Computer Architecture I Mid-term Exam 2

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

E-Mail prefix: _____

Question	Points	Score
1	1	
2	10	
3	12	
4	22	
5	17	
6	20	
7	18	
Total:	100	

- This test contains 12 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 12 times).

2. **MISC.**

[4] (a) Write down four of the great ideas in computer architecture.

1. _____

2. _____

3. _____

4. _____

Solution: Choose any four of the following (1 point for each, up to 4 points):

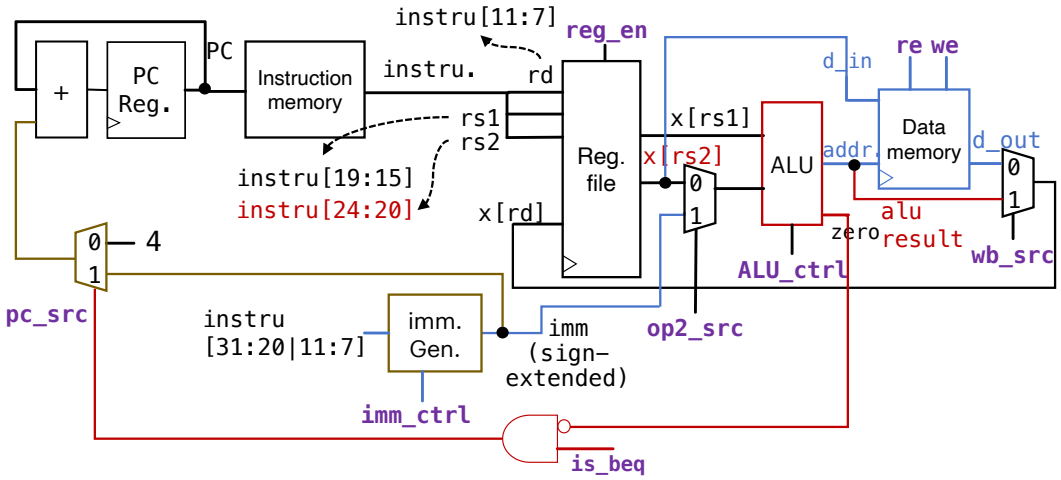
1. Abstraction.
2. Moore's law.
3. Make common case fast.
4. Memory hierarchy.
5. Parallelism.
6. Pipeline.
7. Fault tolerance/dependability with redundancy.
8. Performance evaluation.

[2] (b) **(True or False)** The assembler takes two passes over the code to resolve PC-relative target addresses. ( )

Solution: True

[4] (c) The circuit diagram below shows a single-cycle datapath that supports RV32I R- and I-type arithmetic and logic operations, **`lw`**, **`sw`** and **`beq`** instructions. For some reason, the **`reg_en`** signal stucks at logic '1'. In this case, which of the following instruction(s) can not function properly? ( )

A. **`beq`**.
B. **`sub`**.
C. **`sw`**.
D. **`lw`**.



Solution: Answer: A & C. 1 point for not choosing B/D; 1 point for choosing A/C. **`beq`** and **`sw`** instructions should not write or modify the register file or change the values in the register file. However, with **`reg_en`** signal stuck at '1', the register file takes whatever the input of **`x[rd]`** and would change the value of the corresponding register at a rising clock edge.

3. **C, Basic Cache, and Performance.**

5 (a) Matrix multiplication is a broadly used operation in many applications. In this part, you will optimize matrix multiplication and analyze its performance.

If we perform matrix multiplication of matrices X_{m*n} and Y_{n*k}, the element in the resultant matrix W_{m*k} is obtained as

$$W[i][j] = \sum_{t=1}^{n} X[i][t] * Y[t][j] \ \ (1 \leq i \leq m, 1 \leq j \leq n).$$

Directly computing it in its original form performs bad since the way CPU traverses matrix Y may cause lots of cache misses. One possible method to avoid this is to transpose the matrix Y first and use the transposed matrix for multiplication.

In C, by executing $Z[i][j] = Y[j][i]$, we can transpose a matrix. Please implement the following C program for transposing a matrix. You may use all or part of the blanks below.

```
/* Y : matrix to be transposed
m : number of rows
n : number of columns

You need to create a new matrix to store the transposed Y and
   return the pointer.
*/
```

Solution: The input is 1-D array, so we need to calculate the index.

```
1  int *transpose(int *Y, int m, int n) {
2    int *Z = malloc(m * n * sizeof(int));
3    for (int i = 0; i < m; i++) {
4      for (int j = 0; j < n; j++) {
5        Z[j * m + i] = Y[i * n + j];
6      }
7    }
8    free(Y);
9    return Z;
10 }
```

(b) The following code is a naive implementation of matrix multiplication, using 2-D array for simplicity.

```
1 void multiplication(int **X, int m, int n, int **Z, int k, int
    **W) {
2   //Initially all elements of W are holding zeros.
3   for (int i = 0; i < m; i++) {
4     for (int j = 0; j < k; j++) {
5       for (int t = 0; t < n; t++)
6         W[i][j] += X[i][t] * Z[j][t];
7     }
8   }
9 }
```

To simplify analysis, we assume that

1. Data load/store, multiplication and sum in C will be translated to one instruction each, e.g. X[i][t] is a data load, and it will be translated into only one instruction.
2. Assume multiplication, addition, load and store instructions take 4, 1, 2, 2 cycles, respectively.
3. Ignore the overhead of loops and function call, only consider the execution of **Line 6**.

4 i. Please calculate the average CPI.

Solution: To execute `Line 6`, it requires 1 multiplication, 1 addition, 3 loads (`W[i][j]`, `X[i][t]`, `Z[j][t]`), 1 store (`W[i][j]`). Thus, CPI = $(1 \times 4 + 1 \times 1 + 3 \times 2 + 1 \times 2)/(1 + 3 + 1 + 1) = \frac{13}{6}$

3 ii. For m=32, n=16, k=8, clock rate=2.5 GHz, please estimate the runtime of the program.

Solution: Runtime = $m * n * k * 13$ cycles $* \frac{1}{2.5*10^9} \approx 21.3\mu s$

4. **More Cache.** Suppose we have a function `transpose()`.

```
1  typedef int array[4][4];
2
3  void transpose(array dst, array src) {
4    int i, j;
5    for (i = 0; i < 4; i++) {
6      for (j = 0; j < 4; j++) {
7        dst[j][i] = src[i][j];
8      }
9    }
10 }
```

We run this function in a machine with an 8-bit address space. Let us ignore all memory accesses related to fetching instructions as well as dealing with i and j. In other words, please consider memory accesses only for dst's and src's data elements at **Line 7**. Note that

- The src array starts at 0x40.
- The dst array starts at 0x80.
- The cache is empty at start.
- In this machine `sizeof(int)` is `4`.

(a) Suppose this machine has a 2-way set-associative cache with the NRU replacement policy. The cache size is 32 Bytes and block size is 8 Bytes.

[1] i. (**True** or **False**?) For this cache, the effect of NRU replacement policy is equivalent to that of LRU replacement policy. ( )

Solution: T.

[3] ii. What is the lengths of tag, set index and block offset fields?
The length of Tag is _____.
The length of set index is _____.
The length of block offset is _____.

Solution: tag 4, index 1, offset 3.

[2] iii. What is the hit rate after execution?

Solution: 25%.

[8] (b) Suppose this set-associative cache uses the FIFO replacement policy, instead of the NRU policy. The cache size is still 32 Bytes and block size is 8 Bytes. The cache is empty at start. Fill the following table to indicate the access results of some elements of dst and src arrays with *h* (for 'cache hit') or *m* (for 'cache miss'). The first one is filled for you.

element	access	element	access	element	access	element	access
src[0][0]	*m*	src[1][0]	*m*	dst[0][0]	*m*	dst[1][0]	*m*
src[0][1]	*h*	src[1][1]	*h*	dst[0][1]	*m*	dst[1][1]	*m*
src[0][2]	*m*	src[1][2]	*m*	dst[0][2]	*m*	dst[1][2]	*m*
src[0][3]	*h*	src[1][3]	*h*	dst[0][3]	*m*	dst[1][3]	*m*

8 (c) Now suppose the cache is direct-mapped. The cache size is still 32 Bytes and block size is 8 Bytes. Please fill the following table to indicate the access results of some elements of dst array and src array with *h* (for 'cache hit') or *m* (for 'cache miss'). The cache is empty at start.

element	access	element	access	element	access	element	access
src[2][0]	*m*	src[3][0]	*m*	dst[2][0]	*m*	dst[3][0]	*m*
src[2][1]	*h*	src[3][1]	*h*	dst[2][1]	*m*	dst[3][1]	*m*
src[2][2]	*m*	src[3][2]	*m*	dst[2][2]	*m*	dst[3][2]	*m*
src[2][3]	*h*	src[3][3]	*h*	dst[2][3]	*m*	dst[3][3]	*m*

5. **Multilevel Cache.** Multi-level cache hierarchy is commonly used in today's processors. Now, let us consider its performance and other aspects.

(a) Put **True** or **False** for following statements.

2 i. Assume that a processor has a two-level cache hierarchy. In the classic five-stage for a RISC-V processor, if at the instruction fetch (IF) stage the L2 cache is accessed for an arithmetic instruction, there would be no cache access to either L1 cache or L2 cache for this instruction until it is completed. ( )

2 ii. Assume that another processor also has a two-level cache hierarchy, in which the L2 cache is four-way set-associative. Given a cache line that is not found in L1 cache, it exists in one of four sets in the L2 cache. ( )

2 iii. Assume that the third processor still has a two-level cache hierarchy. If L1 cache follows the write-through policy while the L2 cache follows the write-back policy, a write hit at L1 cache would always cause a write at L2 cache. ( )

1 iv. Given two programs, if one of them has a larger CPI, it shall have a larger cache miss rate than the other one. ( )

Solution: T, F, T, F

(b) Assume that we have the fourth processor with a two-level cache hierarchy, and

1. L1I and L1D caches both are with a hit time of 2 CPU clock cycles;
2. L2 cache is with a hit time of 12 CPU clock cycles;
3. Main memory is with an access time of 80 CPU clock cycles.

Now we have a program of five load/store instructions. Assume that all instructions are hit in the L1I cache.

4 i. If all data load or store operations are missed in L1D cache but hit in L2 cache, how many CPU clock cycles would be spent on all memory accesses of this program?

6 ii. For this particular program, assuming that we need to limit that all memory accesses are no greater than 200 CPU clock cycles, what would be the maximum local miss rate of L2 cache? Give your explanation.

Solution:

1. $5 * 2 + 5 * 12 = 70$ cycles.
2. Let us denote x and y to be the numbers of instructions that hit at L1D and L2 caches, respectively. We use z to stand for the number of instructions that miss at L2 cache and have to access memory. Therefore, $x + y + z = 5$.

   We need to have $5 \times 2 + (2 \times x + 12 \times y + 80 \times z) < 200$. We can have $(x + 6y + 40z) < 95$.

   Check if $z = 1$, so $x+y = 4$, $(x+6y+40z) = (4-y+6y+40) = 5y+44 < 95$, y could be 0, 1, 2, 3, or 4.

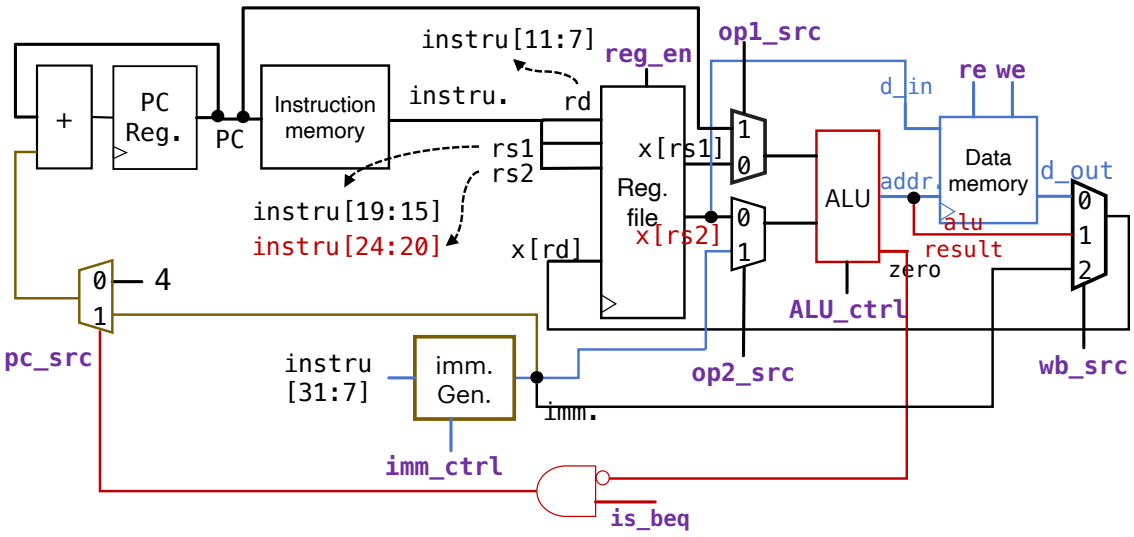
   Check if $z = 2$, so $x+y = 3$, $(x+6y+40z) = (3–y+6y+80) = 5y+83 < 95$, y could be 0, 1, or 2.

   Check if $z = 3$, so $x + y = 2$, $(x + 6y + 40z) = (2–y + 6y + 120) > 95$.

   So the maximum value of z is 2. Given five instructions for loading or storing data, if only one or two of them are not hit in both L1D and L2, the overall access time should not be greater than 200. 1/1 = 100%, and 2/2 = 100%, so the maxium local miss rate of L2 cache can be 100%.

6. **Datapath.** We would like to add supports for more RV32I instructions by modifying the single-cycle CPU datapath we covered in the lectures. It originally supports R- and I-type arithmetic and logic operations, **`lw`**, **`sw`** and **`beq`** instructions. The circuit diagram after modification is shown below. The modified datapath also supports U-type instructions. We first add a multiplexer to select the input of the ALU between **`PC`** and **`x[rs1]`** using **`op1_src`** signal. Besides, an extra option is added to the rightmost write-back multiplexer using a select signal of "2". This option selects the shifted immediate (upper 20 bits indicated by the **`lui`** or **`auipc`** instruction, lower 12 bits are 0) as the value written to the register file. Assume that the shifted immediate is provided by "imm. Gen." block in the circuit diagram.

   "**`reg_en`**" denotes register file write-enable signal; "**`re`**" stands for data memory read-enable signal; "**`we`**" represents data memory write-enable signal. We use logic '1' to enable and logic '0' to disable.



9 (a) Please indicate the values of the control signals in the table below when executing **lui**, **auipc** and **add** instruction. Use 'x' to indicate either '0' or '1' works.

Solution:

	reg_en	op1_src	op2_src	re	we	wb_src
lui	1	x	x	0	0	2
auipc	1	1	1	0	0	1
add	1	0	0	0	0	1

For **re**, it is also acceptable to fill in 'x', but '1' receives half the mark, since we have no reason to read explicitly; for **lui** instruction, either '0' or '1' receives half the mark for control signals **op1_src** and **op2_src**.

6 (b) Use **not**, **and** and **or** logics **only** to design logics that produce **op1_src**, **wb_src[1]** (MSB of **wb_src**) signals. We use i_k to represent the kth bit in the instruction, i.e., i_{31} =**instruction[31]**, i_{30} =**instruction[30]**, ..., i_0 =**instruction[0]**. Note that we still ensure that R- and I-type arithmetic and logic operations, **lw**, **sw** and **beq** work properly. **Write down the logic expressions to complete your logic design.**

Solution:

op1_src = $\bar{i_6}\bar{i_5}i_4\bar{i_3}i_2i_1i_0$.

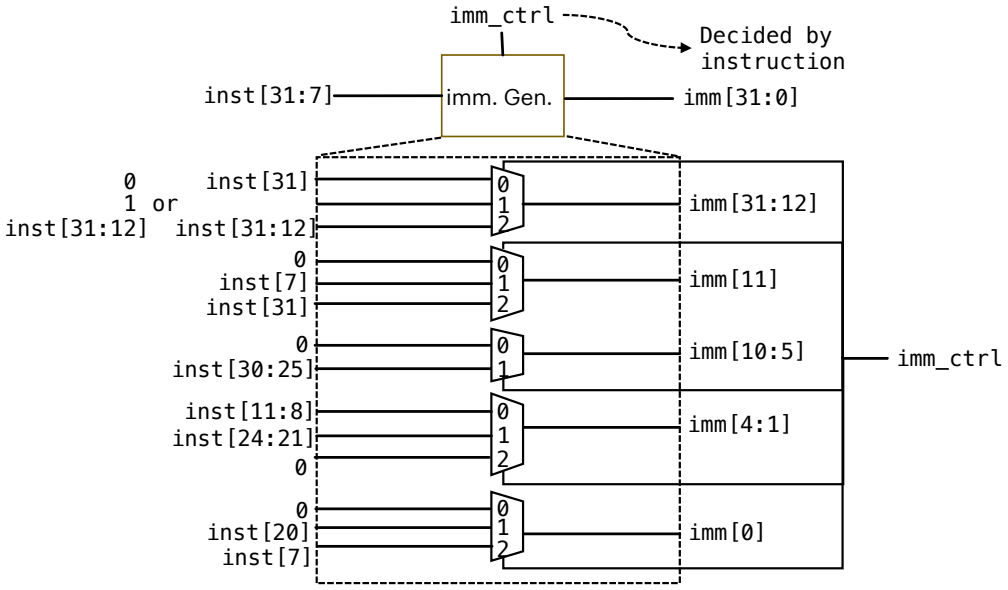
Only for **auipc**, **op1_src** is 1. Only **opcode** (0010111) is used to identify **auipc**.

wb_src[1] = $\bar{i_6}i_5i_4\bar{i_3}i_2i_1i_0$.

Only for **lui**, **wb_src[1]** is 1. Only **opcode** (0110111) is used to identify **lui**. Many of you include the logic to identify `beq`. However, as per the datapath, `PC+imm` is calculated by the leftmost adder instead of the ALU, while the ALU is used to compare `x[rs1]` and `x[rs2]`. So it is not appropriate to include the logic to identify `beq`.

5 (c) Next, we modify the "imm. Gen." block to support the production of the shifted immediate for **`lui`** and **`auipc`** instructions. Please connect the input signals properly. Note that it should still support the production of the immediate for I-type arithmetic and logic operations, **`lw`**, **`sw`** and **`beq`** instructions. Disregard the selection signals for the multiplexers. Hint: You can use **`inst[m:n]`** to indicate the **`m`**th to **`n`**th bits of the instruction. The signal connections for generating **`imm[4:1]`** is given as a reference. Modify it if you found errors in the connections. Add inputs if the input ports for the multiplexer is insufficient; or leave it blank if the input ports are more than enough.

Solution:

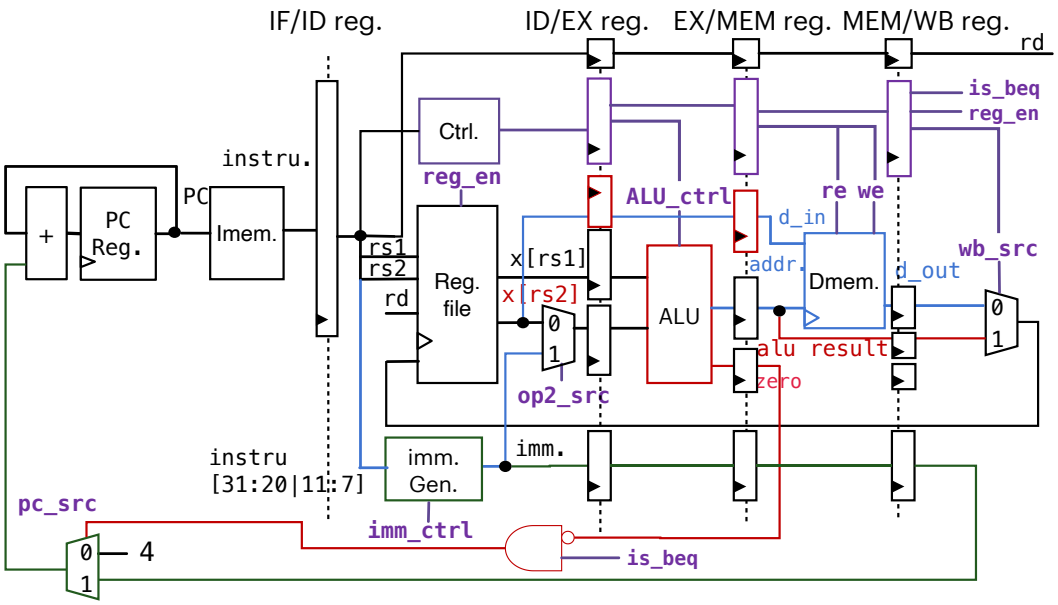


To be more precise, the '1' for `imm[31:12]` should be `FFFFF`, i.e., 20 '1's.

7. **Pipeline.** Consider a classical 5-stage pipelined processor (a reference design covered in the lecture shown below). Assume that the processor has no forwarding or hazard detection mechanisms implemented unless otherwise stated. Additionally, data and instruction memory are separated, and the register file can only be written one 32-bit value at the rising edge of the clock. The register file can be read and written simultaneously.

   The program executed on the processor is listed below:

```
1 lw t1, 0(a0)
2 add t2, t1, t1
3 xor t4, t3, t2
4 sub t6, t5, t2
5 sw t5, 1(t1)
```



6 (a) Identify the instructions and registers that cause data hazards using the blanks below, one item for each blank. Leave it blank if you do not use all of the blanks below.

> **Solution:** A. Register __t1__ in instructions __lw(1)__ and _add(2)_ causes a data hazard;
>
> B. Register _t2_ in instructions _add(2)_ and _xor(3)_ causes a data hazard;
>
> C. Register _t2_ in instructions _add(2)_ and _sub(4)_ causes a data hazard;
>
> D. Register _____ in instructions _____ and _____ causes a data hazard.
>
> If the fourth hazards or more are identified, minus 1 point.

4 (b) Which of the above data hazard(s) can be avoided completely, i.e., no stall or **nop** inserted to function correctly, by forwarding only? Use (A/B/C/D) in the previous question (a) to indicate your answer. ( )

> **Solution:** B and C depending on the order of your solution to the previous question (a). 1 point for not choosing A/D. 1 point for choosing B/C.

2 (c) Below shows the time for each stage. Please calculate the maximum frequency of this pipelined CPU.

IF	ID	EXE	MEM	WB
200 ps	100 ps	100 ps	500 ps	100 ps

> **Solution:** The maximum or longest delay of each stage is 500 ps. So $1/500$ ps $=$ 2 GHz

2 (d) **True of False.** After pipelining, the execution of each instruction (from IF to WB) takes a longer time than its un-pipelined version using the data from the previous question (c).

( )

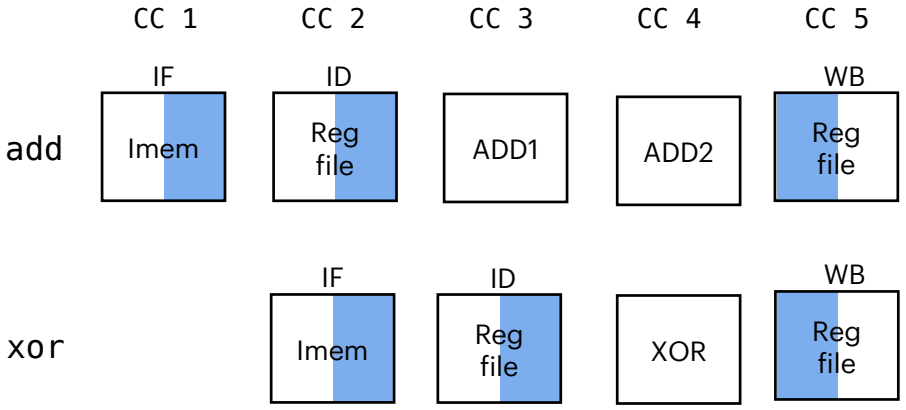
Solution: T.

2 (e) Assume we have a strange pipelined CPU design, where the execution of **`add`** instruction takes 5 stages and thus 5 clock cycles to complete, IF, ID, ADD1, ADD2 and WB. An **`xor`** instruction, on the other hand, takes 4 stages, IF, ID, XOR and WB. An assembly program

```
1 add t2, t1, t0
2 xor t3, t1, t0
```

Assume that the stages of the above two instructions are arranged in the time slots as shown in the figure below, _____ hazard occurs. "**CC**" stands for "clock cycle".

A. Structural B. Data
C. Control D. No



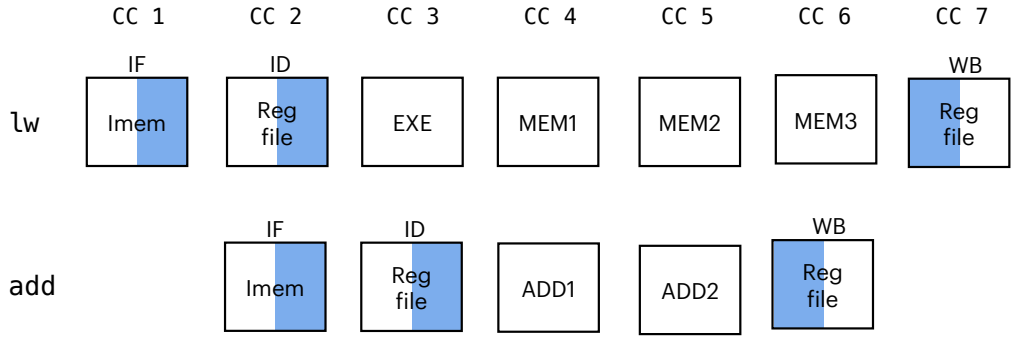
Solution: A. In the 5th clock cycle, both instructions write the register file, while in the question, we assume "the register file can only be written one 32-bit value at the rising edge of the clock". The ALU should not cause structural hazard since `add` uses ADD1 and ADD2, while `xor` uses XOR. But if we use the ALU design in the lecture (EXE stage for both `add` and `xor`), it causes also the structural hazard.

2 (f) For the same pipelined CPU in the previous question (e), a **`lw`** instruction takes 7 stages or clock cycles to complete, IF, ID, EXE, MEM1, MEM2, MEM3 and WB. For the following program,

```
1 lw t2, 0(a0)
2 add t2, t1, t0
```

If the stages of the instructions are arranged in the time slots as shown in the figure below. A _____ type data hazard occurs.

A. Read-after-write (RAW)
B. Write-after-read (WAR)
C. Write-after-write (WAW)
D. None of the above.



Solution: C. It is clearly a WAW hazard. `lw` changes the value of `t2` at the 7th clock cycle to a value in the memory. However, according to the program, the value of `t2` should be the sum of `t1` and `t0`.