课程编号1800860854O4P2008H 2023-2024学年秋季学期

# 计 算 机 体 系 结 构

# 第 17b 讲 复习与讨论

主讲教师：刘珂

2024年12月31日

中国科学院大学 University of Chinese Academy of Sciences
ICT 中科院计算所 INSTITUTE OF COMPUTING TECHNOLOGY,CAS

# HW4.1

- 假设你正在运行一个程序，该程序具有以下数据访问模式。这种模式只执行一次：
- 0x0 0x8 0x10 0x18 0x20 0x28

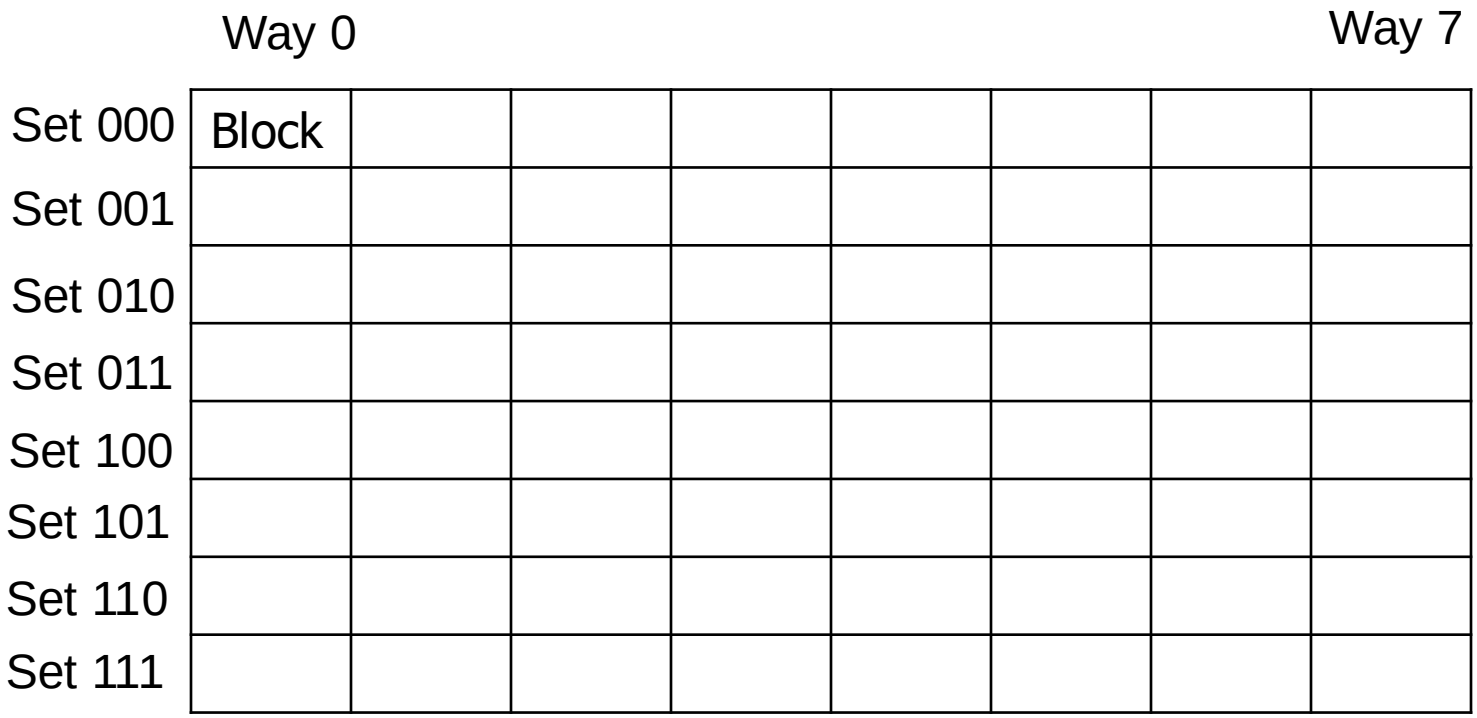
- (1)如果使用直接映射缓存(Direct Mapped Cache)，缓存大小为1 KB，块大小(Block Size)为8字节(bytes)，缓存中有多少组(sets)？

- (2)使用与问题(1) 中相同的缓存和块大小，对于给定的内存访问模式，直接映射缓存的缺失率(miss rate)是多少？

- (3)对于给定的内存访问模式，提出一种降低缺失率的缓存设计(保持缓存大小不变)，并给出相应的解释。

# HW4.1.a

- Sets: Cache size/block size
- = 2^10/2^3=2^7=128 sets



# HW4.1.b

- Index: 7b
- Offset: 3b


- 0x0: 0b0000000000 ==> Index :0
- 0x8: 0b0000001000 ==> Index :1
- 0x10:0b0000010000 ==> Index :2
- 0x18:0b0000011000 ==> Index :3
- 0x20:0b0000100000 ==> Index :4
- 0x28:0b0000101000 ==> Index :5

- 都是第一次访问(cold/compulsory miss), miss rate = 100%

# HW4.1.c

- (i) Increasing the degree of associativity to 2.
- 2-Way Set Associative:
- Index: 6b
- Offset:3b

- 0x0: 0b0000000000 ==> Index :0
- 0x8: 0b0000001000 ==> Index :1
- 0x10:0b0000010000 ==> Index :2
- 0x18:0b0000011000 ==> Index :3
- 0x20:0b0000100000 ==> Index :4
- 0x28:0b0000101000 ==> Index :5

- Miss rate = 100%

| block | |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |

# HW4.1.c

- (ii) Increasing the block size to 16 bytes.
- Sets = 2^10/2^4=2^6
- Index : 6b
  Offset: 4b

- 0x0: 0b0000000000 ==> Index :0
- 0x8: 0b0000001000 ==> Index :0
- 0x10:0b0000010000 ==> Index :1
- 0x18:0b0000011000 ==> Index :1
- 0x20:0b0000100000 ==> Index :2
- 0x28:0b0000101000 ==> Index :2

- Miss rate = 50%

# HW4.1 (4) and HW4.2 are Similar

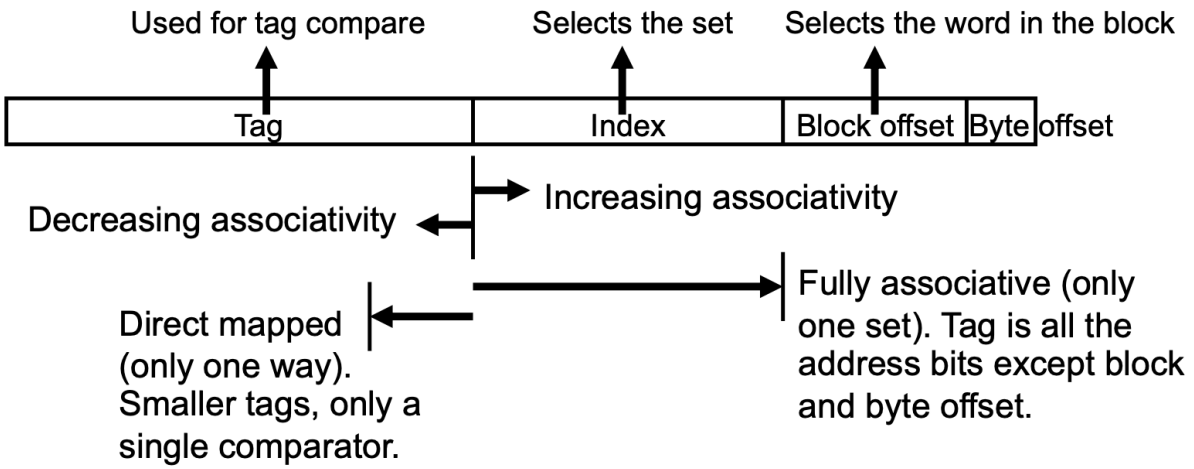
- (4)如果使用全相联数据缓存(Fully-associative Data Cache)，缓存大小为8 KB，块大小为 32 字节，假设地址位为 32 位(bits)。该缓存有多少组(sets)、有多少路(ways)、有多少索引位(index bits)、偏移位(offset bits)、标记位(tag bits)、标记数组(tag store)有多大？
  - Sets = 1 due to fully associativity
  - As cache size = #sets x #ways x blocksize, we have Ways = 2^13/2^5= 2^8=256,
  - Index bits = log2 (#sets) = log2(1) = 0
  - Offset bits = log2 (blocksize) = 5 bits
  - We have Tag bits + Index bits + Offset bits = Address width, thus Tag bits = 32–0–5=27 bits
  - Tag array size = #sets x #ways x Tag bits, we have 27*1*256=6912 bits

# HW4.5

- 如图所示，考虑一个具有以下参数的缓存：关联度（associativity）= 2，块（block）大小= 2个字（word），字大小= 32位，缓存大小= 32K个字，地址大小= 32位。你只需要考虑字地址。
- (a) 显示地址的标签（tag）、组（set）、块偏移（block offset）和字节偏移位（byte offset）。说明每个字段需要多少位。
- (b) 所有缓存标签（tag array）的大小是多少位？
- (c) 假设每个缓存块还有一个有效位（V）和一个脏位（D）。包括数据、标签和状态位在内的每个缓存组的大小是多少？

# HW4.5

- (a) 显示地址的标签（tag）、组（set）、块偏移（block offset）和字节偏移位（byte offset）。说明每个字段需要多少位。




- a)
  - Total number of block = 32K/2=16K
  - Block size = 2 words = 2*32b = 64b = 8B => 3b offset bits
  - Sets = 16K/2=8K => index = 13b
  - 2 word per block => block offset = 1b
  - Byte offset = 3-1 = 2b
  - Tag = 32 – 13 - 3 = 16b

# HW4.5

- b) 所有缓存标签（tag array）的大小是多少位？
  - Cache tags size 16b*16K=256Kb

- c) 假设每个缓存块还有一个有效位（V）和一个脏位（D）。包括数据、标签和状态位在内的每个缓存组的大小是多少？
  - Data size = 2 words = 64b
  - Tag = 16b
  - Valid bit = 1b
  - Dirty bit = 1b
  - Each set = 2*(64+16+1+1) = 164b

# HW 4.3

- 考虑以下循环，在有16 字节大小的指令缓存（I-cache）的系统上执行。数据缓存（D-cache）是全相联的，大小为1 KB。两个缓存都使用16字节的块（Block）。指令长度和数据字大小为4 B。寄存器 Register $1的初始值为40。寄存器 Register $0的值为0。（注意：假设循环的第一个指令与缓存块的开始对齐）。

```
Loop:   lw      $6, X($1)
        addi    $6, $6, 1
        sw      $6, Y($1)
        subi    $1, $1, 4
        beq     $1, $0, Exit
        j       Loop
Exit:   ...
```

- (a) 计算I-cache和D-cache的miss rate，考虑以下情况：
- X和Y是不同的数组。
- X和Y是相同的数组。

# HW4.3

- 在循环内，由于I-cache大小只有4个指令，所以有两个指令会miss (1 and 5 指令)，对于D-cache而言，每4个循环整个DCache都会被访问完，如果X和Y地址不同则会产生2次cache miss，相同则产生1次。

- 如果考虑10次循环，最后一次循环在beq就结束了，所以是I-cache miss rate 20/59.
- 对于D-cache而言，相当于是20次访问有3个循环需要重新载入缓存。6/20=0.3 (每次iteration 2次访问，所以20)
- 对于地址相同的情况答案减半(定位到相同的访问), 所以0.15

# HW4.3

- (b)　计算每条指令的平均周期数（CPI），使用基线理想CPI（理想缓存）等于2，并且cache miss 延迟为10个时钟周期。
- CPI = 2 + 0.34×10 + 2/6×0.30×10 cycles = 6.390 cycles, if X and Y are different arrays.
- CPI = 2 + 0.34×10 + 2/6×0.15×10 cycles= 5.895 cycles, if X and Y are the same array.
- ***考虑10次循环的时候，CPI在每条指令的基础上要加上I/D-cache miss的乘法。分别是20/59*10+2/6*(0.3 or 0.15)*10***

- (c) 如果使用一个32字节的I-cache，(a)的结果会如何改变？
- If the size of the I-cache is 32 B, the entire loop fits in it. There will be only two cold misses (fetch 2 16B blocks) in the first iteration. Thus, the I-cache miss rate will be 2/59 = 0.033.

# HW5.1

- 假设一个处理器使用 32 位虚拟地址和 32 位物理地址，处理器缓存参数如下：8组、2路组关联、256 字节缓存块大小。处理器系统使用基于1KB页面大小的虚拟内存系统。设计者需要选择：是采用虚拟地址寻址方式的Cache（即虚拟索引和虚拟标记），还是采用物理地址寻址方式的Cache（即物理索引和物理标记）。

- 1)　如果虚拟地址 VA = 0x00006825 (0b0000_0000_0000_0000_0110_1000_0010_0101)。在下表中标记出与此虚拟地址对应的数据可能驻留在虚拟寻址和物理寻址高速缓存的位置。如果数据可以驻留在所有条目中，则将“所有条目”填入相应位置。

# HW5.1

| 虚拟地址寻址cache | | |
|---|---|---|
| Index | Way0 | Way1 |
| 0 | X | X |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

| 物理地址寻址cache | | |
|---|---|---|
| Index | Way0 | Way1 |
| 0 | X | X |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | X | X |
| 5 | | |
| 6 | | |
| 7 | | |

- 页偏移地址：1K页面，最低10位地址（bit9~bit0：00_0010_0101）；
- Cache索引地址：bit10 ~ bit8（8组），因为Cache行是256字节，行内便宜用bit7 ~ bit0来寻址；
- 因此，索引地址的最高位可能会根据VA→PA转换过程发生变化。
-
- Vaddr: 0000100101
- Phyaddr: X0000100101
- X= 0, 000, index=0
- X=1, 100, index=4

# HW5.1

- 2) 如果物理地址：PA = 0x00006825 (0b0000_0000_0000_0000_0110_1000_0010_0101)。在下表中标记出与此物理地址对应的数据可能驻留的位置。如果数据可以驻留在所有条目中，则将“所有条目”填入相应位置。

| 虚拟地址寻址cache | | | | 物理地址寻址cache | | |
|---|---|---|---|---|---|---|
| Index | Way0 | Way1 | | Index | Way0 | Way1 |
| 0 | X | X | | 0 | X | X |
| 1 | | | | 1 | | |
| 2 | | | | 2 | | |
| 3 | | | | 3 | | |
| 4 | X | X | | 4 | | |
| 5 | | | | 5 | | |
| 6 | | | | 6 | | |
| 7 | | | | 7 | | |

- 这个问题是把上一问反过来说，对应同一个物理地址，其虚拟地址的**bit10**可能会改变
- Phyaddr: 000_0010_0101
- Vadd: X00_0010_0101

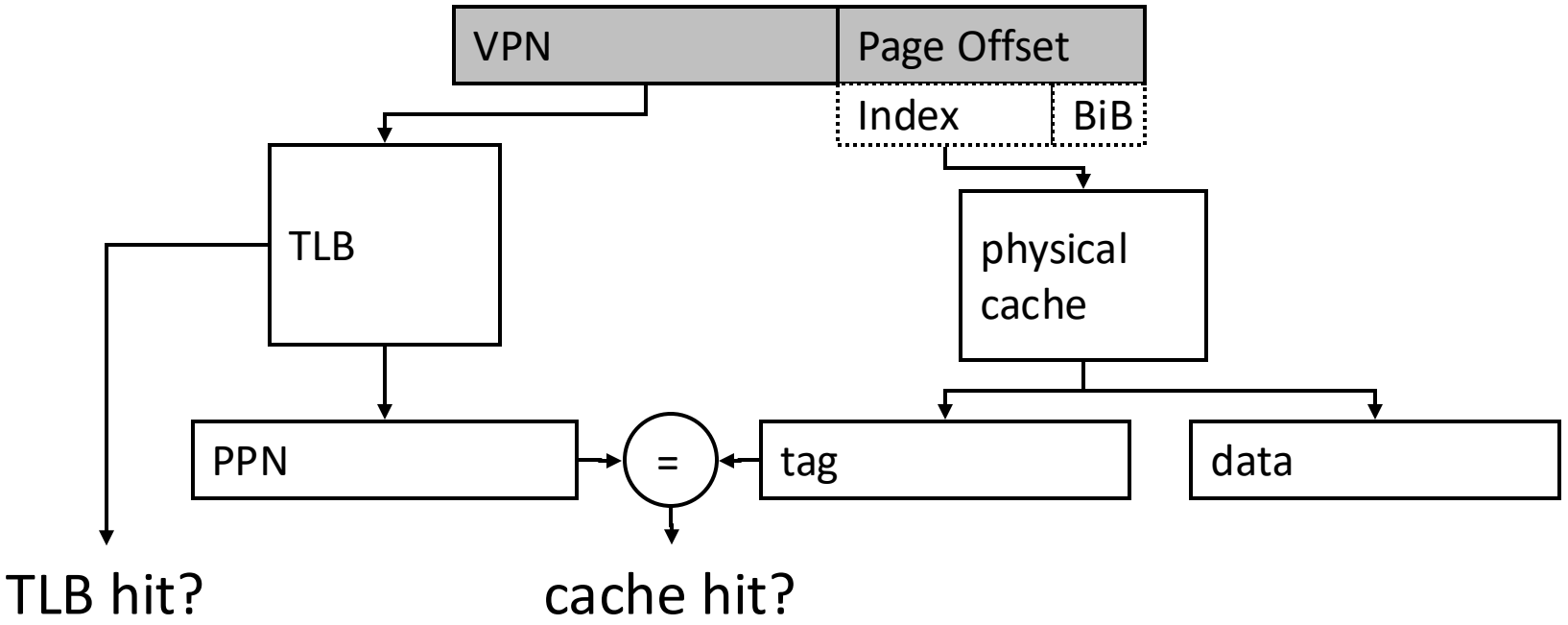
# More Problem

- 假设页面大小为 16KB，缓存块大小为 32 字节。 如果我想实现一个虚拟索引、物理标记的 L1 缓存（virtually indexed physically tagged L1 cache），我可以实现的最大直接映射 L1 缓存（direct-mapped L1 cache）大小是多少？我可以实现的最大 2 路（2-way）缓存大小是多少？

- 缓存行数：页面大小/缓存块大小=16KB/32B=512
- 直接映射:每个页面只能映射到缓存中唯一的一个缓存块，最大直接映射缓存大小取决于缓存块的数量
- 最大直接映射大小=缓存块数量 * 缓存块大小=512*32B=16KB
- 2路缓存:每个页面可以映射到两个缓存块中的任意一个，最大2路缓存大小是最大直接映射缓存大小的两倍
- 最大2路缓存大小=缓存块数量 * 缓存块大小 * 2=512*32B*2=32KB

# More Problem

- If C≤(page_size × associativity), the cache index bits come only from page offset (same in VA and PA)
- If both cache and TLB are on chip
  - index both arrays concurrently using VA bits
  - check cache tag (physical) against TLB output at the end



# HW5.2

- 一个ISA支持8位的、字节可寻址(byte-addressable)的虚拟地址空间。相应的物理内存仅有128字节。每个页16字节。使用了一个简单的单级转换方案，并且页表存储在物理内存中，物理内存的初始内容如下图所示。

| Frame Number | Frame Contents |
|---|---|
| 0 | Empty |
| 1 | Page 13 |
| 2 | Page 5 |
| 3 | Page 2 |
| 4 | Empty |
| 5 | Page 0 |
| 6 | Empty |
| 7 | Page Table |

- 使用LRU替换策略的TLB（3-entry）添加到该系统中。最初，这个TLB包含了页0、2和13。对于以下仿存（memory references）序列，请在TLB命中的entry周围画一个圆圈，并在产生错页（page fault）的条目周围画一个矩形。这个访存序列的TLB命中率是多少？（注意：在物理内存中使用替换页面的LRU策略）

- References (to pages): 0, 13, 5, 2, 14, 14, 13, 6, 6, 13, 15, 14, 15, 13, 4, 3.

# HW5.2

- Page table address is known

| Frame Number | Frame Contents |
|---|---|
| 0 | Empty->14 |
| 1 | Page 13 |
| 2 | Page 5->3 |
| 3 | Page 2 |
| 4 | Empty->6 |
| 5 | Page 0->4 |
| 6 | Empty->15 |
| 7 | Page Table |

| VPN | PPN |
|---|---|
| 0 | 5 |
| 1 | |
| 2 | 3 |
| 3 | 2 |
| 4 | 5 |
| 5 | 2 |
| 6 | 4 |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | 1 |
| 14 | 0 |
| 15 | 6 |

# HW5.2

References (to pages): (0), (13), 5, 2, [14], (14), 13, [6], (6), (13), [15], 14, (15), (13), [4], [3].

TLB Hit Rate = 7/16

13, 0, 2, when accessing 5, found in page table, 5->2

5, 13, 0, when accessing 2, found in page table, 2->3

2, 5, 13, when accessing 14, not found in page table, thus page fault, new an entry 14->0

14, 2, 5, when accessing 14,

14, 2, 5, when accessing 13, found in page table. 13->1

13, 14, 2, when accessing 6, not found in page table, thus page fault, new an entry 6->4

6, 13, 14, when accessing 6,

5, 13, 14, when accessing 13,

13, 5, 14, when accessing 15, not found in page table, thus page fault, new an entry 15->6

15, 13, 5, when accessing 14, found in page table, 14->0

14, 15, 13, when accessing 15,

15, 14, 13, when accessing 13,

13, 15, 14, when accessing 4, not found in page table, thus page fault, evict 0->5, and replace with 4->5,

4, 13, 15, when accessing 3, not found in page table, thus page fault, evict 5->2, and replace with 3->2,

3, 4, 13

# HW5.2

- (a) 在这个序列结束时，TLB中包含了哪三个条目？
- 4, 13, 3
- 
- (b) 8个物理页（physical pages）的内容是什么？
- Pages 14, 13, 3, 2, 6, 4, 15, Page table

# HW5.3

- 一个具有完美LRU替换策略的2路组相联(2-way set associative)写回(writeback)缓存需要15×$2^9$位的存储空间来实现标签存储(包括有效位、脏位和LRU位)。缓存是虚拟索引、物理标记(virtually indexed physically tagged)。虚拟地址空间为1 MB，页面大小为2 KB，缓存块大小为8字节。
- (a) 缓存的数据存储(data array)大小是多少字节？

- The cache is 2-way set associative.
- So, each set has 2 tags, each of size *t*, 2 valid bits, 2 dirty bits and 1 LRU bit (because a single bit is enough to implement perfect LRU for a 2-way set associative cache).
- Let i be the number of index bits.
- Tag store size = $2^i \times (2 \times t + 2 + 2 + 1) = 15 \times 2^9$
- Therefore,
- $2t = 10 \Rightarrow t = 5$
- $i = 9$
- Data store size = $2^i \times (2 \times 8)$ bytes = $2^9 \times (2 \times 8)$ bytes = 2^13 = 8 KB.

# HW 5.3

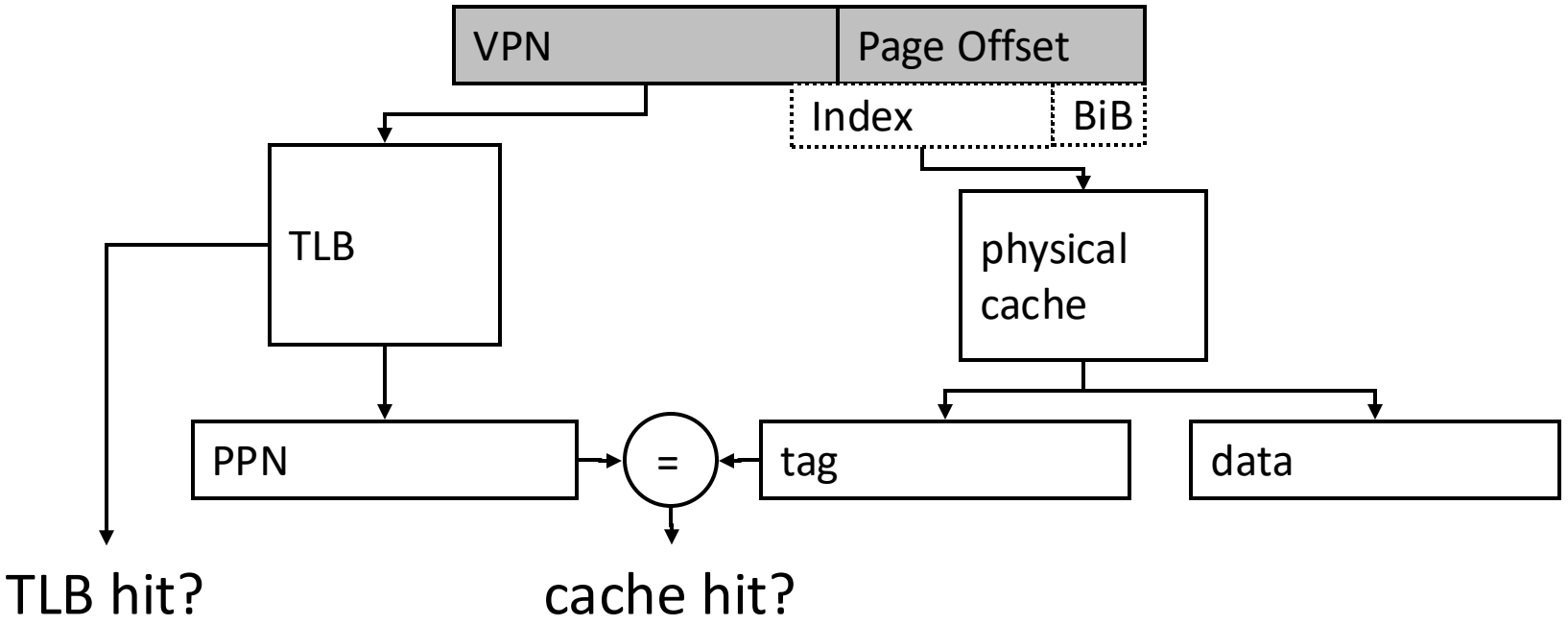
- (b) 虚拟索引中有多少位来自虚拟页号？

- 1 bit
- Page size is 2 KB. Hence, the page offset is 11 bits (bits 10:0).
- The cache block offset is 3 bits (bits 2:0) and the virtual index is 9 bits (bits 11:3).
- Therefore, one bit of the virtual index (bit 11) comes from the virtual page number.

- (c) 这个内存系统的物理地址空间有多大？ 64 KB
  - The page offset is 11 bits.
  - tag bits = 5, VIPT 用PPN 做tag
  - The physical frame number, which is the same as the physical tag is 5 bits. Therefore, the physical address space is $2^{(11+5)} = 2^{16}$ bytes = 64 KB.

# More Problem

- If C≤(page_size × associativity), the cache index bits come only from page offset (same in VA and PA)
- If both cache and TLB are on chip
  - index both arrays concurrently using VA bits
  - check cache tag (physical) against TLB output at the end



# HW4

- 一个具有16位地址的字节可寻址系统（A byte-addressable system with 16-bit addresses），配备了一个两路组相联、写回策略的缓存，并且具有完美的LRU替换功能。标签存储（tag array）（包括标签和其他所有元数据）总共需要4352位的存储空间。缓存的块大小是多少？假设LRU信息是按set维护，占1位。
  - 4352 = 2^index ∗ (2 ∗ (tag + dirty + valid) + LRU)  Eq. (1)
  - tag + index + offset = 16  Eq. (2)
- There is one dirty bit and one valid bit per block, and one LRU bit per set. So, now the Eq. 1 looks like:
  - 4352 = 2^index ∗ (2 ∗ (tag + 2) + 1) = (2^index ∗ (2*tag + 4)) + 2^index
- By using the hint (4352 = 2^12 + 2^8), we get 2^index = 2^8, so index = 8, and
- From 2^index ∗ (2*tag + 4) = 2^12, we get (2*tag + 4) = 2^4, so tag = 6.
- By solving the Eq. 2, we get offset = 2, so, the block size is 2^2 = 4 bytes

# HW 5.5

- （Rowbuffer）对于以下内存访问模式，估计在两种不同的调度机制下每次内存访问完成的时间：open-page和close-page。假设可以重新排序已经在内存控制器中等待的请求。访问模式只指定被访问的行（row）。所有访问都是针对同一个bank。假设总线延迟为零。假设bank在时间0已经预充电（pre-charge）。假设预充电需要20 ns，加载行缓冲区（activation）需要20 ns，缓存行传输到输出引脚也需要20 ns（换句话说，行缓冲区命中（Rowbuffer-hit）需要20 ns，空行访问（Rowbuffer-empty）需要40 ns，行缓冲区冲突（Rowbuffer-conflict）需要60 ns）。

| 访问行 | 到达内存控制器时间 | Open-page | Close-page |
|---|---|---|---|
| X | 10 ns | 50 ns | 50 ns |
| X | 75 ns | 95 ns | 115 ns |
| Y | 100 ns | 160 ns | 175 ns |
| X | 190 ns | 250 ns | 235 ns |
| X | 280 ns | 300 ns | 320 ns |
| Y | 290 ns | 360 ns | 380 ns |

# More Problem

- For the following access stream, estimate the finish times for each access with the following scheduling policies:

| Req | Time of arrival | Open | Closed | Oracular |
|---|---|---|---|---|
| X | 10 ns | 50 | 50 | 50 |
| X+1 | 15 ns | 70 | 70 | 70 |
| Y | 100 ns | 160 | 140 | 140 |
| Y+1 | 180 ns | 200 | 220 | 200 |
| X+2 | 190 ns | 260 | 300 | 260 (or 285) |
| Y+2 | 205 ns | 320 | 240 | 320 (or 225) |

Note that X, X+1, X+2, X+3 map to the same row and Y, Y+1 map to a different row in the same bank. Ignore bus and queuing latencies. The bank is precharged at the start.
** A more sophisticated oracle can do even better.

# HW 5.6

- （Memory hierarchy）考虑一个系统，它有两个处理器插槽（socket）；每个插槽有6个DDR4内存通道（memory channel）。每个通道最多可以容纳4 ranks。假设你今天可以购买容量为2Gb、4Gb或8Gb的DRAM芯片。假设这些芯片可以有4、8或16的数据输出宽度（data output width）。这个内存系统可以支持的最大容量是多少？如果每个内存通道以1.2 GHz的频率运行，系统支持的内存带宽是多少？
- 
- 2 sockets x 6 channels x 4 ranks x 16 chips x 8Gb capacity = 768 GB
- 
- 2 sockets x 6 channels x 1.2G (cycles per second) x 2 (DDR, hence 2 transfers per cycle) x 64 (bits per transfer) = 230.4GB/s

# More Problems

以下服务器支持的最大内存容量是多少：
-有2个处理器插槽（processor sockets），
-每个插槽有 4 个内存通道（memory channels），
-每个通道支持 2个双排列DIMMs（dual-ranked DIMMs），
-并且是 x4 4Gb 的 DRAM 芯片？

2*4*2*2*16*4Gb/8 = 256GB

2个处理器插槽

4个内存通道

2个DIMM

2 Ranks

64 / 4 = 16 chips

Channel width = 64bits

Chip width = 4 bits, to fully utilize 64 bits output/input,
We need 16 chips

# More Problem

- What is the maximum memory capacity supported by the following server: 2 processor sockets, each socket has 4 memory channels, each channel supports 2 dual-ranked DIMMs, and x4 4Gb DRAM chips?

  2 sockets x 4 channels x 2 DIMMs x 2 ranks x
  16 chips x 4Gb capacity = 256 GB

  What is the memory bandwidth available to the server if each memory channel runs at 800 MHz?
  2 sockets x 4 channels x 800M (cycles per second) x
  2 (DDR, hence 2 transfers per cycle) x 64 (bits per transfer)
  = 102.4 GB/s

# HW6.1

- （基于监听的缓存一致性Snoop-based Cache Coherence (CC)）考虑一个对称共享内存多处理器系统（symmetric shared memory multiprocessor model）（3个处理器共享一个总线），实现了一种监听式缓存一致性协议（Snoop-based CC with write invalidation）。对于下面的每个事件，解释一致性协议的步骤（缓存是否标记命中/未命中，总线上放置了什么请求，谁响应，是否需要写回等），并提及每个处理器缓存中数据块的最终状态。假设在序列开始时，X和Y不在任何缓存中，缓存是直接映射(Direct-mapped Cache)的，并且块X和Y在每个缓存中映射到同一组（set）（在任何时候，缓存中不能同时存在X和Y）。请完成以下表内容。

  - P1：写入X
  - P2：写入X
  - P3：读取X
  - P1：读取X
  - P3：写入X
  - P3：读取Y
  - P2：写入Y

# HW6.1

| Request | Cache Hit/Miss | Request on the bus | Who responds | State in Cache 1 | State in Cache 2 | State in Cache 3 |
|---|---|---|---|---|---|---|
| | | | | Inv | Inv | Inv |
| P1: Wr X | Miss | Wr req X | Memory | M(X) | Inv | Inv |
| P2: Wr X | Miss | Wr req X | P1 | Inv | M(X) | Inv |
| P3: Rd X | Miss | Rd req X | P2 responds (Memory Writeback) | Inv | S(X) | S(X) |
| P1: Rd X | Miss | Rd req X | Memory | S(X) | S(X) | S(X) |
| P3: Wr X | Permissio n miss (or Hit) | Upgrade X | No response. Other caches invalidate | Inv | Inv | M(X) |
| P3: Rd Y | Miss | Rd req Y | Memory | Inv | Inv | S(Y) |
| P2: Wr Y | Miss | Wr req Y | Memory | Inv | M(Y) | Inv |

# HW6.2

- 基于目录的缓存一致性(Directory-based Cache Coherence (CC))考虑与问题1相同的内存访问序列。假设4个处理器(P1, P2, P3和P4)通过点对点互联网络连接，并实现带有目录的分布式共享内存(Distributed shared memory multiprocessor model)和基于目录的缓存一致性协议(Directory-based CC with write invalidation)。对于上述指令序列，基于写无效协议，互联消息传输的总数是多少？
- 对于每条指令，列出必须在网络上发送的消息以及缓存和目录中缓存块或者行的状态(参考例子)。假设一个消息可以包括一些控制信息以及地址和缓存行。还假设内存位置X和Y的主节点都与处理器P4关联。假设在序列开始时，X和Y不在任何缓存中，缓存是直接映射的，并且块X和Y在每个缓存中映射到同一组(在任何时候，缓存中不能同时存在X和Y)。

# HW6.1

| Request | Cache Hit/Miss | Messages | Directory State | State in Cache 1 | State in Cache 2 | State in Cache 3 |
|---|---|---|---|---|---|---|
| | | | X: S:<br>Y: S: | Inv | Inv | Inv |
| P1: Wr X | Miss | Wr-req to Dir in P4.<br>Dir responds with the copy | X: M:1<br>Y: S: | M(X) | Inv | Inv |
| P2: Wr X | Miss | Wr-req to Dir in P4. Inv to P1. P1 sends data to Dir. Dir sends data to P2 | X: M:2<br>Y: S: | Inv | M(X) | Inv |
| P3: Rd X | Miss | Rd-req to Dir. Dir forwards req to P2. P2 sends to data to Dir. Memory writeback. Dir sends data to P3 | X: S:2,3<br>Y: S | Inv | S(X) | S(X) |
| P1: Rd X | Miss | Rd-req to Dir. Dir responds P1 with data | X:S:1,2,3<br>Y: S | S(X) | S(X) | S(X) |
| P3: Wr X | Permission Miss | Upgrade-req to Dir. Dir sends Inv to P1 and P2, and grant to P3 | X:M:3 | Inv | Inv | M(X) |
| P3: Rd Y | Miss | Rd-req sends to Dir. Dir responds P3 with the copy | X:S:<br>Y:S:3 | Inv | Inv | S(Y) |
| P2: Wr Y | Miss | Wr-req sends to Dir. Dir sends Inv to P3, and forwards data to P2. | X:S:<br>Y:M:2 | Inv | M(Y) | Inv |

# More Example

- 考虑一个对称的共享内存多处理器系统(4个处理器共享一个总线)，实现类似于课堂上讨论过的Snooping(MSI)缓存一致性协议。对于以下的每个事件，请解释一致性协议的步骤(缓存是否命中/未命中，总线上发送了什么请求，谁会响应，是否需要写回等)，并提及每个处理器缓存中数据块的最终状态。假设在序列开始时，X 不在任何缓存中。已经给出第一步，请参考第一步完成以下表格。P 表示 Processor，C 表示 Cache。

| Request | Cache Hit/Miss | Request on the bus | Who responds | State in Cache 1 | State in Cache 2 | State in Cache 3 | State in Cache 4 |
|---|---|---|---|---|---|---|---|
| | | | | Inv | Inv | Inv | Inv |
| P1: Rd X | Miss | Rd X | Memory | S | Inv | Inv | Inv |
| P2: Rd X | Miss | Rd X | Memory | S | S | Inv | Inv |
| P2: Wr X | Perms Miss | Upgrade X | No response. Other caches invalidate. | Inv | M | Inv | Inv |
| P3: Wr X | Write Miss | Wr X | P2 responds | Inv | Inv | M | Inv |
| P3: Rd X | Read Hit | - | - | Inv | Inv | M | Inv |
| P4: Rd X | Read Miss | Rd X | P3 responds. Mem wrtbk | Inv | Inv | S | S |

# More Example

- 假设有 4 个处理器通过点对点互连相连，并使用基于目录的缓存一致性协议实现分布式共享内存。对于以下指令序列，对于每条指令，列出必须发送到网络的消息以及缓存和目录中线的状态。假设消息可以包括一些控制信息以及地址和缓存行。假设在序列开始时，X 不在

| Request | Cache Hit/Miss | Messages | Directory State | State in C1 | State in C2 | State in C3 | State in C4 |
|---|---|---|---|---|---|---|---|
| | | | | Inv | Inv | Inv | Inv |
| P1: Rd X | Miss | Read request to Directory Directory responds. | X: S: 1 | S | Inv | Inv | Inv |
| P2: Rd X | Miss | Rd-req to Dir. Dir responds. | X: S: 1, 2 | S | S | Inv | Inv |
| P2: Wr X | Perms Miss | Upgr-req to Dir. Dir sends INV to P1. P1 sends ACK to Dir. Dir grants perms to P2. | X: M: 2 | Inv | M | Inv | Inv |
| P3: Wr X | Write Miss | Wr-req to Dir. Dir fwds request to P2. P2 sends data to Dir. Dir sends data to P3. | X: M: 3 | Inv | Inv | M | Inv |
| P3: Rd X | Read Hit | - | - | Inv | Inv | M | Inv |
| P4: Rd X | Read Miss | Rd-req to Dir. Dir fwds request to P3. P3 sends data to Dir. Memory wrtbk. Dir sends data to P4. | X: S: 3, 4 | Inv | Inv | S | S |

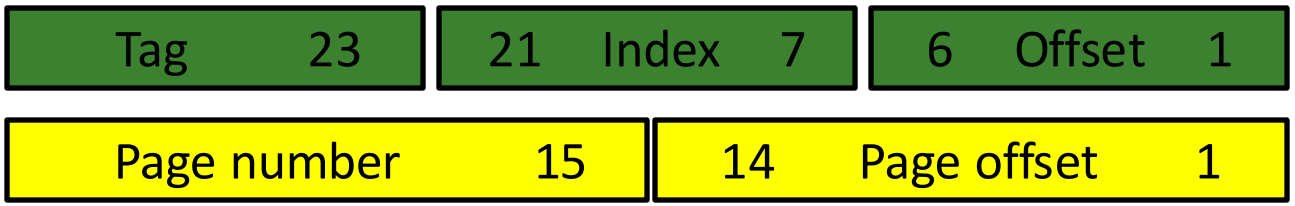
# SMP Example

| | A | B | C | |
|---|---|---|---|---|
| A: Rd X | S | | | Rd-miss req; mem responds |
| B: Rd X | S | S | | Rd-miss req; mem responds |
| C: Rd X | S | S | S | Rd-miss req; mem responds |
| A: Wr X | M | I | I | Upgrade req; no resp; others inv |
| A: Wr X | M | I | I | Cache hit |
| C: Wr X | I | I | M | Wr-miss req; A resp & inv; no wrtbk |
| B: Rd X | I | S | S | Rd-miss req; C resp; wrtbk to mem |
| A: Rd X | S | S | S | Rd-miss req; mem responds |
| A: Rd Y | S (Y) | S (X) | S (X) | Rd-miss req; X evicted; mem resp |
| B: Wr X | S (Y) | M (X) | I | Upgrade req; no resp; others inv |
| B: Rd Y | S (Y) | S (Y) | I | Rd-miss req; mem resp; X wrtbk |
| B: Wr X | S (Y) | M (X) | I | Wr-miss req; mem resp; Y evicted |
| B: Wr Y | I | M (Y) | I | Wr-miss req; mem resp; others inv; X wrtbk |

# Directory Example

| | A | B | C | Dir | Comments |
|---|---|---|---|---|---|
| A: Rd X | S | | | S: A | Req to dir; data to A |
| B: Rd X | S | S | | S: A, B | Req to dir; data to B |
| C: Rd X | S | S | S | S: A,B,C | Req to dir; data to C |
| A: Wr X | M | I | I | M: A | Req to dir;inv to B,C;dir recv ACKs;perms to A |
| A: Wr X | M | I | I | M: A | Cache hit |
| C: Wr X | I | I | M | M: C | Req to dir;fwd to A; sends data to dir; dir to C |
| B: Rd X | I | S | S | S: B, C | Req to dir;fwd to C;data to dir;dir to B; wrtbk |
| A: Rd X | S | S | S | S:A,B,C | Req to dir; data to A |
| A: Rd Y | S(Y) | S | S | X:S: A,B,C (Y:S:A) | Req to dir; data to A |
| B: Wr X | S(Y) | M | I | X:M:B | Req to dir; inv to A,C;dir recv ACK;perms to B |
| B: Rd Y | S(Y) | S(Y) | I | X: - Y:S:A,B | Req to dir; data to B; wrtbk of X |
| B: Wr X | S(Y) | M(X) | I | X:M:B Y:S:A,B | Req to dir; data to B |
| B: Wr Y | I | M(Y) | I | X: - Y:M:B | Req to dir;inv to A;dir recv ACK; perms and data to B;wrtbk of X |

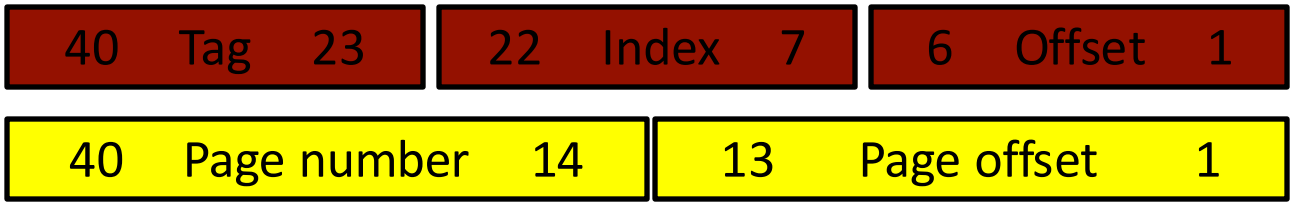
# HW 6.5

- 大型缓存(Large Cache)假设有一个大型共享的LLC(最后一级缓存)，它是tiled分布的，并分布在处理器芯片上。假设操作系统的页面大小为16KB。整个LLC的大小为32MB，使用64字节的缓存块，并且是16路组相联。那么最大的tile数量是多少，以便操作系统可以完全灵活地在其选择的tile中放置页面？
- 
- Number of cache blocks = 2^25 / 2^6= 2^19
- Number of sets = 2^19 / 2^4 = 2^15, thus we have 15 bits for set indexing, and 6 bits for block offset, total 21 bits, while 14 bits for page size.
- We can use all 21 – 14 = 7 bits for tile mapping, so we have 2^7 =128 tiles at most.
- 


  - 所有tile一共要32MB/64B/16=32768组，然后每个页面要16KB/64B=256块
  - 所以至少要32768 / N >= 256保证放的下

# More Problem

- Assume a large shared LLC that is tiled and distributed on the chip. Assume 16 tiles. Assume an OS page size of 8KB. The entire LLC has a size of 32 MB (2^25), uses 64-byte (2^6) blocks, and is 8 (2^3)-way set-associative. Which of the 40 physical address bits are used to specify the tile number? Provide an example page number that is assigned to tile 0.




The cache has 64K sets (2^25/2^6/2^3=2^16), i.e., 6 block offset bits, 16 index bits, and 18 tag bits. The address also has a 13-bit page offset, and 27 page number bits. Nine bits (bits 14-22) are used for the page number and the index bits. Any four of those bits can be used to designate the tile number, say, bits 19-22. An example page number assigned to tile 0 is xxx…xxx0000xxx…xxx

bit 22 19

# HW7.1

- Memory Consistency，考虑在一个实现了顺序一致性(sequential consistency)的多核处理器上，使用两个不同的线程分别(并行地)运行以下两个C程序。

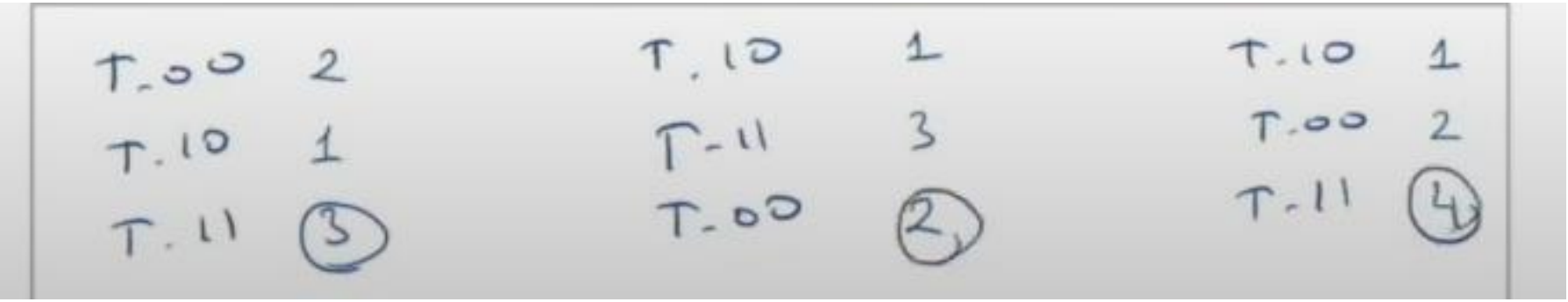
| Thread T0 | Thread T1 |
|---|---|
| Instr. T0.0 X[0] = 2; | Instr. T1.0 X[0] = 1; |
| Instr. T0.1 flag[0] = 1; | Instr. T1.1 X[0] += 2; |
| Instr. T0.2 a = X[0] * 2; | Instr. T1.2 while(flag[0] == 1); |
| Instr. T0.3 b = Y[0] − 1; | Instr. T1.3 a = flag[0]; |
| Instr. T0.4 c = X[0]; | Instr. T1.4 X[0] = 2; |
| | Instr. T1.5 Y[0] = 10; |

- 其中 X 和 flag 已经分配在内存中，T0 和 T1 使用处理器每个核心的私有寄存器来存储变量 a, b, c 的值，对任意变量的读或写都会产生一次内存请求(memory request)。现在，内存上的所有位置以及所有变量的初始值是1. 假设上面C程序的每一行代码都对应单条指令。

- (a)说明上面的C程序可能存在的问题。
- Thread 1 will never finish.
- Explanation:
- The while loop in instruction T1.2 is an infinite loop, because the value of flag[0] is 1 since the beginning of the program.

# HW7.1

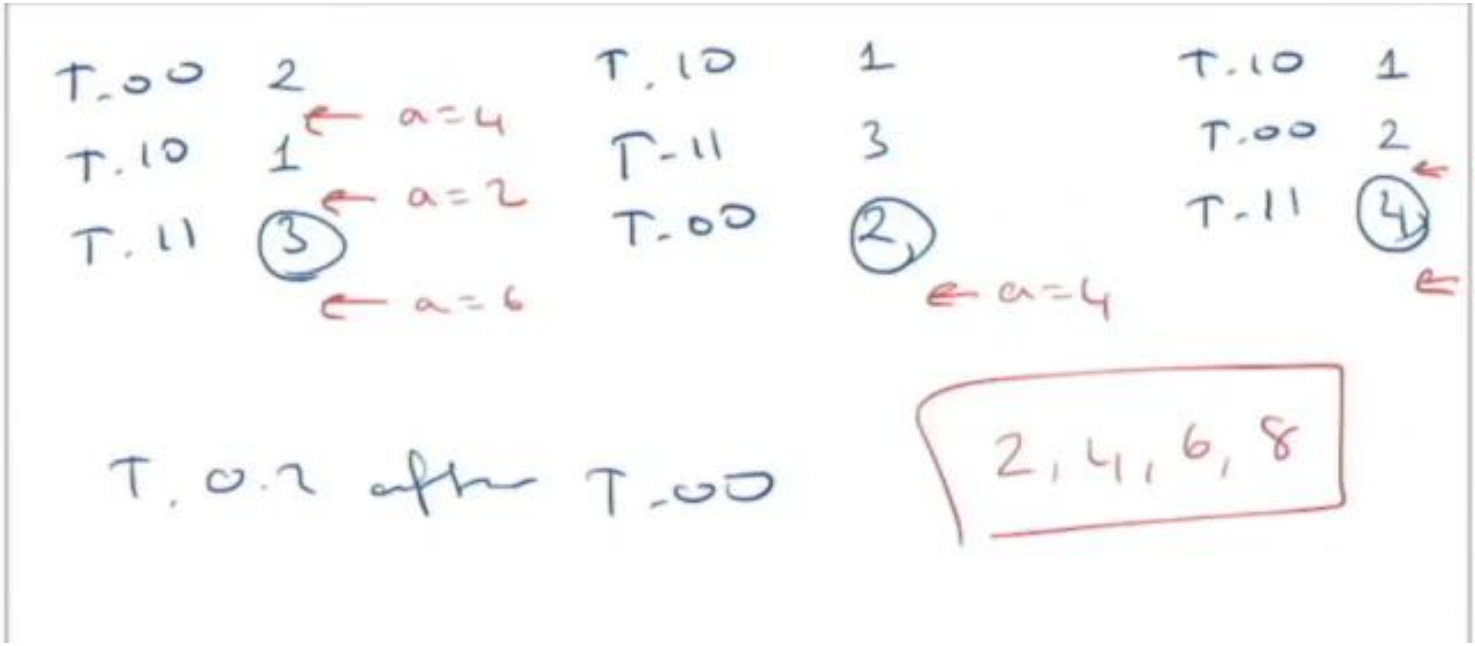
- (b) 在题设的处理器上运行这个程序，X[0] 的最终值有哪些可能？给出所有可能值并解释出现该值时程序的运行情况。




- Ordering 1: T1.0 → T1.1 → T0.0, Final value: X[0] = 2.
- Ordering 2: T0.0 → T1.0 → T1.1, Final value: X[0] = 3.
- Ordering 3: T1.0 → T0.0 → T1.1, Final value: X[0] = 4.

# HW7.1

- c) 在题设的处理器上运行这个程序，a的最终值有哪些可能？ 给出所有可能值并解释出现该值时程序的运行情况。
- 2, 4, 6, or 8.



# HW7.1

- d) 在题设的处理器上运行这个程序，b 的最终值有哪些可能？给出所有可能值并解释出现该值时程序的运行情况。

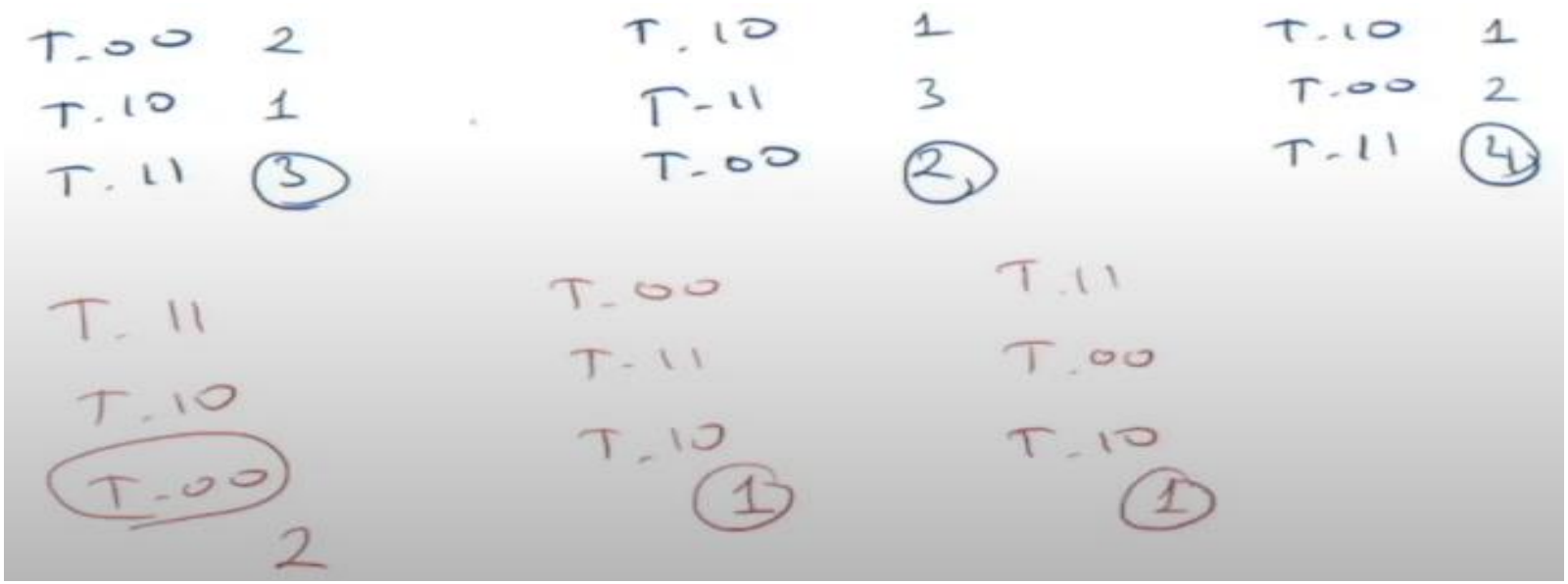
- 0.

- Explanation: Because the value of b depends only on the value of Y[0] (instruction T0.3). The initial value of Y[0] is 1. Instruction T1.5 will not be executed as T1 enters an infinite loop after executing instruction T1.2.

# HW7.1

- e) 考虑在另一个没有实现内存一致性(memory consistency)的多核处理器上运行这个程序, X[0] 的最终值有哪些可能？给出所有可能值并解释出现该值时程序的运行情况。

- 1, 2, 3, or 4.



# More Problem

- What are possible outputs for the program below?

Assume x=y=0 at the start of the program

| | Thread 1 | | Thread 2 |
|---|---|---|---|
| A | x = 10 | a | y=20 |
| B | y = x+y | b | x = y+x |
| C | Print y | | |

Possible scenarios: 5 choose 2 = 10

| ABCab | ABaCb | ABabC | AaBCb | AaBbC |
|---|---|---|---|---|
| 10 | 20 | 20 | 30 | 30 |
| AabBC | aABCb | aABbC | aAbBC | abABC |
| 50 | 30 | 30 | 50 | 30 |

# HW8.3

- (GPUs and SIMD)我们将程序在GPU上运行的SIMD利用率定义为：某程序运行期间繁忙的SIMD通道(busy SIMD lanes)数量比上该程序运行使用的线程数量。现在在GPU上运行程序P，每个线程执行程序P中循环的单次迭代(包含两条指令)

```
for (i = 0; i < N; i++) {
    if (A[i] % 3 == 0) {      // Instruction 1
       A[i] = A[i] * B[i];    // Instruction 2
    }
}
```

- 假设数组A和B的数据已经位于向量寄存器中（此时执行程序P中不需要加载和存储操作）。假设GPU的一个wrap有32个线程，GPU有32个SIMD通道。假设每条指令花费的时间相同。
- 用N表示执行程序P所需要的wrap数量?

- N iterations, warp size 32,
- So Ceiling(N/32)

# HW8.3

- 假设整数数组A具有重复模式，24个1后跟8个0重复出现，而整数数组B具有另一种重复模式，48个0后跟64个1。此时程序P的SIMD利用率是多少？
  - 11111…11111(24 1s)00000000
  - (32 + 8) / (32+32) = 0.625
  - Or ((24+8×2)/(32×2)) = 40/64 = 0.625
- 程序P的SIMD的利用率有可能达到100%么？如果有可能，请给出数组A和B满足的条件；如果不可能，请给出理由。
  - Yes, Starting from A[0], consecutive 32 elements of A should be either divisible by 3 or not divisible by 3.
  - B can be any array of integers.

# HW8.3

- 程序P的SIMD的利用率有可能达到56.25%么？如果有可能，请给出数组A和B满足的条件；如果不可能，请给出理由。
  - Yes,
  - 4 out of every 32 elements of A are divisible by 3
  - B can be any array of integers.

(32+x)/(32*2) = 0.5625, x=4

- 程序P的SIMD的利用率有可能达到50%么？如果有可能，请给出数组A和B满足的条件；如果不可能，请给出理由。
  - the minimum is when only 1 out of every 32 elements of A is divisible by 3. This yields a 51.5625% usage.
  - 最差的利用率是只有一个thread 进入 if branch

# HW8.1

- （Vector Processing）现在有一台向量计算机VC，它的指令延迟如下：
  - VLD 和 VST：每个向量元素（Vector Element）开销为50个周期，支持fully interleaved and pipelined
  - VADD：每个向量元素开销为4个周期（fully pipelined）
  - VMUL：每个向量元素开销为16个周期（fully pipelined）
  - VDIV：每个向量元素开销为32个周期（fully pipelined）
  - VRSHFA：每个向量元素开销为1个周期（fully pipelined）

- 现在假设：

1. VC所支持的流水线是有序流水线（In-Order Pipeline）。
2. VC支持向量功能单元之间的链式（Chaining）操作。
3. 为支持向量元素的单周期内存访问，VC将向量元素交错存储在内存的多个bank中，一个向量中的多个元素在内存中的排布如下：第一个元素映射到Bank 0，第二个元素映射到Bank 1，依此类推。
4. 每个Bank有一个8 KB的行缓冲区（Row Buffer）。
5. 向量元素大小为64位。
6. 每个Bank有两个端口（支持允许加载/存储操作并行），并且有两个加载/存储功能单元可用。

# HW8.1

- 根据上述条件回答以下问题:
- 为了使内存访问永不阻塞，Bank数量（2的幂次）至少是多少？（假设向量步长为1）
  - 64 banks (because memory latency is 50 cycles and the next power of two is 64)
- 假设VC的Bank数量如 (a) 所描述，且执行下列程序P需要花费111个时钟周期:

```
VLD    V1, A        // V1   ← A
VLD    V2, B        // V2   ← B
VADD   V3, V1, V2   // V3   ← V1 + V2
VMUL   V4, V3, V1   // V4_i ← V3_i × V1_i
VRSHFA V5, V4, 2    // V5_i ← V4_i >>> 2
```

# HW8.1

- 请问向量长度L（向量中的元素个数）是多少?

- L = 40
- 
- VLD | - - - -50 - - - - - - -| - - -(L -1) - - - |
- VLD |1| - - - -50 - - - - - - -|
- VADD | -4 -| --- (L-1)--- |
- VMUL | -16 -| --- (L-1)--- |
- VRSHFA |1| --- (L-1) --- |
- 
- 1 + 50 + 4 + 16 + 1 + (L – 1) = 71 + L = 111
- L = 40

# HW8.1

- 若衍生型号VC-SE不支持向量功能单元之间的链式操作，但仍具有VC的其他特性。请问VC-SE执行程序P需要多少个时钟周期?

- 228 cycles
- VLD | - - - - -50 - - - - -| - - -(L -1) - - -|
- VLD |1| - - - - -50 - - - - -| - - -(L -1) - - -|
- VADD | -4 -| - -( L -1) - - -|
- VMUL | -16 -| - -( L -1) - - -|
- VRSHFA |1|-- ( L -1) - -|
-

- $1 + 50 + 4 + 16 + 1 + 4 \times (L-1) = 68 + 4 \times L = 228$ cycles

# HW8.1

- 若衍生型号VC-Mini将内存的bank数量相比于 (a) 中的描述砍了一半，其余特性和VC保持一致。此时对内存中的向量访问会产生阻塞，所以每个Bank上增加了仲裁器以使得最早的访问被最先处理。请问在VC-SE上执行程序P需要多少时钟周期?
-
- 129 cycles
- VLD [0] | - - - -50 - - - -| bank 0 (takes port 0)
- …
- [31] | - -31 - -| - - - -50 - - - -| bank 31
- [32] | - - -50 - - -| bank 0 (takes port 0)
- ……
- [39] | - -7 - -| with bank 7
- VLD [0] |1| - - - -50 - - - -| bank 0 (takes port 1)
- …
- [31] |1| - -31 - -| - - - -50 - - - -| bank 31
- [32] | - - -50 - - - -| bank 0 (takes port 1)
- …
- [39] | - -7 - -| with bank 7
- VADD | - -4 - -| (tracking last elements)
- VMUL | - -16 - -|
- VRSHFA |1|
- (B[39]: 1 + 50 + 50 + 7) + 4 + 16 + 1 = 129 cycles
- 2*50 + 7 + 1 + 4 + 16 + 1 = 129 cycles

# HW8.1

- 若VC-Tiny进一步缩减bank数量（但始终是2的幂次），使得执行完程序P需要279个时钟周期。请问VC-Tiny上有多少个Bank数量?

VLD [0] | - - -50 - - -|

...

[8] | - - -50 - - -| Bank 0

...

[16] | - -50 - -| Bank 0

...

[24] | - -50 - -| Bank 0

...

[32] | - -50 - -| Bank 0

...

[39] | - -7 - -| Bank 7

VLD [39] |1|

VADD | - -4 - -|

VMUL | - -16 - -|

VRSHFA |1|

- X * 50 + 7+1+16+1=279 => X=5 = 40/5 = 8 Banks
- 5 × 50 + 7 + 1 + 4 + 16 + 1 = 279 cycles => 8 banks

# HW8.1

- 若VC-Ultra-100支持多核处理，其具有4个向量处理器，共享一个内存系统，bank数量是VC的4倍。现在在VC-Ultra-100的每个核心上运行测试程序，发现每个核心消耗的时间甚至比单核VC配1/4数量的bank还要多。请问为什么会出现这种情况？

  - Row-buffer conflicts (all cores interleave their vectors across all banks).

- 若VC-Ultra-200只改变VC-Ultra-100的共享内存架构，请问需要怎么做缓解 (f) 中出现的情况？

  - Partition the memory mappings, or using better memory scheduling.

# HW8.2

- （SIMD Processing）现在希望设计一个能够支持向量长度为16的SIMD处理器，考虑以下两个方案：传统的向量处理器和传统的阵列处理器。
- 哪种方案芯片面积最大？请给出理由。
  - An array processor requires 16 functional units for an operation whereas a vector processor requires only 1.
- 假设两种处理器的加法操作延迟都是5周期，且加法器是fully pipelined的，请计算并给出理由：
- 考虑向量长度是1，两种处理器执行VADD操作花费的时钟周期？

- The traditional vector processor:
- 5 cycles

- The traditional array processor:
- 5 cycles

- 考虑向量长度是4，两种处理器执行VADD操作花费的时钟周期?

  - The traditional vector processor:
  - 8 cycles (5 for the first element to complete, 3 for the remaining 3)

  - The traditional array processor:
  - 5 cycles

- 考虑向量长度是16，两种处理器执行VADD操作花费的时钟周期?

  - The traditional vector processor:
  - 20 cycles (5 for the first element to complete, 15 for the remaining 15)

  - The traditional array processor:
  - 5 cycles

# HW8.5

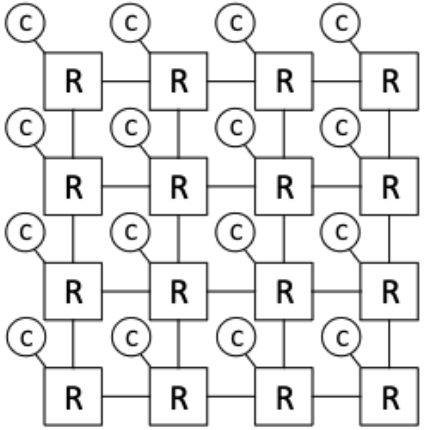
- （Interconnects）考虑连接一个包含$2^N$处理器的系统，使用下面三种拓扑结构：

i. $\sqrt{2^N} \times \sqrt{2^N}$ 2D mesh
ii. $\sqrt{2^N} \times \sqrt{2^N}$ 2D torus
iii. Hypercube

- 请回答下列问题：

(a) 绘制N=4时候每种拓扑对应的网络结构图（适当使用省略号简化绘图）

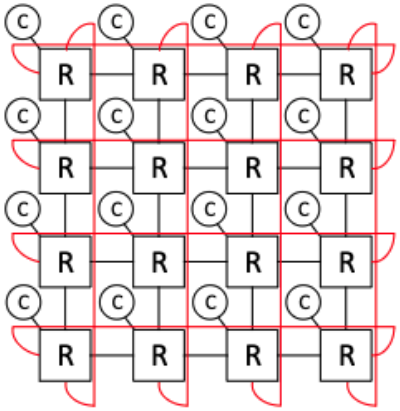
**2D Mesh:**



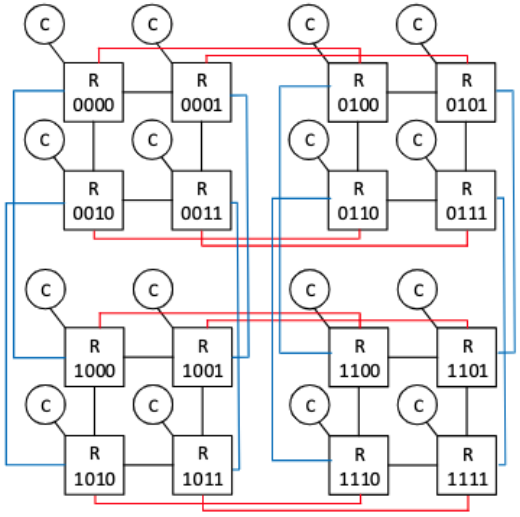
- 请回答下列问题:

(a) 绘制N=4时候每种拓扑对应的网络结构图（适当使用省略号简化绘图）

**2D Torus:**




**Hypercube:**



- (b) 计算N=8时候每种拓扑的网络链路数量（每个链接时双向的）。

$$\text{2D mesh: } 2 \times (\sqrt{2^N} - 1)(\sqrt{2^N}) \text{ links} \rightarrow 2 \times 15 \times 16 = 480 \text{ links}$$
$$\text{2D torus: } 2 \times (\sqrt{2^N})(\sqrt{2^N}) \text{ links} \rightarrow 2 \times 16 \times 16 = 512 \text{ links}$$
$$\text{Hypercube: } 2^N \times N/2 \text{ links} \rightarrow 256 \times 8/2 = 1024 \text{ links}$$

  - 16 nodes 15 links for mesh, 16 nodes 16 links for torus

- (c) 计算N=8时候每种拓扑中的输入/输出端口数量（包括router的端口）。对于irregular network，还需要给出每种类型router的端口数量。
  - 2D mesh: (4+1) inputs/outputs, (3+1) inputs/outputs, and (2+1) inputs/outputs
  - 2D torus: (4+1) inputs/outputs
  - Hypercube: N+1 inputs/outputs => 9 inputs/outputs

# HW 7.3

- 考虑网络N中有2048个处理器，现在有两种网络拓扑方案来连接这些处理器：

A. 双向环形总线（Bi-directional Ring）

B. 超立方体（Hypercube）

对于下列问题，分别计算网络N在两种拓扑下的答案：网络N中的链接数量（认为每个链接是双向的）。

- Ring: 2048
- Hypercube: 2^k × k/2 links -> 2048 × 11/2 = 11264 links

- 网络N中的最大路由距离(即网络直径)，按照跃点数(hop count)计算。

- Ring: 2048 / 2 = 1024 hops
- Hypercube: Log(2048) = 11 hops
  - 2^11, 11 bits to represent a node, so a node has at most of 11 bits difference

- 若网络N中存在至少一个处理器不能访问其它所有处理器，则最少有多少个链接发生了故障？
- Ring: 2 Links
- Hypercube: Log(2048) links -> 11 links
  - Each Hypercube node has 11 paths, so we need to break them all

# More Problem

- We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program. The SIMD utilization of a program is computed across the *complete run* of the program. The following code segment is run on a GPU. Each thread executes a single iteration of the shown loop. Assume that the data values of the arrays A, B, and C are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 6 instructions in each thread.) A warp in the GPU consists of 64 threads, and there are 64 SIMD lanes in the GPU. **Assume that all values in array B have magnitudes less than 10 (i.e., |B[i]| < 10, for all i).**

```
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;
    }
}
```

# More Problem

- Q1: How many warps does it take to execute this program?

  16 warps

  Number of Warps = $\lceil$(Number of threads) / (Number of threads per warp)$\rceil$
  Number of threads = $2^{10}$ (i.e., one thread per loop iteration).
  Number of threads per warp = $64 = 2^6$ (given).
  $\therefore$ Number of Warps = $\lceil 2^{10}/2^6 = 2^4 \rceil$

- Q2: What is the maximum possible SIMD utilization of this program?
  - 100%
- Q3: Please describe what needs to be true about array B to reach the maximum possible SIMD utilization asked in part (b).
  - All 0s, negatives or positives

# More Problem

- Q4: What is the minimum possible SIMD utilization of this program?
  - $(64 \times 2 + 1 \times 4)/(64 \times 6) = 132/384$

- Q5: Please describe what needs to be true about array B to reach the minimum possible SIMD utilization asked in part (d).
  - Exactly 1 of every 64 consecutive elements must be negative.
  - The others are 0

# More Problem

- For the following access stream, estimate the finish times for each access with the following scheduling policies:

| Req | Time of arrival | Open | Closed | Oracular |
|---|---|---|---|---|
| X | 0 ns | 40 | 40 | 40 |
| Y | 10 ns | 100 | 100 | 100 |
| X+1 | 100 ns | 160 | 160 | 160 |
| X+2 | 200 ns | 220 | 240 | 220 |
| Y+1 | 250 ns | 310 | 300 | 290 |
| X+3 | 300 ns | 370 | 360 | 350 |

Note that X, X+1, X+2, X+3 map to the same row and Y, Y+1 map to a different row in the same bank. Ignore bus and queuing latencies. The bank is precharged at the start.

# 关于期末考试与成绩

- 考试地点
  - 教一楼 207
- 考试时间
  - 2025年1月7日，13:30 - 15:30，考试时长：120分钟
- 考试形式
  - 闭卷（无需携带计算器等设备）
- 考试题型与分值分布
  - 单项选择题30分，问答与计算题70分
- 总成绩=平时成绩*40% + 期末卷面成绩*60%
  - 平时成绩包括:作业、演讲汇报及简报、出勤

# 题目分布

- 10 单选题 + 6 - 8道计算题
  - 高婉铃：5道选择题，3 - 4道计算题
  - 刘珂： 5道选择题，3 - 5道计算题

祝大家新年快乐，考试顺利，未来繁花似闪星，科研摘硕果

谢谢！

# Acknowledgements


- EPFL, Onur Mutlu, Digital Design and Computer Architecture, 2020, 2023
- Utah University, Rajeev, CS/ECE 6810, Computer Architecture
- 计算机体系结构:量化研究方法(中文版第六版)
- UCSD CSE 240
- Washington University, CSE3 78
- 国科大, 胡伟武, 计算机体系结构
- CMU, CS 447 Computer Architecture
- UC Berkeley, CS 152 and CS 61C
- 国科大南京学院, 安学军等, 计算机体系结构
- 浙江大学, 计算机体系结构
- 南京大学, 计算机体系结构