Homework 3

Fall 2021

Deadline: Thursday, Nov. 11, 11:55 PM

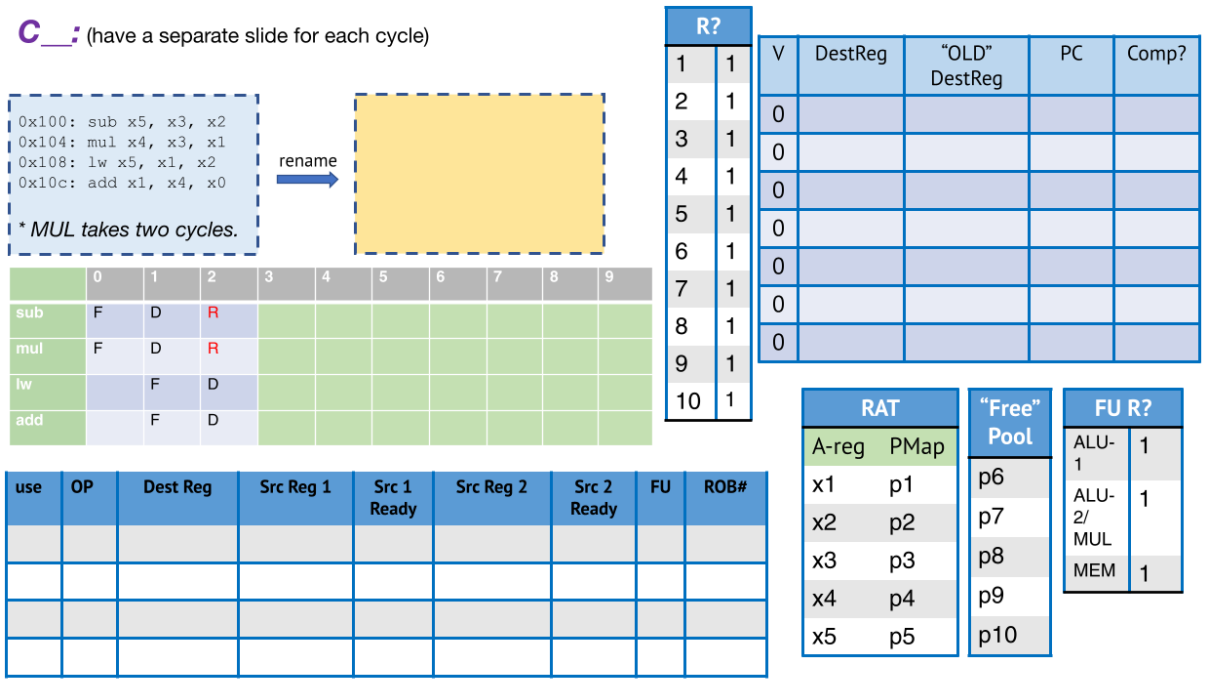
(Upload it to Gradescope.)

Q1. We have the following set of instructions (every instruction takes one cycle to complete except MUL):

```
0x100: sub x5, x3, x2
0x104: mul x4, x3, x1
0x108: lw x5, x1, x2
0x10c: add x1, x4, x0
```

* MUL takes two cycles.

We have a 2-issue out-of-order processor with the following units: (you can access the template by selecting the image and clicking on the link that appears).

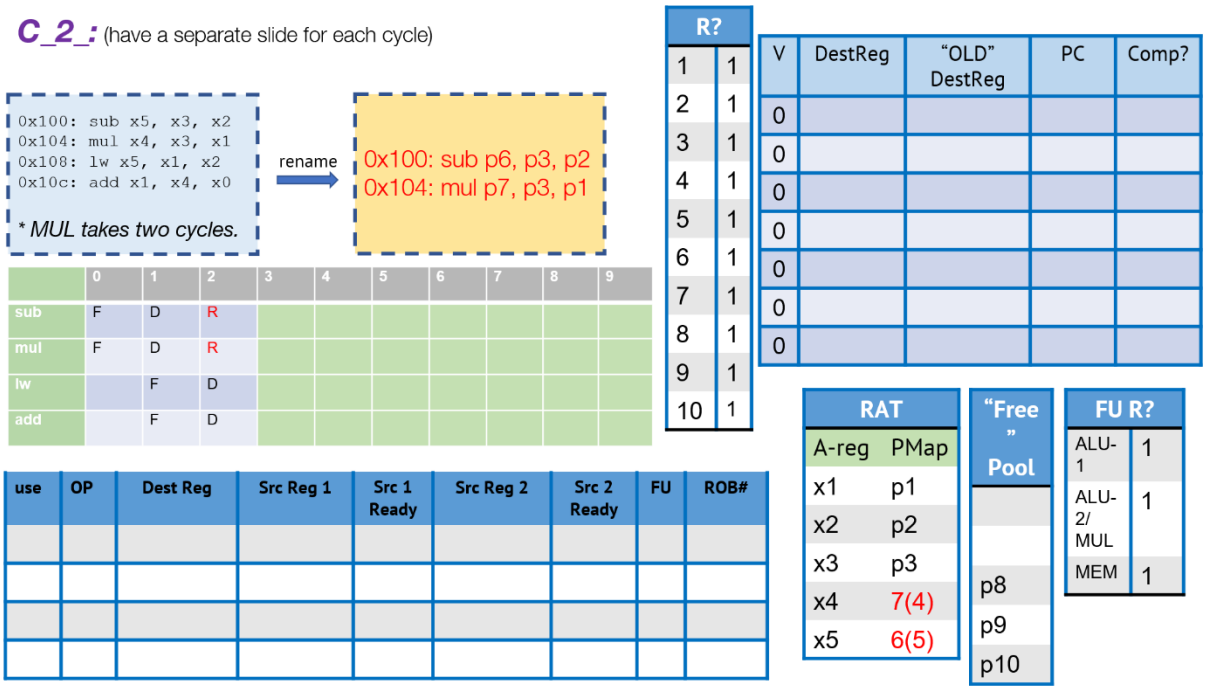


Start from cycle 2, with Rename, and fill all the tables. In the green table (the one above the reservation station on the left), you should put R, Di (for dispatch), I (for issue), C, and Rt (for retire). If an instruction stays in one stage for multiple cycles, repeat the same character for it (e.g., F, D, R, Di, Di, I, C, C, Rt). `Comp` in ROB shows whether the

instruction is completed or not. (Assume that you can retire a maximum of two instructions per cycle.)

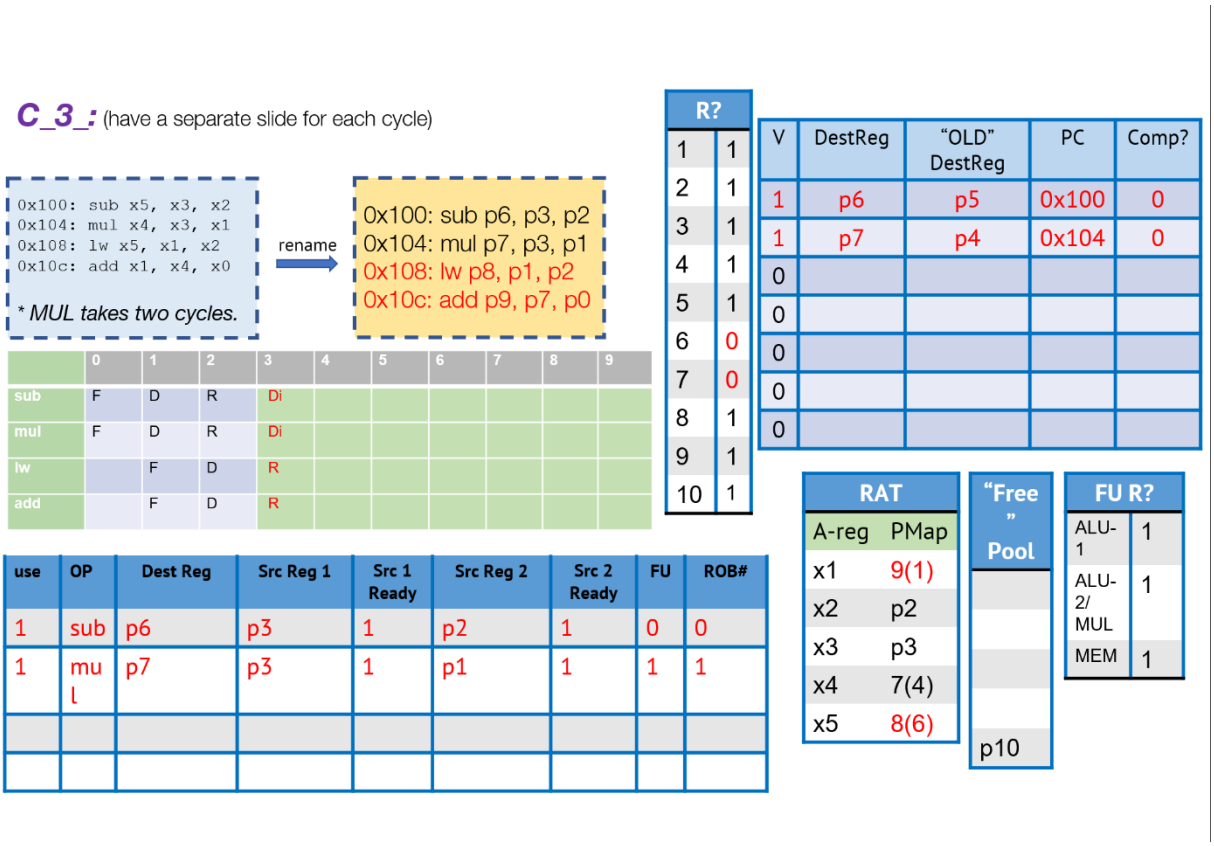
SOLUTION:

In cycle 2, we rename the destination registers of the sub and mul instructions (updating the free pool and the RAT in the process).



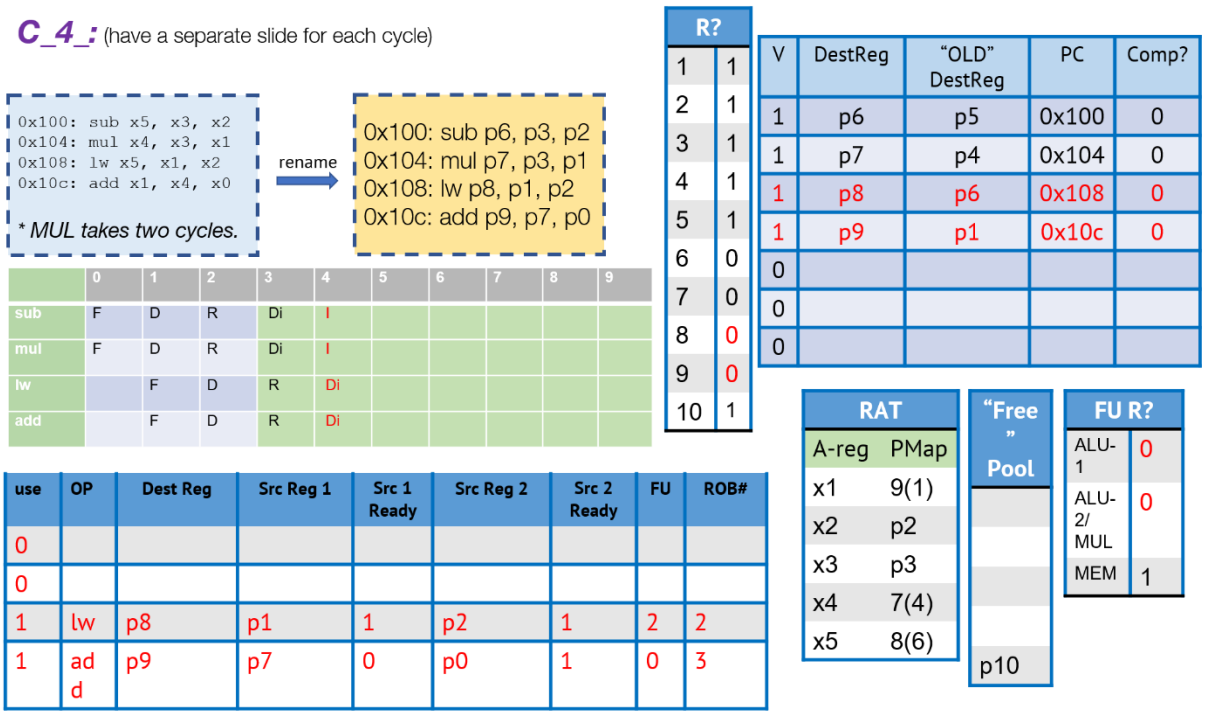
In cycle 3, we need to dispatch the sub and mul instructions, meaning we place them in the reservation station and in the ROB. In the readiness table, we mark the destination registers as "not ready". We look at the readiness table to see if the source registers are ready (they are for both instructions).

We rename the lw and add instructions as well.

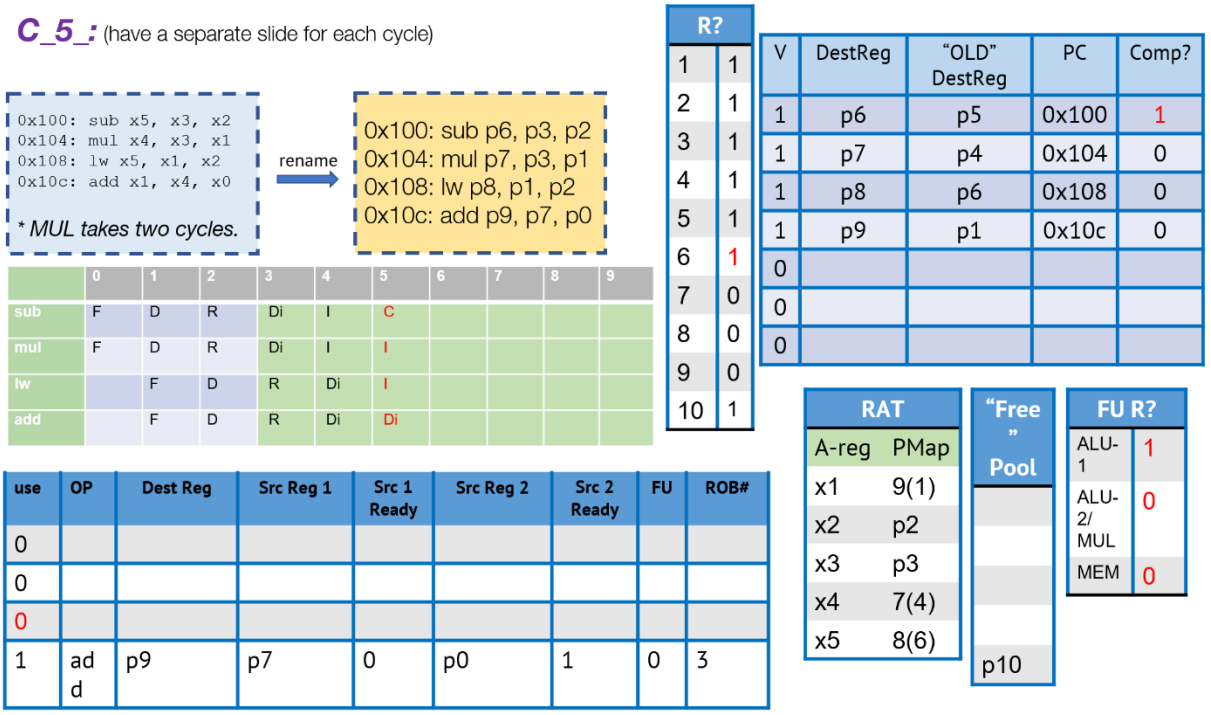


In cycle 4, we issue the sub and mul instructions (the source registers for both instructions are ready, and the functional units are ready). We also clear them from the reservation station. We update the respective functional units to "not ready".

We also dispatch the lw and add instructions. We update the readiness table and add these instructions to the reservation station and the ROB. Note that p7 is not ready yet (used by mul).



In cycle 5, we complete the sub instruction. We mark the instruction as "complete" in the ROB and free the functional unit. The mul is still in issue (takes 2 cycles). Thus, we can't issue the add instruction (it depends on the result of the mul), but we can issue the lw.



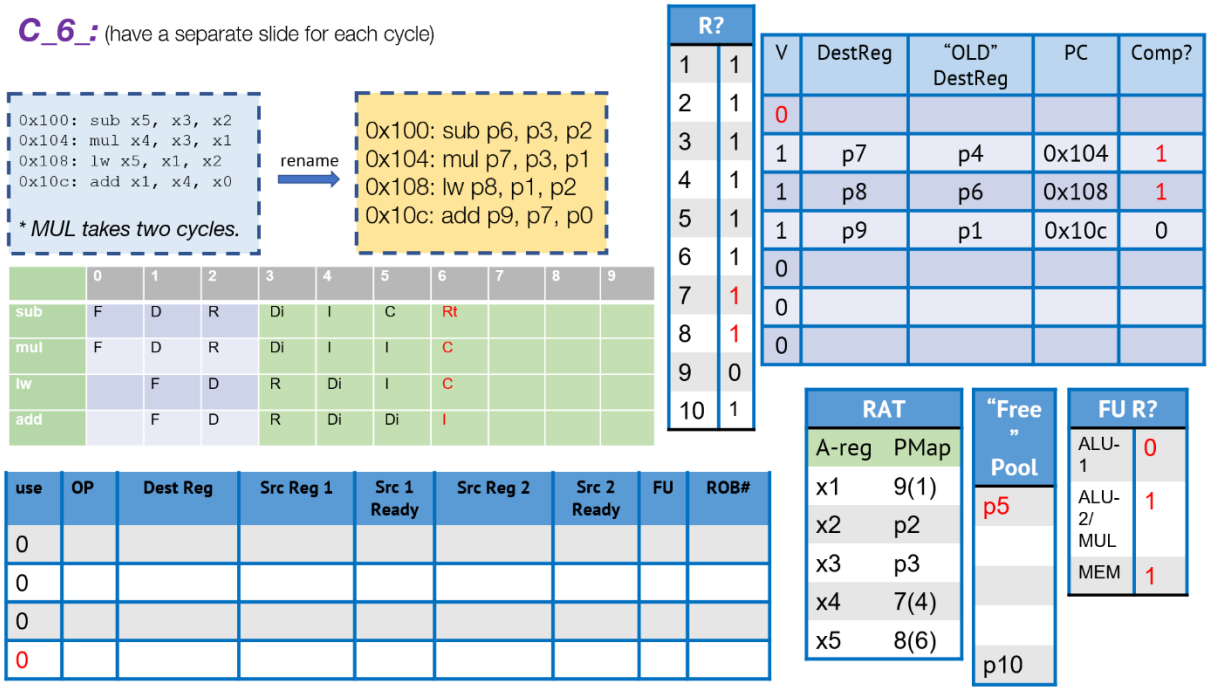
	0	1	2	3	4	5	6	7	8	9
sub	F	D	R	Di	I	C				
mul	F	D	R	Di	I	I				
lw		F	D	R	Di	I				
add		F	D	R	Di	Di				

use	OP	Dest Reg	Src Reg 1	Src 1 Ready	Src Reg 2	Src 2 Ready	FU	ROB#
0								
0								
0								
1	add	p9	p7	0	p0	1	0	3

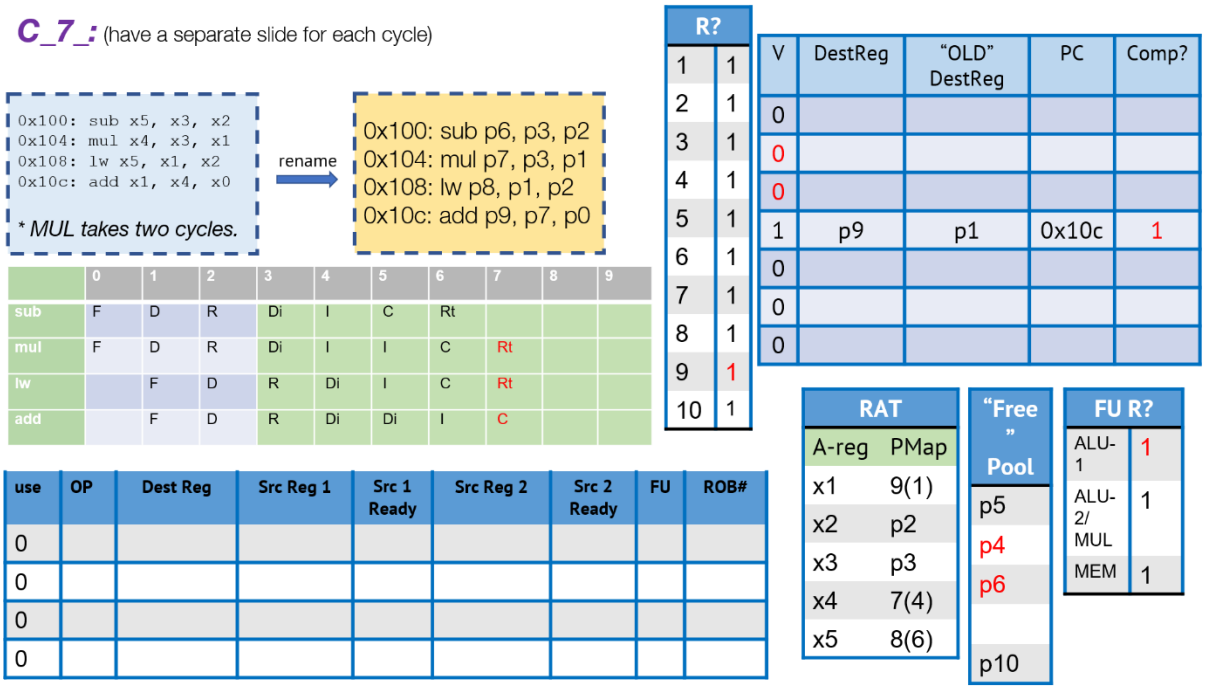
In cycle 6, we can retire the sub. We clear it from the ROB and add p5 back to the free pool (p5 was the old physical register. When we retire in order, we can guarantee that p5 is free).

We complete the mul and lw instructions.

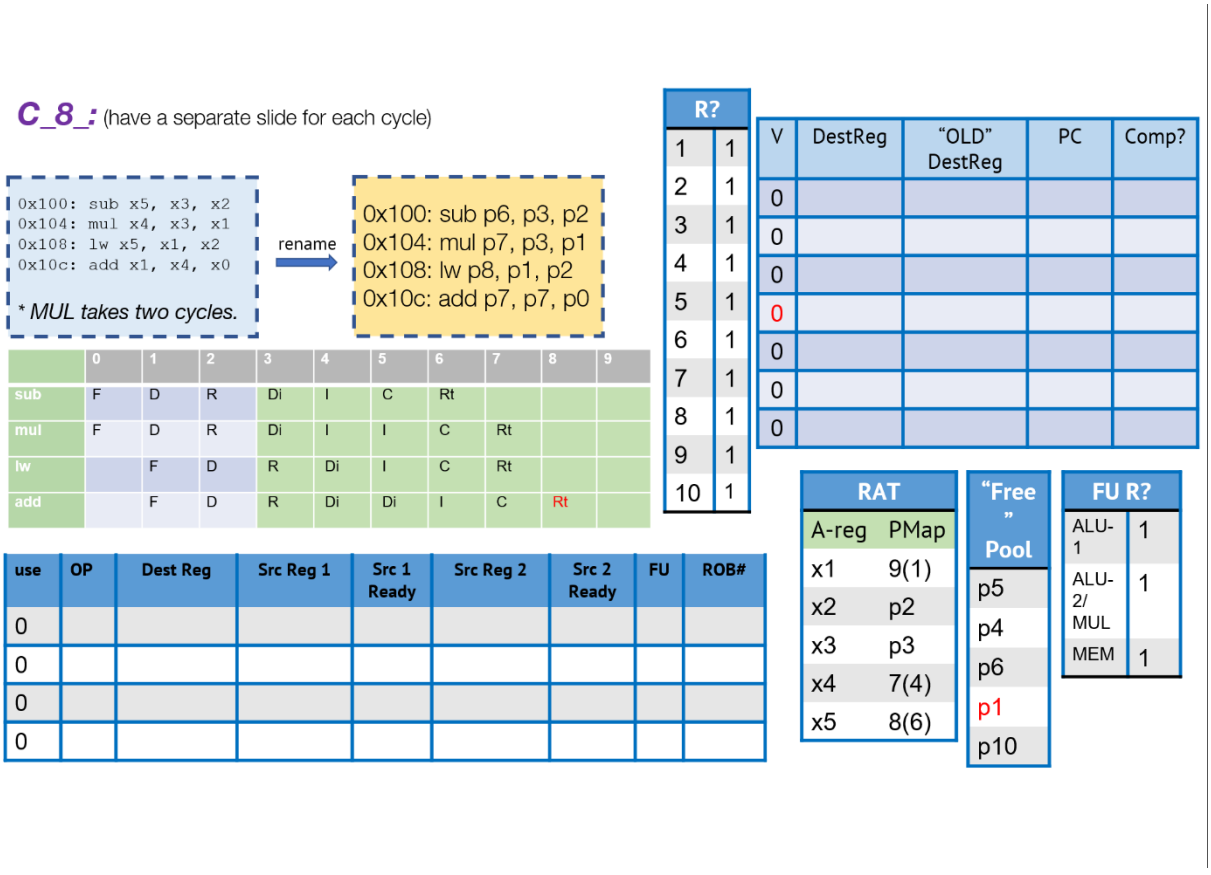
We can also issue the add instruction as p7 is now ready to be used.



In cycle 7, we retire the mul and lw instructions, freeing p4 and p6 in the process. The add instruction is now complete.



Finally, in cycle 8, we retire the add instruction, freeing p1.



Q2. Assume that we have two caches, a direct-mapped, and a 4-way set-associative. Assume that the addresses are 12 bits. Assume that the Direct-mapped cache has 8 lines/sets, thus the 4-way set-associative cache should have 2 sets (why?). Answer the following questions:

NOTE: The addresses are CPU addresses (so they need to be shifted).

a. Assuming that each line is 32 bytes (i.e., we need 5 bits for offset), what is the index and tag size for each of the caches?

SOLUTION:

Direct-mapped cache:

Index Size: 3 bits (need 3 bits for 8 cache slots)

Tag Size: 4 bits (12 total bits, minus 3 bits for index, minus 5 bits from CPU address shift).

4-way set-associative cache:

Index Size: 1 bit (need 1 bit for 2 sets)

Tag Size: 6 bits (12 total bits, minus 1 bit for index, minus 5 bits from CPU address shift).

b. Complete the following table for the direct-mapped cache. ('inv'=invalid and the column of a particular cache line contains the tag of that line).

SOLUTION:

To convert hex address 136 into a cache address:

136 in binary = 000100110110

Offset: 10110 = 22 in decimal. This address is the 22nd position byte within this cache line. Since the granularity of our CPU address is higher than our cache address (we bring in 32 bytes worth), we shift out this offset from our CPU address in the calculation of tag and index.

Index: 001 = 1 in decimal. This is the 1st index into our cache.

Tag: 0001 = 1 in hex. This is the tag we place into our table to identify the location of memory brought into the cache.

The existing tag in position C1 matches our new tag, so it is a hit.

To convert hex address 1A3 into a cache address:

1A3 in binary = 000110100011

Offset: 00011 = 3 in decimal. The 3rd position byte in our cache line.

Index: 101 = 5 in decimal. This is the 5th index into our cache.

Tag: 0001 = 1 in hex.

The existing tag in position C5 is inv, so it is a miss and we store the new data into C5 (as well as the tag).

To convert hex address 102 into a cache address:

102 in binary = 000100000010

Offset: 00010 = 2 in decimal. The 2nd position byte in our cache line

Index: 000 = 0 in decimal. This is the 0th index into our cache.

Tag: 0001 = 1 in hex.

The existing tag in position C0 is 2 (which does not equal the new tag of 1), so it is a miss and we store the new data into C0 (as well as the tag).

We do this CPU hex address conversion to cache address for every address and analyze whether it is a hit or not, replacing the cache line if there is a miss.

DM	Addresses (coming from the CPU) and tags are shown in HEX. Compute the tag for each Cell/Line (C0, C1, ...)								
Address	C0	C1	C2	C3	C4	C5	C6	C7	Hit/Miss
110	1	inv	inv	inv	Inv	inv	inv	inv	M
136	1	1	inv	inv	Inv	inv	inv	inv	M

202	2	1	inv	inv	Inv	inv	inv	inv	M
136	2	1	inv	inv	Inv	inv	inv	inv	H
1A3	2	1	inv	inv	Inv	1	inv	inv	M
102	1	1	inv	inv	Inv	1	inv	inv	M
361	1	1	inv	3	inv	1	inv	inv	M
102	1	1	inv	3	inv	1	inv	inv	H
274	1	1	inv	2	inv	1	inv	inv	M
114	1	1	inv	2	inv	1	inv	inv	H
1A4	1	1	inv	2	inv	1	inv	inv	H
177	1	1	inv	1	inv	1	inv	inv	M
301	3	1	inv	1	inv	1	inv	inv	M
276	3	1	inv	2	inv	1	inv	inv	M
135	3	1	inv	2	inv	1	inv	inv	H

c. Assume that the SA cache uses a first-in-first-out (FIFO) replacement policy. Complete the table for this cache structure.

SOLUTION:

To convert hex address 136 into a cache address:

136 in binary = 000100110110

Offset: 10110 = 22 in decimal. The 22nd position byte in our cache line.

Index: 1 = 1 in decimal. This is the 1st set of our cache.

Tag: 000100 = 4 in hex.

There is an existing tag in set 1 of our cache that matches our new tag, so this is a hit.

To convert hex address 1A3 into a cache address:

1A3 in binary = 000110100011

Offset: 00011 = 3 in decimal. The 3rd position byte in our cache line.

Index: 1

Tag: 000110 = 6 in hex.

In set 1 of our cache (index 1), we do not have a tag of 6 present, so it is a miss and we need to store the new data (and the tag) into set 1. We pick an empty way (W1), as within a set is associative.

To convert hex address 102 into a cache address:

102 in binary = 000100000010

Offset: 00010 = 2 in decimal. The 2nd position byte in our cache line

Index: 0 = 0 in decimal. This is set 0 in our cache.

Tag: 000100 = 4 in hex.

In set 0, tag 4 already exists (in Way 0), thus this is a hit.

FIFO policy means when a set is full, we evict the entry that was first placed in that set. Until the given set is full, we do not need to evict, as we just insert our entry into an empty block in our set.

The call on Address 177 causes the first eviction, as Set 1 had become full. Since W0 contains the first entry placed into Set 1, we evict the entry at W0 and place the new entry belonging to Address 177 into W0.

The 2nd eviction happens with the call on Address 135. We evict the entry at W1 as W1 is the current entry that has been in Set 1 the longest and place the new entry belonging to Address 135 into W1.

SA **FIFO**	Addresses and tags are shown in HEX. Compute the tag for each Way (W0, W1, ...).

	Set 0				Set 1				
Address	W0	W1	W2	W3	W0	W1	W2	W3	Hit/Miss
110	4	inv	inv	inv	inv	inv	inv	inv	M
136	4	inv	inv	inv	4	inv	inv	inv	M
202	4	8	inv	inv	4	inv	inv	inv	M
136	4	8	inv	inv	4	inv	inv	inv	H
1A3	4	8	inv	inv	4	6	inv	inv	M
102	4	8	inv	inv	4	6	inv	inv	H
361	4	8	inv	inv	4	6	D	inv	M
102	4	8	inv	inv	4	6	D	inv	H
274	4	8	inv	inv	4	6	D	9	M
114	4	8	inv	inv	4	6	D	9	H
1A4	4	8	inv	inv	4	6	D	9	H
177	4	8	inv	inv	5	6	D	9	M
301	4	8	C	inv	5	6	D	9	M
276	4	8	C	inv	5	6	D	9	H
135	4	8	C	inv	5	4	D	9	M

d. Repeat this if the SA cache uses LRU. (LRU bits are not shown, you should keep track of them separately, but don't need to show it here).

SOLUTION:

LRU means Least Recently Used. The least recently used entry, as in the block in the given set that has been accessed the longest time ago (activity being either a cache miss or a cache hit to the block), is evicted. In the table below, for clarity, we show LRU via the number in parentheses next to the Tag. (0) represents the least recently used entry while 3 represents the most recently used entry.

The first eviction happens with the call on Address 177. We still end up evicting the entry at W0 in Set 1 as block W0 is the least recently used.

The second eviction results in a different behavior than for FIFO. With the call on Address 135, we evict the entry at W2 in Set 1 as block W2 is the least recently used.

SA **LRU**	Addresses and tags are shown in HEX. Compute the tag for each Way (W0, W1, …).								
	Set 0				Set 1				
Address	W0	W1	W2	W3	W0	W1	W2	W3	Hit/Miss
110	4 (0)	inv	inv	inv	inv	inv	inv	inv	M
136	4 (0)	inv	inv	inv	4 (0)	inv	inv	inv	M
202	4 (0)	8 (1)	inv	inv	4 (0)	inv	inv	inv	M
136	4 (0)	8 (1)	inv	inv	4 (0)	inv	inv	inv	H
1A3	4 (0)	8 (1)	inv	inv	4 (0)	6 (1)	inv	inv	M
102	4 (1)	8 (0)	inv	inv	4 (0)	6 (1)	inv	inv	H
361	4 (1)	8 (0)	inv	inv	4 (0)	6 (1)	D (2)	Inv	M
102	4 (1)	8 (0)	inv	inv	4 (0)	6 (1)	D (2)	Inv	H
274	4 (1)	8 (0)	inv	inv	4 (0)	6 (1)	D (2)	9 (3)	M

114	4 (1)	8 (0)	inv	inv	4 (0)	6 (1)	D (2)	9 (3)	H
1A4	4 (1)	8 (0)	inv	inv	4 (0)	6 (3)	D (1)	9 (2)	H
177	4 (1)	8 (0)	inv	inv	5 (3)	6 (2)	D (0)	9 (1)	M
301	4 (1)	8 (0)	C (2)	inv	5 (3)	6 (2)	D (0)	9 (1)	M
276	4 (1)	8 (0)	C (2)	inv	5 (2)	6 (1)	D (0)	9 (3)	H
135	4 (1)	8 (0)	C (2)	inv	5 (1)	6 (0)	4 (3)	9 (2)	M

e. What is the miss rate for each case (DM, SA with FIFO, and SA with LRU)?

SOLUTION:

Miss Rate Formula: $\frac{Number\ of\ Cache\ Misses}{Number\ of\ Accesses}$ **x 100**

DM Miss Rate: $\frac{10}{15}$ **x 100 = 66.66%**

SA with FIFO Miss Rate: $\frac{9}{15}$ **x 100 = 60%**

SA with LRU Miss Rate: $\frac{9}{15}$ **x 100 = 60%**