Problem Out-of-Order Execution

For this problem, Lu Jiaxin wants to schedule the following codes on an out-of-order core.

The processor is a single-issue core with integer register (x0, x1, . . .) and float register (f0, f1, . . .). The processor uses free ROB entries from low index to high index. Instructions can commit one cycle after writeback, and ROB entries can be reused one cycle after commit. Instructions that depend on others can issue one cycle after the instruction it depends on writes back. Loads and stores take two cycles each, floating-point multiplies take three cycles, and floating-point adds take five cycles.

All functional units are fully pipelined. All the reservation stations are large enough.

Fill out the table with the cycles at which instructions enter the ROB, issue to the functional units, write back to the ROB, and commit. Also fill out the new register names for each instruction. If the instruction producing a source register had already committed before the dependent instruction enters the ROB, use the architectural register name.

Remember that instructions must enter the ROB and commit in order. On each cycle, only one instruction can enter the ROB, one can issue, one can write back, and one can commit.

1 Question A

Code A :

```
fmul.d  f0, f0, f1
fadd.d  f2, f2, f0
```

The ROB has two entries. Use r0-r1 for the two ROB entries.

	Time				Instruction			
	Enter ROB	Issue	WB	Commit	OP	Dest	Src1	Src2
I_1	-1	0	3	4	FMUL.D	r0	f0	f1
I_2	0	4	9	10	FADD.D	r1	f2	r0

2 Question B

Code B :

```
fmul.d  f0 , f0 , f1
fadd.d  f2 , f2 , f0
fadd.d  f2 , f2 , f0
```

The ROB has two entries. Use r0-r1 for the two ROB entries.

	Time				Instruction			
	Enter ROB	Issue	WB	Commit	OP	Dest	Src1	Src2
I_1	-1	0	3	4	FMUL.D	r0	f0	f1
I_2	0	4	9	10	FADD.D	r1	f2	r0
I_3	5	10	15	16	FADD.D	r0	r1	f0

3 Question C

Code C :

```
fld     f0 , 0(x1)
fld     f1 , 8(x1)
fmul.d  f0 , f0 , f1
fadd.d  f2 , f2 , f0
fld     f0 , 16(x1)
fadd.d  f2 , f2 , f0
```

The ROB has four entries. Use r0-r3 for the four ROB entries.

	Time				Instruction			
	Enter ROB	Issue	WB	Commit	OP	Dest	Src1	Src2
I_1	-1	0	2	3	FLD	r0	x1	
I_2	0	1	3	4	FLD	r1	x1	
I_3	1	4	7	8	FMUL.D	r2	r0	r1
I_4	2	8	13	14	FADD.D	r3	f2	r2
I_5	4	5	8	15	FLD	r0	x1	
I_6	5	14	19	20	FADD.D	r1	r3	r0

Problem Virtual Memory & Aliasing Problem

1 Question A

Suppose Lu Jiaxin has a virtual memory system with 2048-byte pages. Assume that physical addresses are 64 bits, each page table entry takes up 64 bits, and each level of the page table takes up a single page total. How many levels of paging would Lu Jiaxin need in order to cover 32 GB of virtual memory?

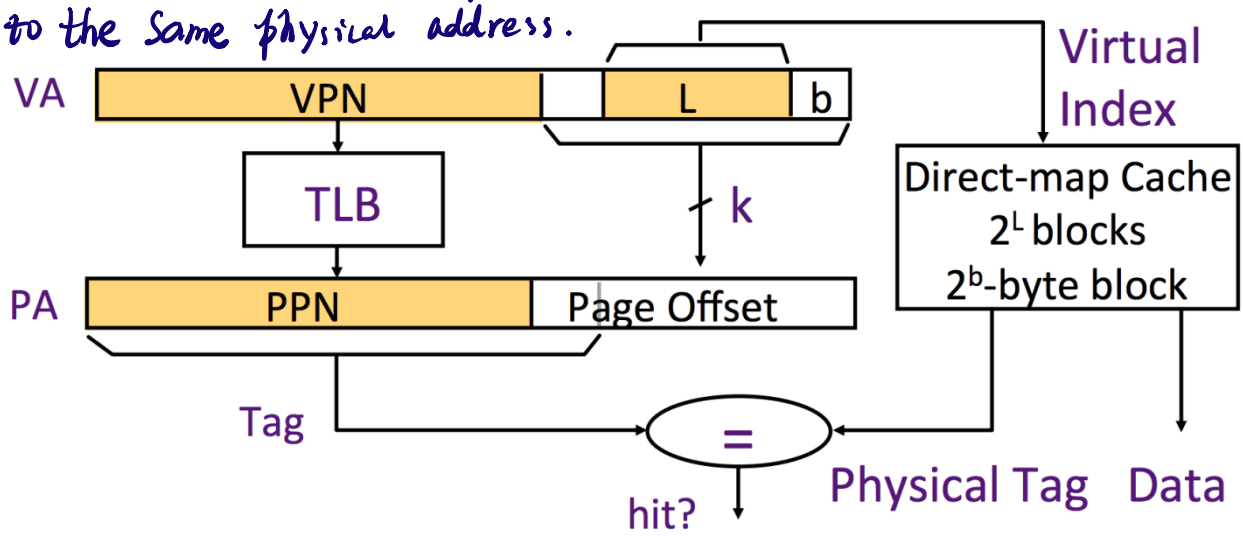
Note that $a = 2048 \text{ byte} / 64 \text{ bits} = 256 = 2^8$.

Hence, she need $N = \log_a \left(\frac{32 \times 2^{30}}{2048}\right) = 3$ levels.

2 Question B

What is the aliasing problem? Describe it in your words. Let's consider a feasible solution for the aliasing problem, Virtual Indexed & Physical Tagged Cache.

Aliasing problems come up when two or more virtual addresses are mapped to the same physical address.



To avoid the aliasing problem, what condition should meet among L, k, b? Why? (L is cache index. k is page offset. b is block offset)

Evidently, we ought to make sure $l+b \leq k$.

On the contrary, if $l+b > k$, cache size is smaller than page size, causing aliasing problem.

3 Question C

You are asked to design a virtually indexed, physically tagged cache. A page is 4096 bytes. The cache must have 256 lines of 64 bytes each. What associativity must the cache have to not worry about aliases? (Hint : cache size is bigger than page size, so the cache should be set- associative)

$$N = \frac{\text{Cache size}}{\text{page size}} = \frac{256 \times 64}{4096} = 4.$$

Therefore, we ought to use 4-way associate cache.

Problem Branch Prediction

For the following question, we are interested in the performance of branches when executing the following code. For each part, assume that N = 4 and the array A has the values {1, 7, 2, 5}.

C code	RISC-V Assembly

`for (int i = 0; i < N; i++) {`
`  int b = A[i];`
`  if (b >= 5)`
`  c += b;`
`  if (b < 4)`
`  c -= b;`
`}`
	`  li x1, A`
	`  li x4, N`
	`loop:`
	`  lw x2, 0(x1)`
	`  li x5, 5`
	`  blt x2, x5, skip1`
	`  add x3, x3, x2`
	`skip1:`
	`  li x5, 4`
	`  bge x2, x5, skip2`
	`  sub x3, x3, x2`
	`skip2:`
	`  addi x1, x1, 4`
	`  addi x4, x4, -1`
	`  bnez x4, loop`

1 Question A

Fill out the table to show what the predictions will be if the branch predictor is a BHT indexed by PC with two-bit counters. If the most- significant bit of a counter is 1, the predictor predicts taken. Otherwise it predicts not taken. The counters are initialized to weakly not-taken (01). The Counter column in the table shows the state of the counter before the branch is executed. Assume that the BHT is large enough that no aliasing of instruction addresses will occur.

	Instruction	Counter	Prediction	Actual	
i = 0	blt x2 x5, skip 1	01	not taken	taken	✗
	bge x2, x5, skip2	01	not taken	not taken	✓
	bnez x4, loop	01	not taken	taken	✗
i = 1	blt x2 x5, skip 1	10	taken	not taken	✗
	bge x2, x5, skip2	00	not taken	taken	✗
	bnez x4, loop	10	taken	taken	✓
i = 2	blt x2 x5, skip 1	01	not taken	taken	✗
	bge x2, x5, skip2	01	not taken	not taken	✓
	bnez x4, loop	11	taken	taken	✓
i = 3	blt x2 x5, skip 1	10	taken	not taken	✗
	bge x2, x5, skip2	00	not taken	taken	✗
	bnez x4, loop	11	taken	not taken	✗

What is the prediction accuracy for each branch? What is the prediction accuracy overall?

Branch	Accuracy
blt	0%
bge	50%
bnez	50%
overall	33%

2 Question B

Now assume we change the branch predictor to a BHT indexed by PC and a single bit of global history. Assume the global history is initialized to 0, the counters are initialized to 01 (weakly not-taken), and there is no aliasing. The Counter columns in the table show the state of the counters before the branch is executed. Fill out the table with the predictions.

	Instruction	Global History	Counter 0	Counter 1	Prediction	Actual
i = 0	blt x2 x5, skip 1	0	01	01	not taken	taken
	bge x2, x5, skip2	1	01	01	not taken	not taken
	bnez x4, loop	0	01	01	not taken	taken
i = 1	blt x2 x5, skip 1	1	10	01	not taken	not taken
	bge x2, x5, skip2	0	01	00	not taken	taken
	bnez x4, loop	1	10	01	not taken	taken
i = 2	blt x2 x5, skip 1	1	10	00	not taken	taken
	bge x2, x5, skip2	1	10	00	not taken	not taken
	bnez x4, loop	0	10	10	taken	taken
i = 3	blt x2 x5, skip 1	1	10	01	not taken	not taken
	bge x2, x5, skip2	0	10	00	taken	taken
	bnez x4, loop	1	11	10	taken	not taken

What is the prediction accuracy for each branch? What is the prediction accuracy overall?

Branch	Accuracy
blt	50%
bge	75%
bnez	25%
overall	50%

3 Question C

If you run this code with a large array containing uniformly randomly distributed values, which branch do you expect to get the most benefit from global history and why?

bge x_2 x_5 skip2. Because this branch is highly related with the branch before it. Actually, if skip1 is taken, skip2 will not be taken, and if skip1 is not taken, skip2 will taken.

4 Question D

Explain the motivation for using both BHT and BTB branch-prediction structures in the same implementation.

BTB comes into play in IF stage, and if hits, the penalty for the branch is reduced to zero. When BTB misses, we can use BHT in ID stage as plan B, allowing us to make fewer mispredictions. In this way, the performance is enhanced compared to using BTB or BHT alone.