Computer Architecture I Final Exam

Chinese Name: ______________________________

Pinyin Name: ______________________________

Student ID: ______________________________

E-Mail prefix: ______________________________

Question	Points	Score
1	1	
2	22	
3	11	
4	10	
5	20	
6	16	
7	8	
8	7	
9	5	
Total:	100	

- This test contains 28 numbered pages, including the cover page, printed on both sides of the sheet.
- We will use GradeScope for grading, so only answers filled in at the obvious places will be used.
- Use the provided blank paper for calculations and then copy your answer here.
- Please turn **off** all cell phones, smartwatches, and other mobile devices. Remove all hats and headphones. Put everything in your backpack. Place your backpacks, laptops and jackets out of reach.
- Unless told otherwise always assume a 32-bit machine.
- The total estimated time is 120 minutes.

- You have 120 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use one A4 page (front and back) of handwritten notes in addition to the provided green sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided.
- Do **NOT** start reading the questions/open the exam until we tell you so!

1 1. First Task: **Fill in you name**.
Fill in your name and email on the front page and your ShanghaiTech email on top of every page (without @shanghaitech.edu.cn) (so write your email in total 28 times).

2. **MISC.**

1 (a) Which of the following(s) is (are) not ISA? ( )

A. RISC-V.
B. PowerPC.
C. X86.
D. ARM.
E. Linux.

Solution: E: 1 point; B and E: 0.5 point; all the other cases: 0. PowerPC is an ISA. You must at least select the correct answer E to get points.

1 (b) Which of the following two pieces of assembly code are likely to be from a RISC ISA? ( )

A.
```
2c:   00a405bb   addw    a1,s0,a0
30:   00000537   lui     a0,0x0
34:   00050513   mv      a0,a0
38:   00000097   auipc   ra,0x0
3c:   000080e7   jalr    ra # 38 <main+0x38>
```

B.
```
0000000140002990 <main>:
140002990:  53                   push %rbx
140002991:  48 83 ec 20          sub  $0x20,%rsp
140002995:  e8 86 eb ff ff       call 140001520 <__main>
14000299a:  ba e1 10 00 00       mov  $0x10e1,%edx
14000299f:  b9 d2 04 00 00       mov  $0x4d2,%ecx
1400029a4:  e8 a7 ea ff ff       call 140001450 <add>
```

C.
```
100003edc: e0 0f 00 b9  str w0, [sp, #12]
100003ee0: e1 0b 00 b9  str w1, [sp, #8]
100003ee4: e8 0f 40 b9  ldr w8, [sp, #12]
100003ee8: e9 0b 40 b9  ldr w9, [sp, #8]
100003eec: 08 01 09 6b  subs w8, w8, w9
```

Solution: A, C. A or C get 0.5 point. All the other cases 0 point. A and C are selected since RISC has equal instruction size, generally. They are actually assembly from RISC-V and ARM assembly/ISA, respectively.

1 (c) **Single choice**. What is the **main** purpose of a cache in a computer system? ( )

A. To improve the dependability of memory hierarchy via redundancy.
B. To provide additional storage space for program and data.
C. To access different parts of the memory.
D. To improve memory efficiency by storing frequently used data closer to CPU core.
E. To provide additional processing power to the CPU for more complex calculations.

Solution: D.

$\boxed{1}$ (d) **(True or False)** When a processor's pipeline is flushed to load the handler for an I/O interrupt, the processor's translation lookaside table (TLB) is flushed too. (　　)

Solution: True

$\boxed{1}$ (e) **(True or False)** MapReduce is a programming model for processing and generating large data sets. Users specify a Map function that merges key/value (KV) pairs to generate a set of KV pairs, and a Reduce function that distributes KV pairs in multiple nodes for computations. (　　)

Solution: False

$\boxed{1}$ (f) **(True or False)** Spectre attacks involve inducing a victim to speculatively perform operations that would not occur during correct program execution but leak the victim's confidential information via a side channel to the adversary. (　　)

Solution: True

$\boxed{1}$ (g) **(True or False)** The direct memory access (DMA) technique enables transferring data between memory and I/O device without the intervention of processor. (　　)

Solution: True

$\boxed{1}$ (h) **(True or False)** RAID 10 offers higher reliability than RAID 01, as you can still access your files even one of the disks in the array fails with RAID 10. (　　)

Solution: True

$\boxed{1}$ (i) **(Fill in the blank)** Assume that you receive a codeword 0x174. It was encoded with Hamming ECC but one bit error has happened in this codeword. After you follow the principle of Hamming ECC for correction, the original dataword is 0x__________ (in the hexadecimal format).

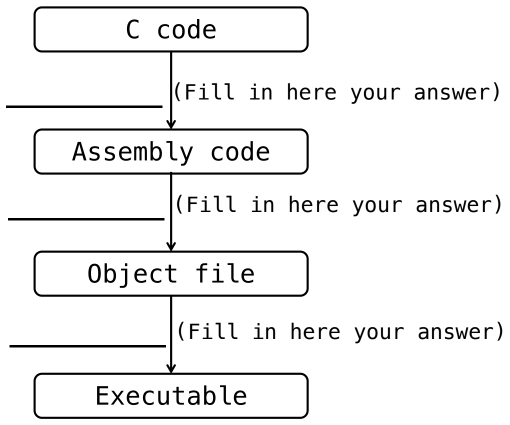
Solution: 24.

$\boxed{1}$ (j) We have two CPUs, and they are exactly the same. They run the same C program successfully, however, the average CPI (cycle per instruction) can be different. It could be the most likely due to the use of different (　　　　).

A. Nonsense. The CPI must be the same.
B. Working frequencies (e.g., by using overclocking).
C. Compilers.
D. ISAs.

Solution: C. B changes the clock cycle only. For D, since the CPUs are exactly the same, they must use the same ISA. For C, an example (not very accurate) is to use -O3 or without -O3, they can produce different numbers of different types of instructions, thus affecting the CPI. This is more true when we use different compilers (e.g., clang vs. gcc or different versions of gcc may produce different assembly instructions).

4 (k) Please fill the tools that fit the most in the following blanks to complete the translation procedure from **`C`** code to binary executable.

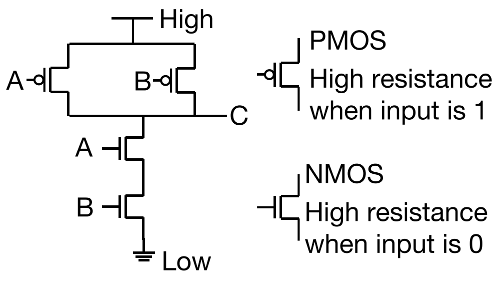

C code
(Fill in here your answer)
Assembly code
(Fill in here your answer)
Object file
(Fill in here your answer)
Executable

A. **`Loader`**.
B. **`Linker`**.
C. **`Assembler`**.
D. **`Compiler`**.

Solution: D, C, B from top to bottom, 1 point each. Extra 1 point for not having **A. Loader**. We do not need the loader to generate the executable file.

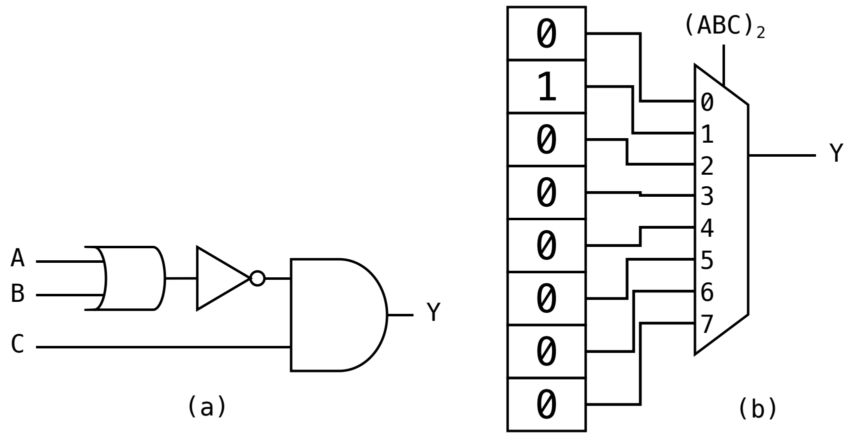
2 (l) Identify the logic function of the following circuit. A and B are the inputs, and C is the output. `(    )`

A. **`AND`**.
B. **`OR`**.
C. **`(not A) and (not B)`**.
D. **`(not A) or (not B)`**.
E. None of the above.


High
A
B
C
A
B
Low
PMOS
High resistance when input is 1
NMOS
High resistance when input is 0

Solution: D.

4 (m) **(Fill in the cells)** Below (b) shows a 3-input look-up-table (LUT) in an FPGA. The inputs A, B and C are used as the selecting signal of the 8-input multiplexer, and they are interpreted as an unsigned number $(\mathrm{ABC})_2$. By storing different values in the SRAM bit cell (small rectangles in (b)), arbitrary 3-input logics can be implemented. Fill "0" or "1" in the rest of the bit cells (the first one given) so that the same function as circuit (a) is realized. (Hint: LUT stores truth table to implement arbitrary logics, so you can start with the truth table of (a).)



Solution: The given cell at the top worth 0.5 point, and it is free. All the other 7 cells, 0.5 each.

2 (n) (**One or Multiple Choices**) Choose the correct statement(s). Select all that apply. ( )

A. Pipeline and multi-issue are both examples of instruction-level parallelism.

B. Both pipeline and multi-issue are guaranteed to reduce the execution time of a program.

C. Pipeline clock rate is limited by the slowest stage.

D. Multi-issue, SIMD and multi-thread are different ways of parallelism, and thus they cannot work together.

Solution: A, C. 0.5 point each for exactly having A and C and having neither B nor D, respectively.

3. **RISC-V.** Given a function **mystery**. It takes one argument **a0**, set up in the prologue following calling convention. The function **mystery** stays somewhere in the memory, but not at address **0x0**.

```
mystery:
   # Prologue (omitted)
   la t6 loop
loop: nop
   lw t5 0(t6)
   addi t5 t5 0x80
   sw t5 0(t6)
   addi a0 a0 -1
   bnez a0 loop
   # Epilogue
   ret
```

2 (a) The pseudo-instruction **nop** is the following RV32I base integer instruction. Translate it into machine code in **hexadecimal** format.

Solution: Answer: **addi x0 x0 0: 0x00000013**

2 (b) Briefly explain what will happen if we call **mystery** with **a0** set to 7?

Solution: Answer: Reset the registers (x1-x6) to 0. (It is OK to include x0, i.e., reset registers x0-x6 to 0).

5 (c) What are the values of **a0** that **bnez** sees running **mystery** at every iteration with **a0** set to 13? Fill your answer one by one. If it loops fewer than 13 times, simply leave blanks. The first three values are given to you.

12	11	10										

Solution: Answer: After resetting `x10 = a0`, `addi a0 a0 -1` sets `a0 = -1`. Therefore, `bnez` sees -1, -2, -3 ... and the code never reaches to the stop condition (`a0 = 0`).

12	11	10	9	8	7	6	5	4	3	-1	-2	-3

2 (d) Which registers need to be recovered in the epilogue of the function **mystery** before returning? Assume **mystery** is a leaf function that does not call the other functions.

A. **a0**
B. **t5**
C. **t6**
D. **ra**

E. None of the above.

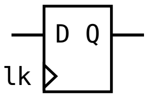
Solution: E
Code Explanation:

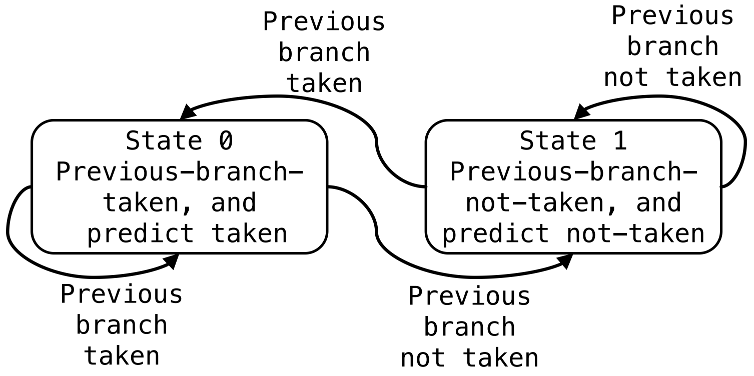
1. **`la t6 loop`**, **`t6`** stores the address of **`loop`**.
2. **`nop = addi x0 x0 0`**, the format of **`addi`** is:

[31:20]	[19:15]	[14:12]	[11:7]	[6:0]
imm[11:0]	rs1	funct3	rd	opcode

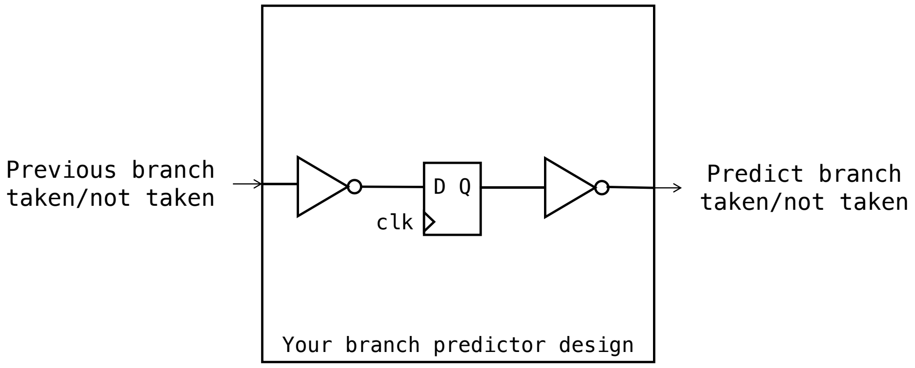
3. **`lw t5 0(t6)`**, load the instruction store at **`t6`** (i.e. **`loop`**) to **`t5`**.
4. **`addi t5 t5 0x80`**, adds 0000 0000 0000 0000 0000 0000 1000 0000 to the instruction at **`loop`**, which means adding 1 to **`rd`**.
5. **`addi a0 a0 -1`**, decrease a0 by 1.
6. **`bnez a0 loop`**, if **`a0 == 0`**, continue. Otherwise, go back to the instruction at **`loop`**.

4. **FSM and SDS.** A dynamic branch predictor can be used to reduce stalls in a pipelined CPU. We will build a dynamic branch predictor in this question. The simplest strategy is that we produce the prediction result based on the previous branch operation. For example, if the previous branch is taken, a branch-taken is predicted and vice versa (predict branch-not-taken if the previous branch is not taken).

$\boxed{6}$ (a) The Moore finite state machine (FSM) model below implements the simplest dynamic branch predictor. An input signal indicates whether the previous branch is taken or not (1 for taken and 0 for not taken), and it changes the state of the branch predictor. If it is at state 0, it predicts that the next branch will be taken (output 1); otherwise, it predicts a not-taken (output 0). Complete the digital circuit that implements this FSM model in the rectangle box below. You may only use 2-input **`AND/OR`** gates, **`not`** gates and D flip-flops (DFFs) (use  as the symbol of a DFF, "`clk`" stands for the clock signal).


Previous branch taken
Previous branch not taken
State 0 Previous-branch-taken, and predict taken
State 1 Previous-branch-not-taken, and predict not-taken
Previous branch taken
Previous branch not taken

Solution:


Previous branch taken/not taken
D Q
clk
Predict branch taken/not taken
Your branch predictor design

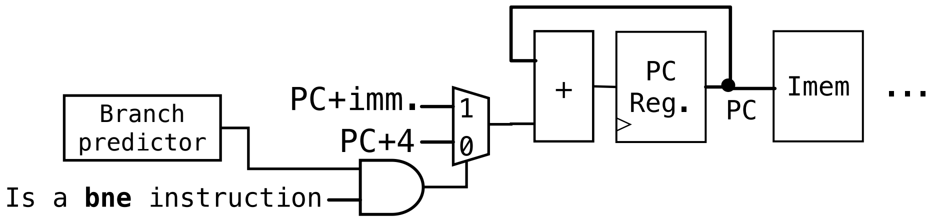
Two points for state change logic (the first inverter); two points for state register (a single DFF); two points for the output logic (the second inverter).

2 (b) Assume that the FSM model in the previous question is at state 0 initially. It is used to predict the branches in the following RISC-V code. Write down the number of wrong prediction(s) it makes.

```
      addi  t2 t1 -20
      addi  s0 x0 1      #initialize s0
Loop: lw    t3 0(t1)     #load array element
      add   t3 t3 s0     #add s to $t3
      sw    t3 0(t1)     #store result
      addi  t1 t1 -4     #t1=t1-4
      bne   t1 t2 Loop   #repeat loop if t1!=t2
```

Solution: There is only 1 wrong prediction at the last loop.

2 (c) With the branch predictor above only, it is not sufficient to decide whether **`PC+4`** (branch-not-taken) or **`PC+imm.`** (branch-taken) is used to fetch the next instruction in an RV32I datapath. We have to further make sure that the branch predictor works **only** on a branch instruction. Thus, the circuit below is designed to take this into consideration with the previously designed "Branch predictor". The problem is simplified so that only **`bne`** instruction is considered for the prediction. Please write down the **logic expression** to identify that an instruction is a **`bne`** instruction or not. Use **`and`** and **`not`** logic only to identify a **`bne`** instruction; use **`i[n]`** to represent the **`n`**-th bit of an instruction (rightmost bit as the 0th bit).

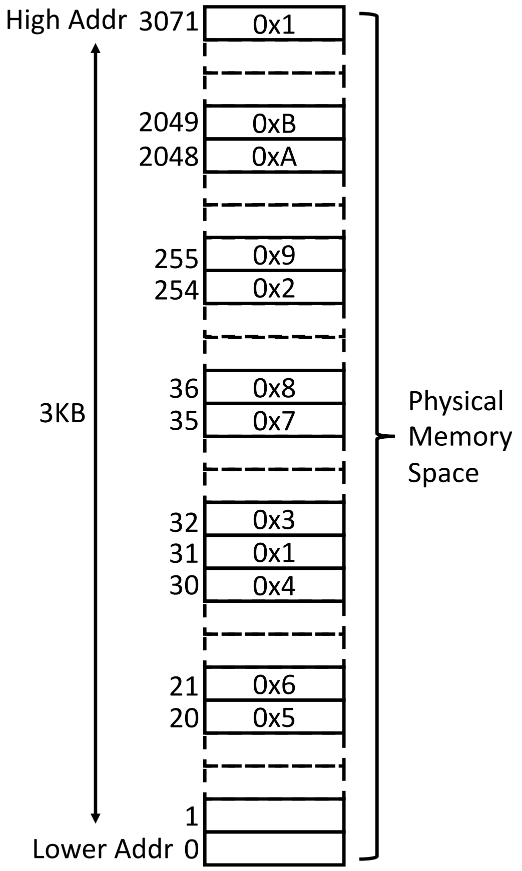


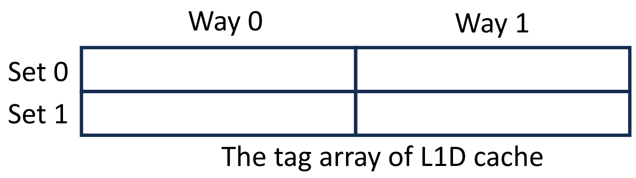
Solution: `~i[14]~i[13]i[12]i[6]i[5]~i[4]~i[3]~i[2]i[1]i[0]`
You should include all the above bits for exact identification of `bne`.

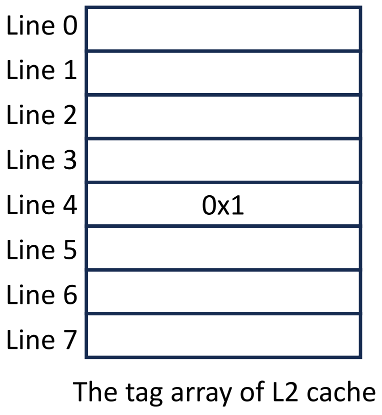
(Note that in reality, we may also have to consider the position of the branch to make the prediction. For ease of implementation, it is also not considered here.)

5. **Virtual memory and cache.**

Now you have a memory hierarchy of two-level caches (L1 and L2) on top of a physical memory space. Please ignore L1 instruction (L1I) cache and consider L1 data (L1D) cache only. A cache line has 32 bytes for both levels. Assume that a missed cache line is put in L2 cache first and then L1D cache. L1D and L2 caches follow the inclusive policy. The capacity of L1D cache is 128 bytes. The capacity of L2 is 256 bytes. L1D cache is a two-way set associative and managed with the NMRU replacement policy. L2 cache is direct mapped. Both L1 and L2 caches are accessed using the physical address. As to the memory management, a page has 256 bytes and you have a linear page table. A virtual address is in 16 bits. A physical address has 12 bits. The capacity of physical memory is 3KB as shown below. The contents at some memory locations are shown too. The base address of page table is kept at 0 of the physical memory. Each PTE takes one byte. There is no TLB and you need to do page table walk for address translation.



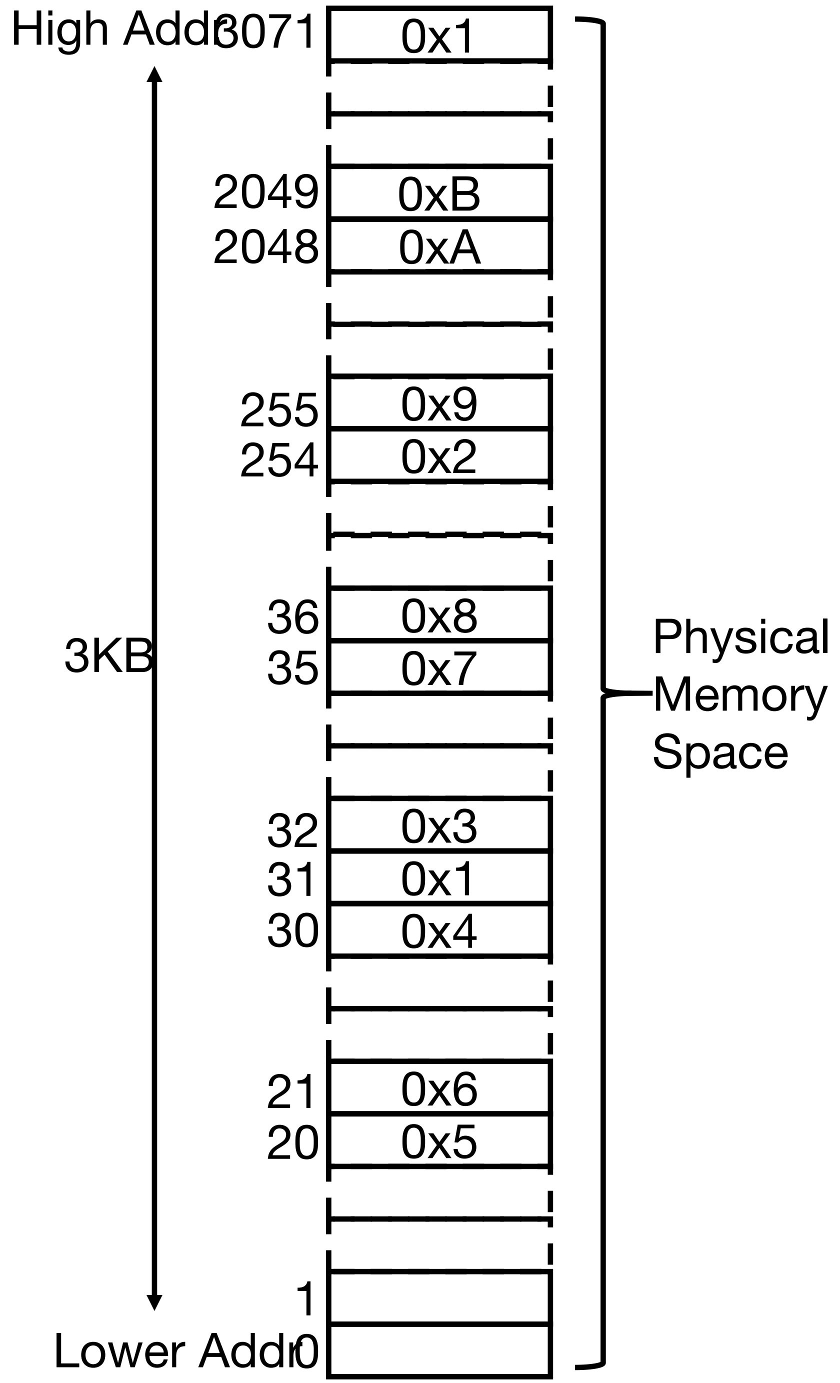


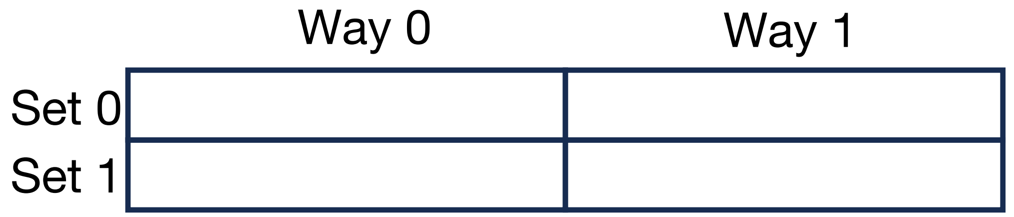


$\boxed{14}$ (a) Assume that you have the following memory access trace with seven virtual addresses: 0x2024, 0x153F, 0x20A0, 0xFEBF, 0x2313, 0x1F82, 0x1520.

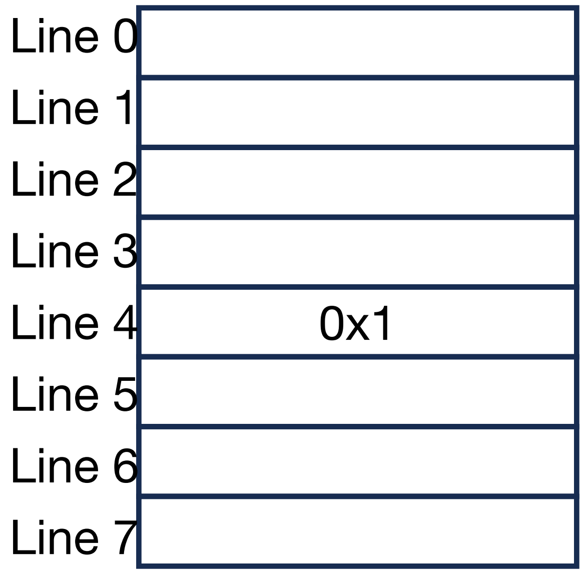
At start, both L1 and L2 caches are empty without any data. Please fill the tag arrays for L1D and L2 caches with the last tags only, after you access these addresses one by one in the above-mentioned order. One such tag has been filled for you in the L2 cache. You can leave blank(s) if you find one or multiple cache lines are not used. Please write your tag(s) in the hexadecimal format, i.e., with 0x used as a prefix.

Solution: See below.



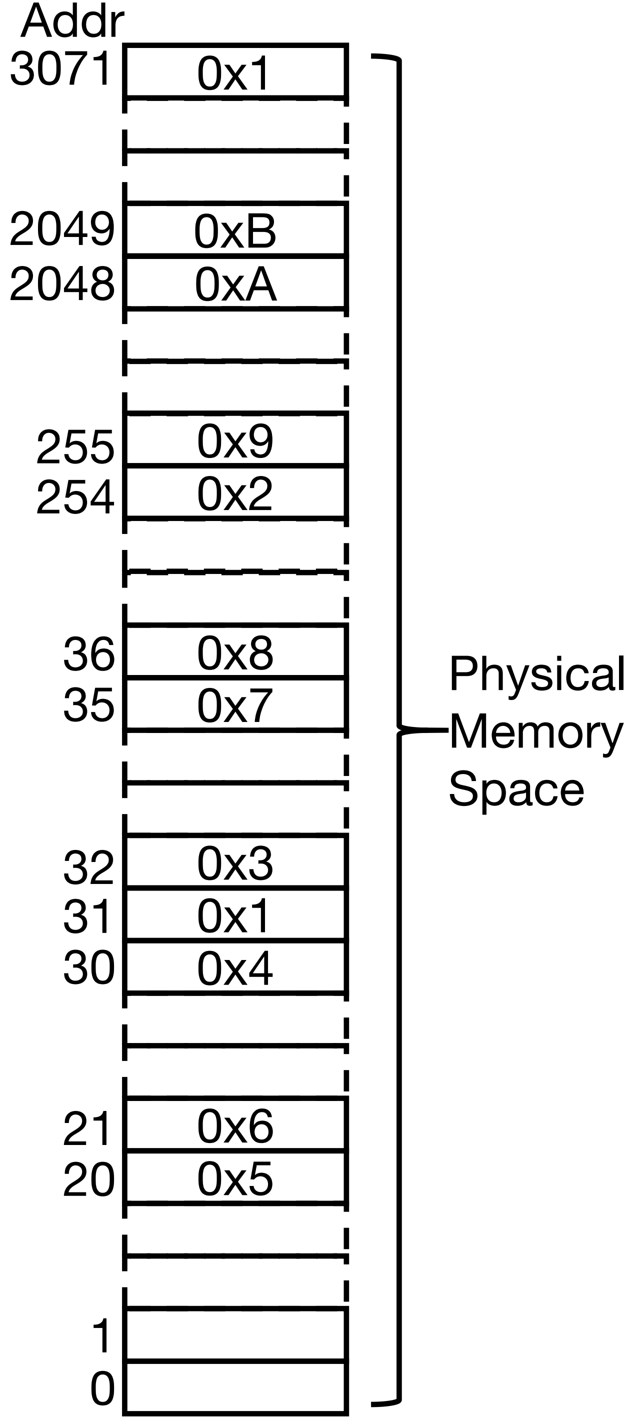


The tag array of L1D cache



The tag array of L2 cache

1. Read 0x2024



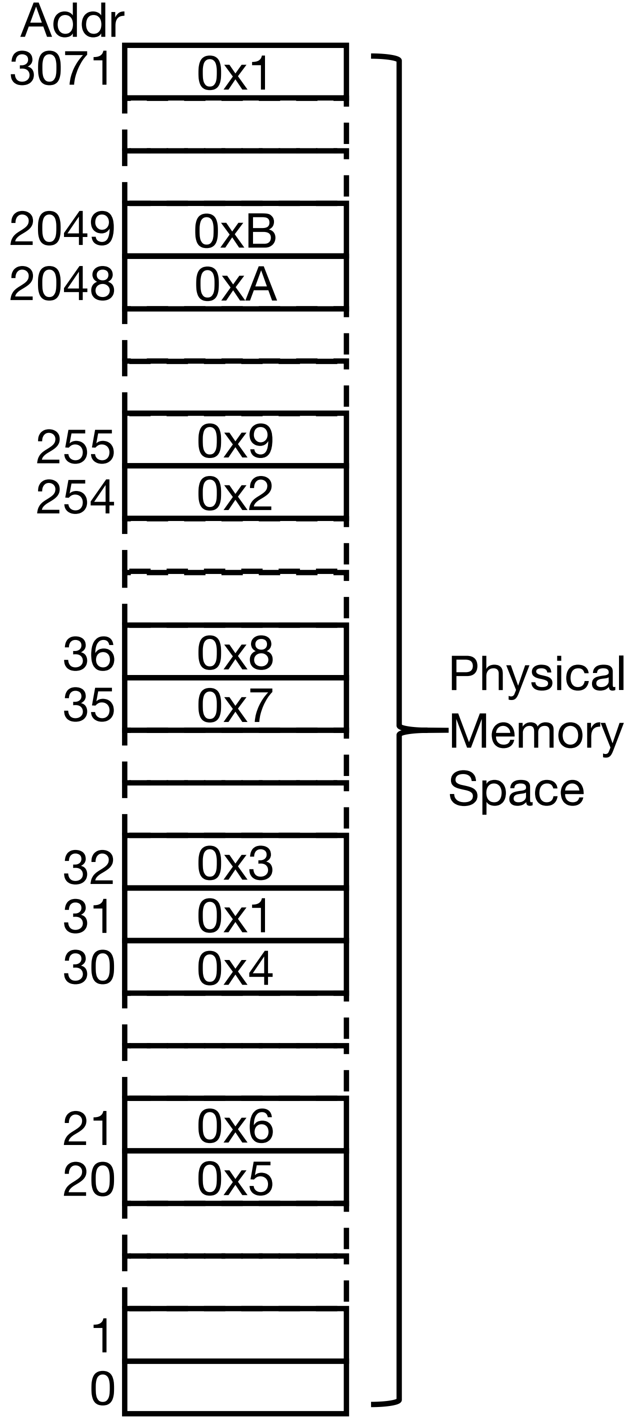
	Way 0	Way 1
Set 0		
Set 1	0x0C	

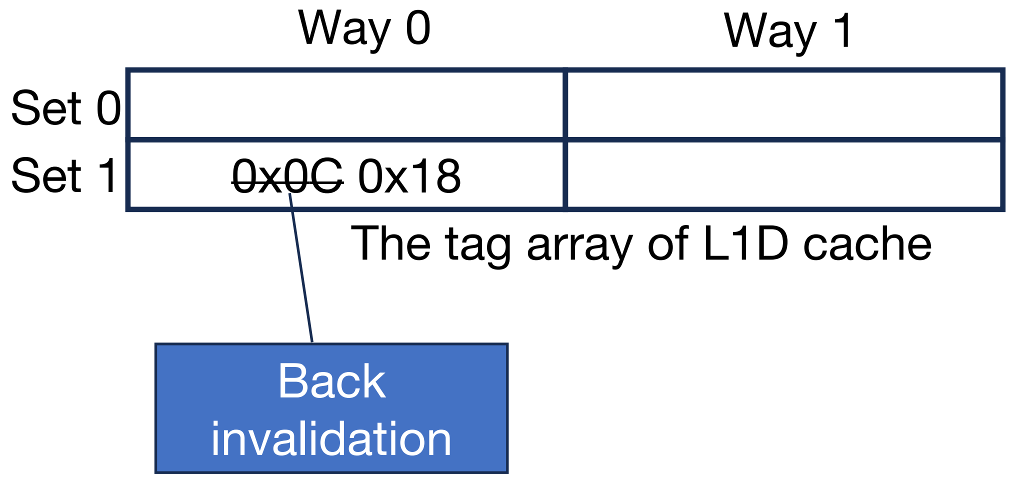
The tag array of L1D cache

Line 0	
Line 1	0x3
Line 2	
Line 3	
Line 4	
Line 5	
Line 6	
Line 7	

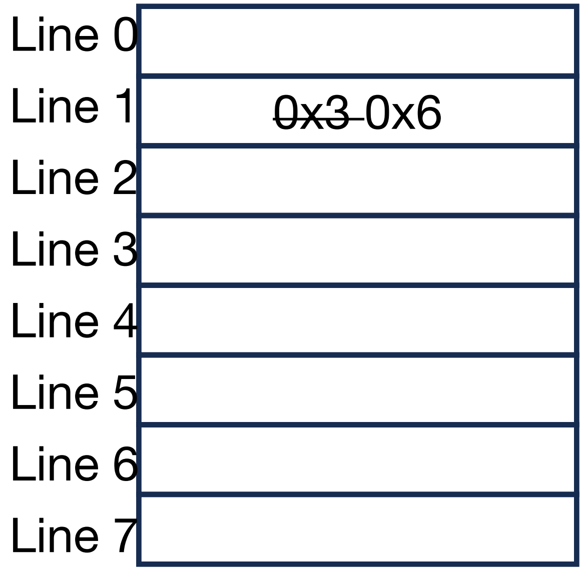
The tag array of L2 cache

2. Read 0x153F



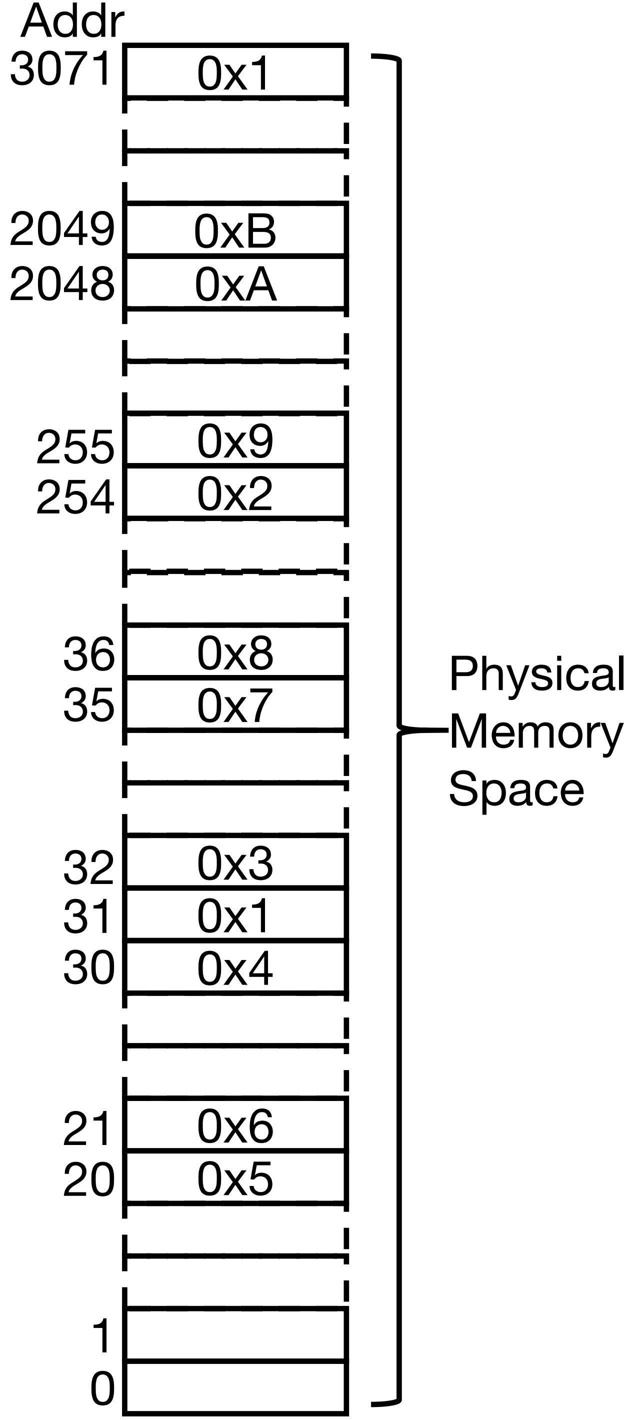


The tag array of L1D cache



The tag array of L2 cache

3. Read 0x20A0



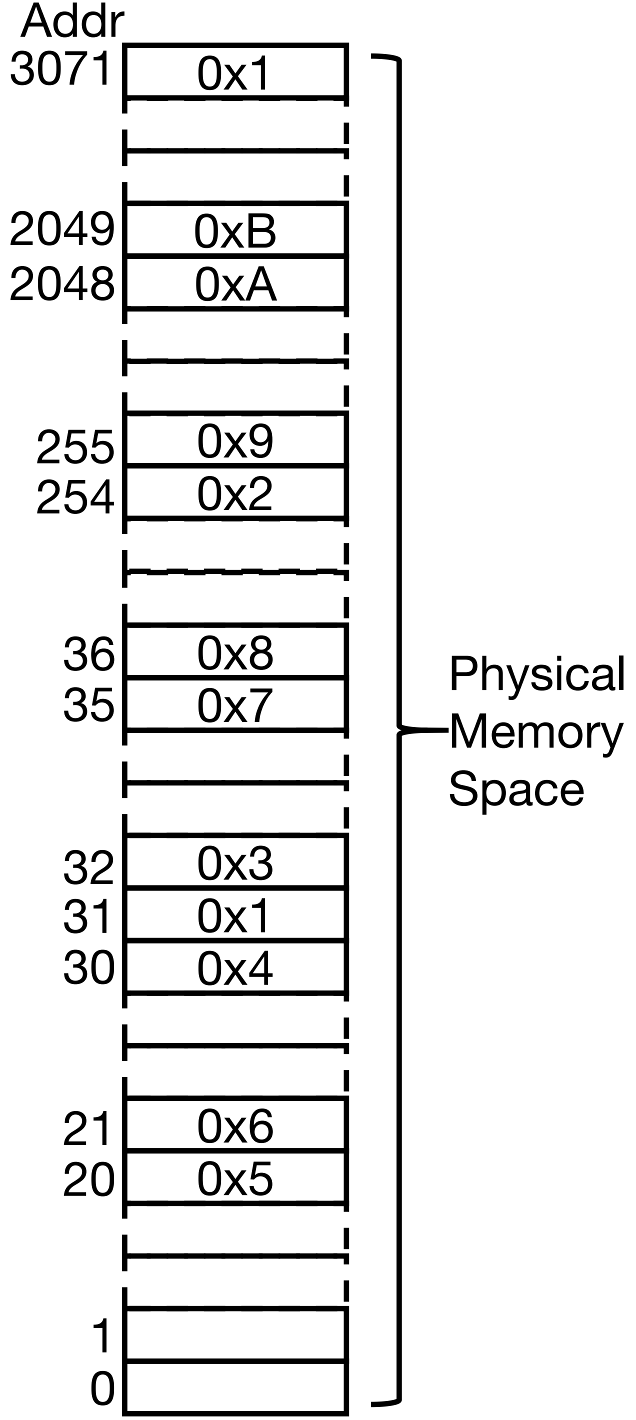
	Way 0	Way 1
Set 0		
Set 1	~~0x0C~~ 0x18	0x0E

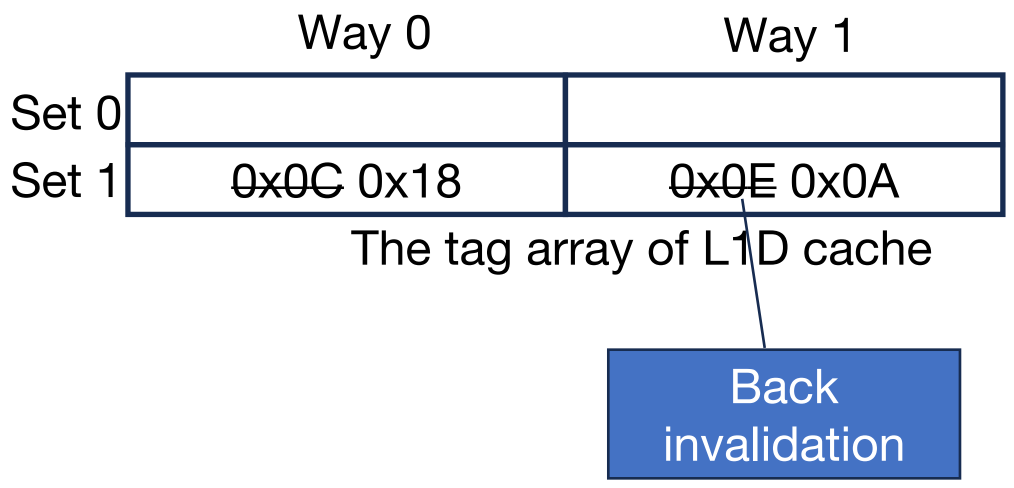
The tag array of L1D cache

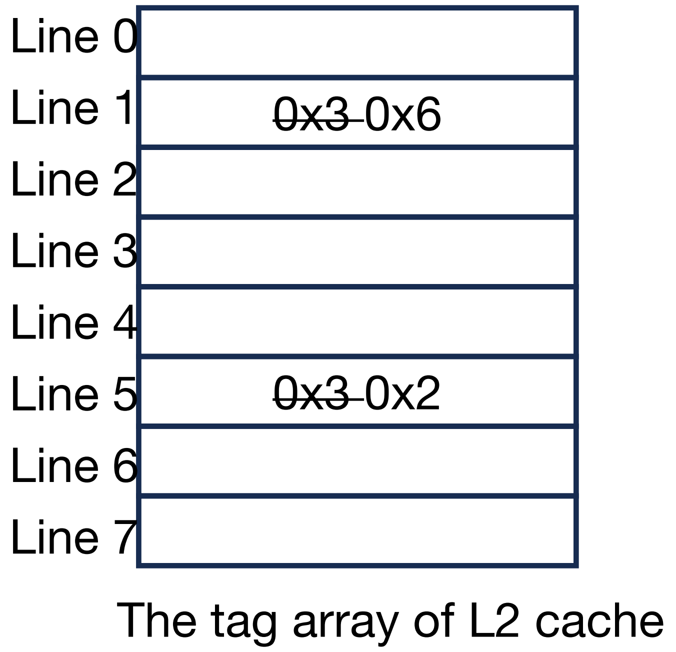
Line 0	
Line 1	~~0x3~~ 0x6
Line 2	
Line 3	
Line 4	
Line 5	0x3
Line 6	
Line 7	

The tag array of L2 cache

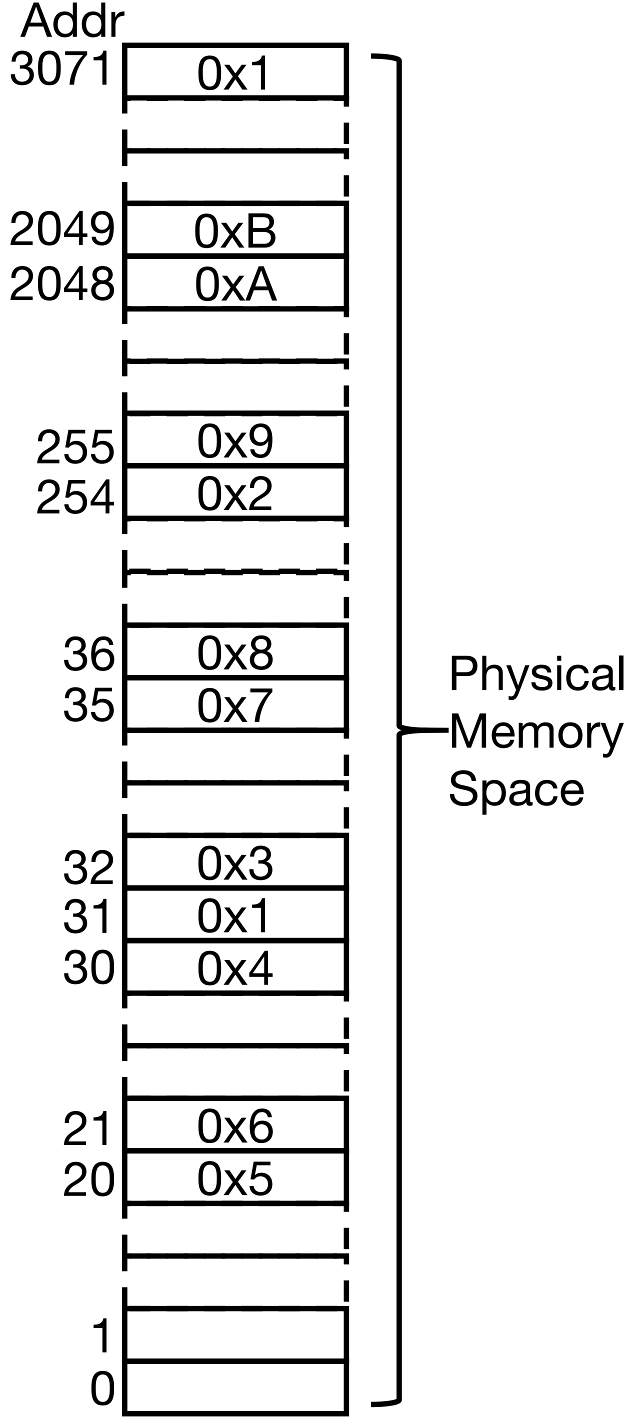
4. Read 0xFEBF







5. Read 0x2313



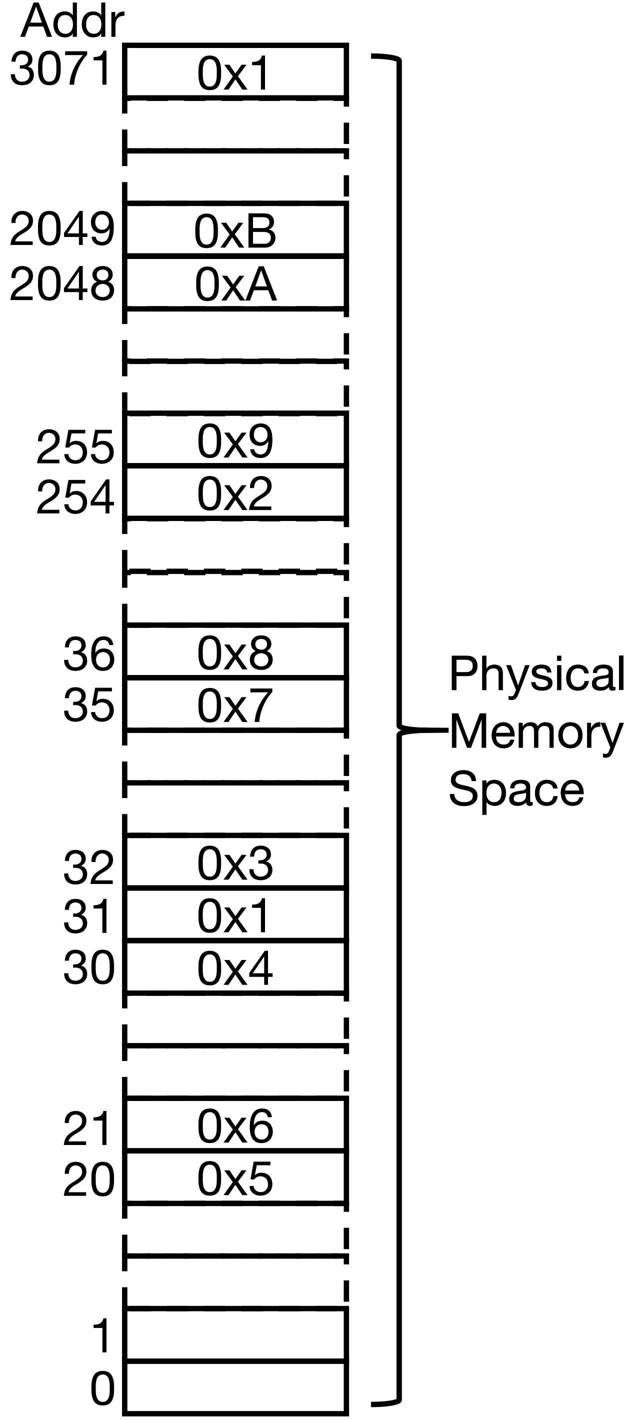
	Way 0	Way 1
Set 0	0x1C	
Set 1	~~0x0C~~ 0x18	~~0x0E~~ 0x0A

The tag array of L1D cache

Line 0	0x7
Line 1	~~0x3~~ 0x6
Line 2	
Line 3	
Line 4	
Line 5	~~0x3~~ 0x2
Line 6	
Line 7	

The tag array of L2 cache

6. Read 0x1F82



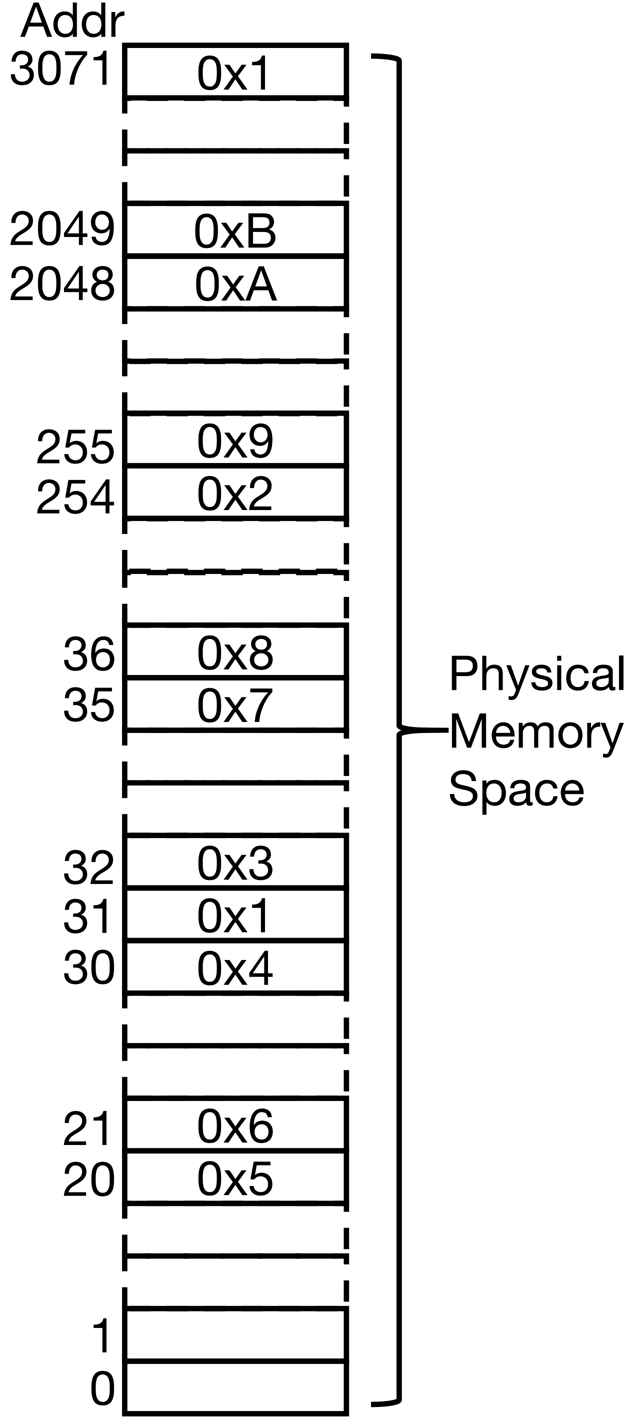
	Way 0	Way 1
Set 0	0x1C	0x06
Set 1	~~0x0C~~ 0x18	~~0x0E~~ 0x0A

The tag array of L1D cache

Line 0	0x7
Line 1	~~0x3~~ 0x6
Line 2	
Line 3	
Line 4	0x1
Line 5	~~0x3~~ 0x2
Line 6	
Line 7	

The tag array of L2 cache

7. Read 0x1520



	Way 0	Way 1
Set 0	0x1C	0x06
Set 1	~~0x0C~~ 0x18	~~0x0E~~ 0x0A

The tag array of L1D cache

Cache hit

Line 0	0x7
Line 1	~~0x3~~ 0x6
Line 2	
Line 3	
Line 4	0x1
Line 5	~~0x3~~ 0x2
Line 6	
Line 7	

Cache hit

The tag array of L2 cache

6 (b) After accessing seven virtual addresses,

i). How many cache hit(s) happen at L2 cache? If there are (is), please explain at which cache line(s) for accessing what address(es) they (it) happen(s).

ii). How many cache replacement(s) happen at L1D? If there are (is), please explain they (it) happen(s) for accessing what address(es).

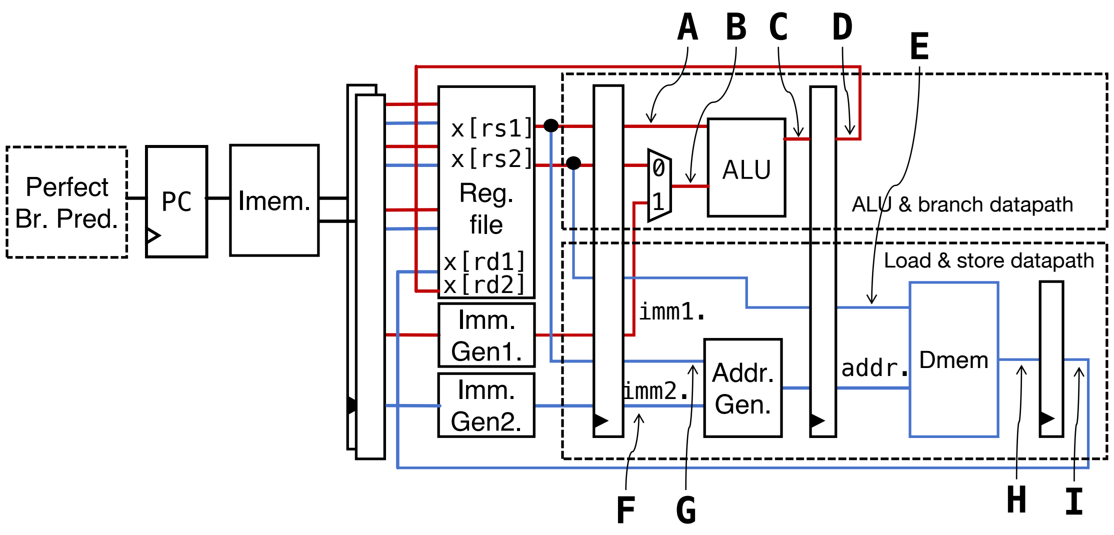
Solution: i). 1 hit. (One Point)

Cache hit only happens when accessing 0x1520, at Line 1 of L2 cache. (2 Points)

ii). 0 replacements. (3 Points)

6. **Pipeline and multi-issue.** Below shows a simplified multi-issue pipelined RISC-V CPU datapath design covered in the lecture. The ALU & branch datapath supports all the RV32I ALU & branch instructions. The load & store datapath supports all the RV32I load and store instructions. An ALU or branch instruction and a load or store instruction can be issued simultaneously to these two datapaths to achieve parallel execution.

   Further assume that we can arbitrarily read and write all the memory elements, i.e., register file (Reg. file), data memory (Dmem) and instruction memory (Imem), so that no structural hazard occurs on these elements. Suppose register file writes can happen before register reads within the same clock cycle. The perfect branch predictor (Perfect Br. Pred.) always fetches the desired instruction from the Imem on branches. Neither forwarding mechanism nor the other hazard detection mechanism is implemented initially.



6 (a) Consider the execution of the following instructions on the multi-issue pipelined CPU. Please arrange the instructions into the time slots below in the table so that they can be executed within the least number of clock cycles. Each column represents a clock cycle (CC). The time slots for different stages of **mv** instruction is already given. We ignore the bubble/**nop** insertions. Leave it blank if you do not use all the time slots.

```
mv x11 x8
addi x12 x11 80
lw x13 0(x17)
sw x11 0(x13)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
mv	IF	ID	EX	WB			
addi							
lw							
sw							
	CC8	CC9	CC10	CC11	CC12	CC13	CC14
mv							
addi							
lw							
sw							

Solution:

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
mv	IF	ID	EX	WB				
addi			IF	ID	EX	WB		
lw	IF	ID	EX	MEM	WB			
sw				IF	ID	EX	MEM	(WB)

Since read/write the register can occur simultaneously in one clock cycle for a certain register, it is safe to arrange WB of `mv` and ID of `addi` in CC4. The same for the following `lw` and `sw` instructions.

Also, since it is a multi-issue pipelined CPU, we can issue two instructions with different types (alu and mem. access) to the pipeline in 1 clock cycle. However, due to data hazards, it cannot be achieved for each clock cycle. Please see the reference answer for details.

6 (b) Then, suppose forwarding is implemented to avoid data hazards in the above assembly code as much as possible. Complete the following time slot arrangement with the multi-issue pipelined CPU using the least number of clock cycles. You can disregard the "WB" stage of the **sw** instruction.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
mv	IF	ID	EX	WB			
addi							
lw							
sw							

Solution:

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
mv	IF	ID	EX	WB			
addi		IF	ID	EX	WB		
lw	IF	ID	EX	MEM	WB		
sw			IF	ID	EX	MEM	(WB)

Since we have the forwarding mechanism, the data hazard between `mv` and `addi` can be resolved. The result is forwarded from EX/WB stage register (`D` point) to the upper input of the upper ALU (`A` point), thus the answer "D" and "A" in the following question. We can save one bubble from this forwarding and `addi` can be loaded in the pipeline right next to `mv` instruction.

Similarly, the value of `x13` can be forwarded from the MEM/WB stage register (`I` point in the diagram) directly to the lower ALU (`G` in the diagram), so that we can arrange EX stage of `sw` after the MEM stage of `lw`. The forwarding path `I` to `G` has to be activated correspondingly.

$\boxed{4}$ (c) Following the previous question, a forwarding datapath between _**`D`**_ and _**`A`**_ (see the points in the circuit diagram) should be activated for the **`addi`** instruction.

Solution: Similarly, a forwarding datapath between __`N/A`___ and __`N/A`___ should be activated for the **`lw`** instruction, since there is NO HAZARD AT ALL.

Similarly, a forwarding datapath between __`I`__ and __`G`__ should be activated for the **`sw`** instruction. See the reference answer in the above question for detailed explanation.

Fill in *N/A* if you think no forwarding datapath is required to be activated to avoid data hazard for that instruction, and fill in multiple letters if you think more than 1 datapaths should be activated.

7. **Number representation.** Let's consider the hexadecimal number 0xFCC48493. How is the number interpreted, if we treat it as. . .

$\boxed{4}$ (a) an array A of 4 signed, 8-bit integers? Please write each number in decimal, assume the machine is little-endian. If the value is unknown, fill in *GARBAGE* (in all caps).

i. A[0]: ______________________________

ii. A[1]: ______________________________

iii. A[2]: ______________________________

iv. A[3]: ______________________________

$\boxed{2}$ (b) an IEEE-754 single precision floating point number? Write down as binary scientific notation, so e.g, an answer that looks like this: $-(1.0100100)_2 \times 2^{15}$

____

$\boxed{2}$ (c) a RV32I instruction? If there's an immediate, write it in decimal. If it is an invalid instruction, fill in *INVALID INSTRUCTION*.

____

Solution:

(a) i. (-109)

ii. (-124)

iii. (-60)

iv. (-4)

in 2's complement; or

i. (-124)

ii. (-68)

iii. (-4)

iv. (-19)

in sign-magnitude; or

i. (-3)
ii. (-59)
iii. (-123)
iv. (-108)

in 1's complement; If got the order wrong, but the numbers are correct, e.g., -4, -60, -124, -109, minus 1 point. Note that in modern computer, we almost always use 2's complement representation.

(b) $-1.10001001000010010010011 \times 2^{122}$

(c) addi x9, x9, -52

8. **Synchronization**

Assume that two threads are going to execute the following RISC-V-like code. They intend to increase the variable `val` to be 2 in the end. The `lock` is the variable they employ for synchronization.

```
// Initially both val and lock hold zeros.
// val and lock are placed in one cache line.
// The register x0 is always holding zero.
// one is a variable that is always holding one (1).
      lw x1, 0(lock) // put the value of lock in register x1
      lw x2, 0(val) // put the value of val in register x2
      li x3, one // put 1 into register x3
wait:
      bne x1, x0, wait // if x1 != x0, goto wait
      sw x3, 0(lock) // store x3 into lock
      addi x2, x2, 1 // x2 + 1 --> x2
      sw x2, 0(val) // store x2 into val
```

$\boxed{5}$ (a) Do you think this code has any problem(s) with regard to correct synchronization and desired result? Please explain.

Solution: Yes. (<u>One Point</u>)

There are two problems. (<u>Each with two points</u>)

1. The 'lock' is not unlocked after use. (One Point)
2. The value of 'lock' is not reloaded after wait. / Loading value is not in critical section. (One Point)
3. Locking is not done atomically. (Two Points)

$\boxed{2}$ (b) Assume that two threads would be running on two physical CPU cores, respectively, with a correct synchronization used in between. With regard to cache hierarchy, will they experience any 'true sharing'? How about 'false sharing'? Please give your reasons.

Solution: Yes, they suffer from true sharing due to using `lock` or `val`. (<u>One Point</u>)

Yes, they suffer from false sharing when one is writing `lock` (resp. `val`) while the other one is reading `val` (resp. `lock`). (<u>One Point.</u>)

9. **Cluster.** Assume that you are given a program, of which a fraction f (measured in percentage) can be optimized for parallel computing by employing N machines built as a cluster.

$\boxed{3}$ (a) Consider that $f = 80\%$ and you have a sufficient budget for purchasing 20 machines made with the same configurations, i.e., $N = 20$. If you plan to accelerate the program's performance by about four times, do you have to use up all the budget for 20 machines? Please explain your answer.

Solution: No. (One Point).
According to Amdahl's Law, $\frac{1}{(1-80\%)+80\%/N} = 4$, $N = 16$. 16 machines suffice for four times improvements. (Two Points)

$\boxed{2}$ (b) Assume that you have eventually received 20 machines. In each machine there are five solid-state drives (SSDs). If you intend to achieve an annualized failure rate of no greater than 5.0%, what is the minimum mean time to failure (MTTF, measured in the unit of hour) for these SSDs? Use the year of 2024 for calculation and explanation.

Solution: 2024 has 366 days. Assume MTTF for these 100 SSDs is x.
$\frac{\frac{24*366*100}{x}}{100} \leq 5\%$, so we have $x \geq 175680$. The minimum MTTF is 175,680 hours.