(a) [1.4 pt] Consider a processor with a 4-way set-associative cache with one-word blocks and a total cache size of 32 words. The cache uses a least recently used replacement policy and is initially empty.

The following sequence of decimal word address references is seen by the cache.

30, 86, 53, 61, 29, 37, 30, 45, 6, 22, 14, 6, 53, 29, 78, 22, 70, 61, 54, 78, 45, 30, 61, 37, 45, 6, 29

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

Address	Set	Hit/Miss
30	2	Miss
86	2	Miss
53	1	Miss
61	1	Miss
29	1	Miss
37	1	Miss
30	2	Hit
45	1	Miss
6	2	Miss
22	2	Miss
14	2	Miss
6	2	Hit
53	1	Miss
29	1	Hit
78	2	Miss
22	2	Hit
70	2	Miss
61	1	Miss
54	2	Miss
78	2	Hit
45	1	Hit
30	2	Miss
61	1	Hit
37	1	Miss
45	1	Hit
6	2	Miss
29	1	Hit

ii. Show the final cache contents.

Set	Contents
0	
1	~~37~~, 61
	~~61, 53~~, 37
	~~53~~, 45
	29
2	~~6~~, 54
	~~30,~~ 78
	~~22~~, 30
	~~86, 14, 70~~, 6
3	
4	
5	
6	
7	

(b) [2 pt] Consider the following short program executing on a simple 5-stage pipeline (Fetch, Decode, Execute, Memory, Writeback). In the notation used below, $N denotes register N. For arithmetic instructions, the destination register is listed first, followed by the source operands. For example, add $3, $2, $1 adds the contents of registers 1 and 2 and stores the result in register 3.

(1) lw $1, 40($6)
(2) add $6, $2, $2
(3) sw $6, 50($1)
(4) add $4, $5, $6
(5) lw $6, 10($4)

i. Identify all the data dependencies in the code given above.

$6 in Instr (2) has WAR dependency with instr (1)
$1 in Instr (3) has RAW dependency with instr (1)
$6 in Instr (3) has RAW dependency with instr (2)
$6 in Instr (4) has RAW dependency with instr (2)
$4 in Instr (5) has RAW dependency with instr (4)
$6 in Instr (5) has WAR dependency with instr (4)

ii. Assume a pipeline that does not implement forwarding. Identify which dependencies from part (i) will cause data hazards if NOPs are not inserted.

Instr (1)	IF	ID	EXE	MEM	WB				
Instr (2)		IF	ID	EXE	MEM	WB/WB			
Instr (3)			IF	ID/ID	EXE	MEM	WB		
Instr (4)				IF	ID	EXE	MEM	WB	
Instr (5)					IF	ID	EXE	MEM	WB
Cycle	1	2	3	4	5	6	7	8	9

Instr (1) loads value to $1 at cycle 5, but instr (3) reads $1 at cycle 4.
Instr (2) writes value to $6 at cycle 6, but instr (3) reads $6 at cycle 4.
Instr (2) writes value to $6 at cycle 6, but instr (4) reads $6 at cycle 5.
Instr (4) writes value to $4 at cycle 8, but instr (5) reads $4 at cycle 6.

iii. If NOP instructions are inserted to avoid hazards in absence of forwarding hardware, how many cycles does it take to execute the code?

Instr (1)	IF	ID	EXE	MEM	WB								
Instr (2)		IF	ID	EXE	MEM	WB							
NOP			NOP										
NOP				NOP									
Instr (3)					IF	ID	EXE	MEM	WB				
Instr (4)						IF	ID	EXE	MEM	WB			
NOP							NOP						
NOP								NOP					
Instr (5)									IF	ID	EXE	MEM	WB
Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13

It takes 13 cycles to execute this code.

iv. Now assume that forwarding paths are added to the pipeline and NOPs are added only in cases where forwarding does not resolve a hazard. With forwarding, how many cycles will it take to execute the code?

	IF	ID	EXE	MEM	WB
Cycle 1	lw $1, 40($6)				
Cycle 2	add $6, $2, $2	lw $1, 40($6)			
Cycle 3	sw $6, 50($1)	add $6, $2, $2	lw $1, 40($6)		
Cycle 4	add $4, $5, $6	sw $6, 50($1)	add $6, $2, $2	lw $1, 40($6)	
Cycle 5	lw $6, 10($4)	add $4, $5, $6	sw $6, 50($1)	add $6, $2, $2	lw $1, 40($6)
Cycle 6		lw $6, 10($4)	add $4, $5, $6	sw $6, 50($1)	add $6, $2, $2
Cycle 7			lw $6, 10($4)	add $4, $5, $6	sw $6, 50($1)
Cycle 8				lw $6, 10($4)	add $4, $5, $6
Cycle 9					lw $6, 10($4)

Instr (1) forwards the load result ($1) from MEM/WB register to ID/EXE as an ALU input at cycle 5.
Instr (2) forwards the add result ($6) from EXE/MEM register to ID/EXE as a store input at cycle 5.
Instr (2) forwards the add result ($6) from MEM/WB register to ID/EXE as an ALU input at cycle 6.
Instr (4) forwards the add result ($4) from EXE/MEM register to ID/EXE as an ALU input at cycle 7.

With forwarding, it takes 9 cycles to execute the code.

(c) [0.6 pt] A student runs a serial (non-parallel) program on a single core of a 128-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	%
work	4163	98
play	85	2

Observing that most of the execution time is spent executing the "work" function, the student decides to write a new version of the program in which the "work" 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 128-core processor?

$$parallel\ execution\ time(N = 128) = 85 + \frac{4163}{128} \approx 117.5$$

$$speedup(N = 128) = \frac{1}{(1-P) + \frac{P}{128}} = \frac{1}{0.02 + \frac{0.98}{128}} = \frac{4163 + 85}{\frac{4163}{128} + 85} \approx 36$$

Given a processor with unlimited cores, what is the maximum speedup that can be achieved by the student's code?

$$maximum\ speedup = \lim_{N\to\infty} \frac{1}{(1-P) + \frac{P}{N}} = \lim_{N\to\infty} \frac{1}{0.02 + \frac{0.98}{\infty}} = 50$$