(a)[1.6 pts] Consider a processor with a 2-way set-associative cache with one-word cache blocks and a total cache size of 16 words. The cache uses a least recently used (LRU) replacement policy and is initially empty.

The following sequence of decimal word address references is seen by the cache:
7, 47, 77, 47, 7, 8, 18, 48, 88, 8, 18, 48, 88, 7, 47, 77, 47, 4, 8, 16, 32, 64, 4, 7, 77, 32, 37, 23, 16, 21, 77, 64

(i) Indicate whether each address reference is a hit or a miss.

Address	Set	Hit/Miss
7	7	M
47	7	M
77	5	M
47	7	H
7	7	H
8	0	M
18	2	M
48	0	M
88	0	M
8	0	M
18	2	H
48	0	M
88	0	M
7	7	H
47	7	H
77	5	H
47	7	H
4	4	M
8	0	M
16	0	M
32	0	M
64	0	M
4	4	M
7	7	H
77	5	H
32	0	M
37	5	M
23	7	M
16	0	M
21	5	M
77	5	M
64	0	M

(ii) Show the final cache contents.

Set	Contents
0	64
	16
1	
2	18
3	
4	4
5	77
	21
6	
7	23
	7

(b) [1.0 pts] Consider the following short program executing on a simple 5-stage in-order pipeline (Fetch, Decode, Execute, Memory, Writeback). For arithmetic instructions, the destination register is listed first, followed by the source registers. For example, ADD R3, R2, R1 adds the contents of R1 and R2 and stores the result in R3.

```
I1: LW   R1, 0(R2)       ;load R1 from address 0+R2
I2: LW   R3, 0(R4)       ;load R3 from address 0+R4
I3: ADD  R5, R1, R3      ;R5 = R1 + R3
I4: SUB  R6, R7, R5      ;R6 = R7 - R5
I5: SW   R6, 4(R2)       ;store R6 to address 4+R2
```

Assume a pipeline that implements forwarding paths, but only from the output of the execute stage (output of Execute/Memory pipeline register) to the input of the execute stage. Insert NOP instructions in the above instruction schedule to avoid any hazards. Show the execution schedule. How many cycles does it take to execute the modified code (with NOPs inserted)? Assume that a register read can take place in the same cycle that a value is written back to the register file.

I1	F	D	X	M	W						
I2		F	D	X	M	W					
NOP			NOP								
NOP				NOP							
I3					F	D	X	M	W		
I4						F	D	X	M	W	
I5							F	D	X	M	W
Cycle	1	2	3	4	5	6	7	8	9	10	11

It takes 11 cycles to execute the code with NOPs inserted.

(c) [1.0 pts] A student executes a program on a single core of a 64-core processor. The student uses gprof to profile the code and observes the following output. (gprof shows the percentage of execution time spent executing each function in the program.)

NAME	TIME	%
swim	2.4	1.7
bike	112.0	79.7
run	26.2	18.6

Observing that most of the execution time is spent executing the "bike" function, the student decides to write a new version of the program in which the "bike" function is replaced by a parallel implementation of the function. What is the fastest execution time the student can expect when running the parallel version of the program on the 64-core processor? Also, what is the corresponding best speedup the student can expect with the parallel version of the program running on the 64-core processor?

$$parallel\ execution\ time(N = 64) = 28.6 + \frac{112}{64} \approx 30.4$$

$$speedup(N = 64) = \frac{1}{(1 - P) + \frac{P}{64}} = \frac{1}{0.203 + \frac{0.797}{64}} = \frac{2.4 + 112 + 26.2}{\frac{112}{64} + 2.4 + 26.2} \approx 4.6$$

In future processor generations, the number of cores in a parallel processor tends to increase. If this trend continues indefinitely, what is the limit of the speedup that can be achieved by the student's code running on a parallel processor?

$$maximum\ speedup = \lim_{N \to \infty} \frac{1}{(1 - P) + \frac{P}{N}} = \lim_{N \to \infty} \frac{1}{0.203 + \frac{0.797}{\infty}} = 4.9$$

Given that the student's program is fixed, what is the type of scaling that describes how the execution time of the program varies as the number of processor cores increases?

Strong scaling

(d) [0.4 pts] Register renaming is a technique that can be used to eliminate hazards in a processor pipeline. What type of data hazards can be eliminated by register renaming?

Register renaming can eliminate WAW and WAR hazards (false dependencies).