令T1.1 flag[0] = 1;
指令T0.2 while(flag [0]] == 0);	指令T1.2 b = X[0];
指令T0.3 a = X[0];	
指令T0.4 X[0] = a*2;	

1. 变量a的最终值可能是多少？ 解释原因

顺序一致性保证每个线程自己的指令顺序执行。跨线程的同步由flag[0]来完成。Thread 0在T0.2停留，直到flag被T1.1置成1。至少有3种不同的顺序一致性顺序：
T1.0->T0.0->T0.1->T0.3 a的值为2
T0.0->T1.0->T0.1->T0.3 a的值为1
T0.0->T0.1->T1.0->T0.3 a的值为0

作业7

一个程序员，写了如下C语言代码。他想在多核处理器上运行该程序，用两个线程，每个线程占一个核。处理器采用顺序一致性协议，变量X和flag存储在内存中，变量a和b存储在寄存器中。所有内存初始化为0。假设每一行C代码代表一条指令。

Thread T0	Thread T1
指令T0.0 X[0] = 1;	指令T1.0 X[0] = 0;
指令T0.1 X[0] += 1;	指令T1.1 flag[0] = 1;
指令T0.2 while(flag [0]] == 0);	指令T1.2 b = X[0];
指令T0.3 a = X[0];	
指令T0.4 X[0] = a*2;	

2. 变量X[0]的最终值可能是多少？解释原因。

X[0]的最终值是a值的2倍，分别为4, 2, 0

作业7

一个程序员，写了如下C语言代码。他想在多核处理器上运行该程序，用两个线程，每个线程占一个核。处理器采用顺序一致性协议，变量X和flag存储在内存中，变量a和b存储在寄存器中。所有内存初始化为0。假设每一行C代码代表一条指令。

Thread T0	Thread T1
指令T0.0 X[0] = 1;	指令T1.0 X[0] = 0;
指令T0.1 X[0] += 1;	指令T1.1 flag[0] = 1;
指令T0.2 while(flag [0]] == 0);	指令T1.2 b = X[0];
指令T0.3 a = X[0];	
指令T0.4 X[0] = a*2;	

3. 变量b的最终值可能是多少？解释原因。

变量b的最终值可能是0, 1, 2, 4。
T1.2的顺序并没有约束，因此变量b可能在任意指令位置执行，即可能等于X[0]的任何可能值。

作业7

一个程序员，写了如下C语言代码。他想在多核处理器上运行该程序，用两个线程，每个线程占一个核。处理器采用顺序一致性协议，变量X和flag存储在内存中，变量a和b存储在寄存器中。所有内存初始化为0。假设每一行C代码代表一条指令。

Thread T0	Thread T1
指令T0.0 X[0] = 1;	指令T1.0 X[0] = 0;
指令T0.1 X[0] += 1;	指令T1.1 flag[0] = 1;
指令T0.2 while(flag [0]] == 0);	指令T1.2 b = X[0];
指令T0.3 a = X[0];	
指令T0.4 X[0] = a*2;	

4. 假设该程序员想让变量a和b的取值在程序执行完毕时是相同的，那么在保留T1.1和T0.2两条指令的前提下，需要对原始程序如何做最小的改动（填写下表）？（提示：可以利用更多的flags）

需要保证指令a=X[0]和指令b=X[0]之间不能有其它修改变量X[0]的指令。

Thread T0	**Thread T1**
T0.0 X[0] = 1;	T1.0 X[0] = 0;
T0.1 X[0] += 1;	T1.1 flag[0] = 1;
T0.2 while(flag[0] == 0);	T1.2 while(flag[1] == 0);
T0.3 a = X[0];	T1.3 b = X[0];
T0.4 flag[1] = 1;//可以放在T0.1和T0.3间	T1.4 flag[2] = 1;
T0.5 while(flag[2] == 0);	
T0.6 X[0] = a*2;	

作业8

GPU的利用率通常定义为处于busy状态的GPU 核心(PE，单个计算单元)占所有GPU核心的比例。考虑下面的代码片段。每个thread执行循环中的1次迭代（包含6条指令）。假设数组A、B、C已经在寄存器中（不需要从内存读入）。该GPU的一个warp包含64个threads，该GPU包含64个核心。假设数组B的每个元素的绝对值都小于10。

```
for (i = 0; i < 1024; i++) {
  A[i] = B[i] * B[i];
  if (A[i] > 0) {
    C[i] = A[i] * B[i];
    if (C[i] < 0) {
      A[i] = A[i] + 1;
    }
    A[i] = A[i] - 2; }
}
```

1.执行该代码段需要多少个warps?

2.执行整个代码段，最大的GPU利用率可能是多少?

3.获得最大的GPU利用率时，数组B的值有何特征?

4.执行整个代码段，最小的GPU利用率可能是多少?

作业8

GPU的利用率通常定义为处于busy状态的GPU 核心(PE，单个计算单元)占所有GPU核心的比例。考虑下面的代码片段。每个thread执行循环中的1次迭代（包含6条指令）。假设数组A、B、C已经在寄存器中（不需要从内存读入）。该GPU的一个warp包含64个threads，该GPU包含64个核心。假设数组B的每个元素的绝对值都小于10。

```
for  (i  =  0;  i  <  1024;  i++)  {
  A[i]  =  B[i]  *  B[i];
  if  (A[i]  >  0)  {
    C[i]  =  A[i] *  B[i];
    if  (C[i]  <  0)  {
      A[i]  =  A[i]  +  1;
    }
    A[i]  =  A[i]  -  2; }
}
```

1.执行该代码段需要多少个warps?

1024/64 = 16

作业8

GPU的利用率通常定义为处于busy状态的GPU 核心(PE，单个计算单元)占所有GPU核心的比例。考虑下面的代码片段。每个thread执行循环中的1次迭代（包含6条指令）。假设数组A、B、C已经在寄存器中（不需要从内存读入）。该GPU的一个warp包含64个threads，该GPU包含64个核心。假设数组B的每个元素的绝对值都小于10。

```
for (i = 0; i < 1024; i++) {
  A[i] = B[i] * B[i];
  if (A[i] > 0) {
    C[i] = A[i] * B[i];
    if (C[i] < 0) {
      A[i] = A[i] + 1;
    }
    A[i] = A[i] - 2; }
}
```

2.执行整个代码段，最大的GPU利用率可能是多少?

100%

作业8

GPU的利用率通常定义为处于busy状态的GPU 核心(PE，单个计算单元)占所有GPU核心的比例。考虑下面的代码片段。每个thread执行循环中的1次迭代（包含6条指令）。假设数组A、B、C已经在寄存器中（不需要从内存读入）。该GPU的一个warp包含64个threads，该GPU包含64个核心。假设数组B的每个元素的绝对值都小于10。

```
for (i = 0; i < 1024; i++) {
  A[i] = B[i] * B[i];
  if (A[i] > 0) {
    C[i] = A[i] * B[i];
    if (C[i] < 0) {
      A[i] = A[i] + 1;
    }
    A[i] = A[i] - 2; }
}
```

3.获得最大的GPU利用率时，数组B的值有何特征?

对于每64个连续值，或者都是0，或者都是正数，或者都是负数。

作业8

GPU的利用率通常定义为处于busy状态的GPU 核心(PE，单个计算单元)占所有GPU核心的比例。考虑下面的代码片段。每个thread执行循环中的1次迭代（包含6条指令）。假设数组A、B、C已经在寄存器中（不需要从内存读入）。该GPU的一个warp包含64个threads，该GPU包含64个核心。假设数组B的每个元素的绝对值都小于10。

```
for (i = 0; i < 1024; i++) {
  A[i] = B[i] * B[i];
  if (A[i] > 0) {
    C[i] = A[i] * B[i];
    if (C[i] < 0) {
      A[i] = A[i] + 1;
    }
    A[i] = A[i] - 2; }
}
```

4.执行整个代码段，最小的GPU利用率可能是多少?

$(64 \times 2 + 1 \times 4)/(64 \times 6) = 132/384$

作业8

GPU的利用率通常定义为处于busy状态的GPU 核心(PE，单个计算单元)占所有GPU核心的比例。考虑下面的代码片段。每个thread执行循环中的1次迭代（包含6条指令）。假设数组A、B、C已经在寄存器中（不需要从内存读入）。该GPU的一个warp包含64个threads，该GPU包含64个核心。假设数组B的每个元素的绝对值都小于10。

```
for (i = 0; i < 1024; i++) {
  A[i] = B[i] * B[i];
  if (A[i] > 0) {
    C[i] = A[i] * B[i];
    if (C[i] < 0) {
      A[i] = A[i] + 1;
    }
    A[i] = A[i] - 2; }
}
```

4.执行整个代码段，最小的GPU利用率可能是多少?

$(64 \times 2 + 1 \times 4)/(64 \times 6) = 132/384$

仅有1个thread通过了第1个分支语句，其他threads都没通过，因此，只有一个PE busy，其余都是空闲的。也就是，每连续64个值，有1个是负数，其余都是0。