1. Imagine we have a non-pipelined processor running at 1MHz and want to run a program with 1000 instructions.

a) How much time would it take to execute the program?

1 instruction per cycle·

1MHz clock

1000 instructions = 1000 cycles @ (10^6)Hz = $1000/10^6$ = 1000us

Assuming ideal conditions (perfect pipelining and no hazards), how much time would it take to execute the same program in:

b) A 5-stage pipeline?

1 instruction per cycle·

5MHz clock

+4 extra cycles to load the pipeline

(1000 + 4 cycles)/$(5*10^6)$Hz = 200·8us ~5x faster

c) A 20-stage pipeline?

1 instruction per cycle·

20MHz clock

+19 extra cycles to load the pipeline

(1000 + 19 cycles)/$(20*10^6)$Hz = 50·95us ~20x faster

d) A 100-stage pipeline?

1 instruction per cycle·

100MHz clock

+99 extra cycles to load the pipeline

(1000 + 99 cycles)/$(100*10^6)$Hz = 10·99us ~100x faster

Looking at those results, it seems clear that increasing pipeline should increase the execution speed of a processor. Why do you think that processor designers (see below) have not only stopped increasing pipeline length but, in fact, reduced it?

Pentium III – Coppermine (1999) – 10-stage pipeline
Pentium IV – NetBurst (2000) – 20-stage pipeline
Pentium Prescott (2004) – 31-stage pipeline
Core i7 9xx – Bloomfield (2008) – 24-stage pipeline
Core i7 2xx/3xx – Sandy Bridge (2013) – 19-stage pipeline

Incidence of Data Hazards: The more stages in the pipeline the further away a previous instruction can be while still maintaining a dependency

Impact of both Data and Control Hazards: The more stages in the pipeline, the more cycles we need to stall the pipeline when a hazard occurs·

Hardware costs: The more stages in the pipeline the more extra buffering is needed and the more extra wiring to communicate stages· Implementing forwarding is specially a concern as the number of forwarding paths needed increases nearly quadratically with the number of stages·

Clock limitation: The clock can not be made infinitely fast· Power dissipation increases with clock frequency, rapidly increasing the consumed energy and the temperature of the chip·

2. Consider a simple program with two nested loops as the following:

```
while (true) {
    for (i=0; i<x; i++) {
        do_stuff
    }
}
```

With the following assumptions:

`do_stuff` has 20 instructions that can be executed ideally in the pipeline.
The overhead for control hazards is 3-cycles, regardless of the branch being static or conditional.
The two loops can be translated into a single branch instruction each.

Calculate the instructions-per-cycle that can be achieved for different values of x (2, 4, 10, 100):

a) Without branch prediction.

outer loop: always branches → always fails :: 3-cycle penalty per outer loop iteration

inner loop: branches (x-1) out of x iterations → (x-1) fails :: 3(x-1) cycles penalty per outer loop iteration*

Instructions = x(20+1) + 1*

Cycles = Instructions + penalties = x(20+1) + 1 + 3*(x-1) + 3*

b) With a simple branch prediction policy - do the same as the last time.

1st loop: always branches → always predicts (except first time) ::: no penalty

2nd loop: first (predicts no branch, but branches) and last (predicts branch but doesn't) iterations always fail, rest always predict (predict branch AND branches) ::: two 3-cycle penalties per outer iteration

Instructions = x(20+1) + 1*

Cycles = x(20+1) + 1 + 3*(2)*

c) With perfect branch prediction - always predicts correctly.

1st loop: always predicts ::: no penalty

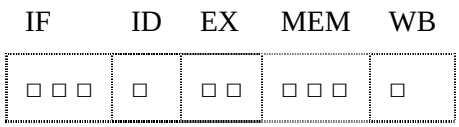
2nd loop: always predicts ::: no penalty

Instructions = x(20+1) + 1*

Cycles = x(20+1) + 1*

	x = 2	x = 4	x = 10	x = 100
Instructions (per 1st loop iteration)	43	85	211	2101
a) Cycles - w/o BP	49	97	241	2401
a) IPC - w/o BP	0·878	0·876	0·876	0·875
b) Cycles - simple BP	49	91	217	2107
b) IPC - simple BP	0·878	0·934	0·972	0·997
c) Cycles - perfect BP	43	85	211	2101
c) IPC - perfect BP	1·000	1·000	1·000	1·000

3. Consider a 10-stage fully pipelined processor as the one below.



(IF and MEM: 3 stages each; EX: 2 stages; ID and WB: 1 stage each)

a) How many cycle penalties will be incurred by the different kinds of Hazards in such processor?

Data Hazards:

Up to 5 cycle penalty (1EX+3MEM+1WB) for 2 consecutive instructions with data dependencies if forwarding is not implemented. 1st instruction needs to WB before the 2nd advances to EX (see below)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
add r1,r0,r3	IF	IF	IF	ID	EX	**EX**	**MEM**	**MEM**	**MEM**	**WB**						
sub r1,r0,r3		IF	IF	IF	ID	stall	stall	stall	stall	stall	EX	EX	MEM	MEM	MEM	WB

Control Hazards:

Unconditional branches: 3 cycles (2IF + 1ID). We don't know it's a branch until ID stage (3 instructions already fetched)

	1	2	3	4	5	6	7
b n	IF	IF	IF	**ID**	EX	EX	MEM
inst 1		IF	IF	IF	ID	EX	EX
inst 2			IF	IF	IF	ID	EX
inst 3				IF	IF	IF	ID
n					IF	IF	IF

Conditional branches: 5 cycles (2 IF + 1 ID + 2 EX). We don't know whether we need to branch or not until end of EX stage (5 instructions already fetched)

	1	2	3	4	5	6	7
Beq n	IF	IF	IF	ID	EX	**EX**	MEM
inst 1		IF	IF	IF	ID	EX	EX
inst 2			IF	IF	IF	ID	EX
inst 3				IF	IF	IF	ID
inst 4					IF	IF	IF
inst 5						IF	IF
n							IF

4. Consider the following program which implements $R = A^2 + B^2 + C^2 + D^2$

```
LD r1 @A
MUL r2 r1 r1            -- A^2
LD r3 @B
MUL r4 r3 r3            -- B^2
ADD r11 r2 r4           -- A^2 + B^2
LD r5 @C
MUL r6 r5 r5            -- C^2
LD r7 @D
MUL r8 r7 r7            -- D^2
ADD r12 r6 r8           -- C^2 + D^2
ADD r21 r11 r12         -- A^2 + B^2 + C^2 + D^2
ST r21 @R
```

a) Simulate its execution in a basic 5-stage pipeline without forwarding.

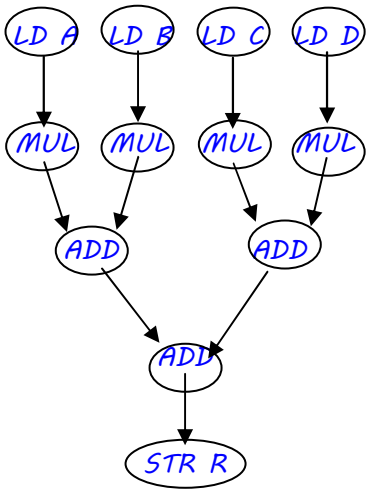
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
LD r1, A	IF	ID	EX	MEM	WB																											
MUL r2 r1 r1		IF	ID	Stall	Stall	EX	MEM	WB																								
LD r3, B			IF	Stall	Stall	ID	EX	MEM	WB																							
MUL r4 r3 r3						IF	ID	Stall	Stall	EX	MEM	WB																				
ADD r11 r2 r4							IF	Stall	Stall	ID	Stall	Stall	EX	MEM	WB																
LD r5, C										IF	Stall	Stall	ID	EX	MEM	WB															
MUL r6 r5 r5													IF	ID	Stall		EX	MEM	WB												
LD r7, D														IF	Stall		ID	EX	MEM	WB											
MUL r8 r7 r7																	IF	ID	Stall	Stall	EX	MEM	WB								
ADD r12 r6 r8																		IF	Stall	Stall	ID	Stall	Stall	EX	MEM	WB					
ADD r21 r11 r12																					IF	Stall	Stall	ID	Stall	Stall	EX	MEM	WB		
ST r21, R																								IF	Stall	Stall	ID	Stall	Stall	EX	MEM	WB

b) Simulate the execution of the original code in a 5-stage pipeline with forwarding.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LD r1, A	IF	ID	EX	MEM	WB															
MUL r2 r1 r1		IF	ID	Stall	EX	MEM	WB													
LD r3, B			IF	Stall	ID	EX	MEM	WB												
MUL r4 r3 r3					IF	ID	Stall	EX	MEM	WB										
ADD r11 r2 r4						IF	Stall	ID	EX	MEM	WB									
LD r5, C								IF	ID	EX	MEM	WB								
MUL r6 r5 r5									IF	ID	Stall	EX	MEM	WB						
LD r7, D										IF	Stall	ID	EX	MEM	WB					
MUL r8 r7 r7												IF	ID	Stall	EX	MEM	WB			
ADD r12 r6 r8													IF	Stall	ID	EX	MEM	WB		
ADD r21 r11 r12															IF	ID	EX	MEM	WB	
ST r21, R																IF	ID	EX	MEM	WB

Note: Colored stages represents dependencies between instructions (e.g. cyan in MUL r2,r1,r1 EX has a dependency on LDR, r1,A MEM

c) Draw the dependency graph of the application.



d) Based on the dependency graph discuss the suitability of the code to be run in a 2-way superscalar

Looking at the dependency graph we can see that there are many independent instructions that could be issued in parallel to exploit instruction level parallelism. Only the last two instructions (ADD and STR) would not be able to be issued in parallel.

e) Simulate the execution of the original code in a 5-stage 2-way superscalar pipeline with forwarding.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD r1, A	IF	ID	EX	MEM	WB												
MUL r2 r1 r1		IF	ID	Stall	EX	MEM	WB										
LD r3, B		IF	Stall	ID	EX	MEM	WB										
MUL r4 r3 r3				IF	ID	Stall	EX	MEM	WB								
ADD r11 r2 r4					IF	Stall	ID	EX	MEM	WB							
LD r5, C					IF	Stall	ID	EX	MEM	WB							
MUL r6 r5 r5							IF	ID	Stall	EX	MEM	WB					
LD r7, D							IF	Stall	ID	EX	MEM	WB					
MUL r8 r7 r7									IF	ID	Stall	EX	MEM	WB			
ADD r12 r6 r8										IF	Stall	ID	EX	MEM	WB		
ADD r21 r11 r12												IF	ID	EX	MEM	WB	
ST r21, R													IF	ID	EX	MEM	WB

f) Simulate the execution of the reordered code in a 5-stage 2-way superscalar pipeline

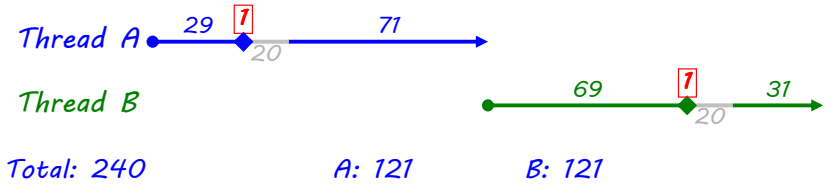
	1	2	3	4	5	6	7	8	9	10	11
1 LD r1, A	IF	ID	EX	MEM	WB						
2 LD r3, B	IF	ID	EX	MEM	WB						
3 LD r5, C		IF	ID	EX	MEM	WB					
4 LD r7, D		IF	ID	EX	MEM	WB					
5 MUL r2 r1 r1			IF	ID	EX	MEM	WB				
6 MUL r4 r3 r3			IF	ID	EX	MEM	WB				
7 MUL r6 r5 r5				IF	ID	EX	MEM	WB			
8 MUL r8 r7 r7				IF	ID	EX	MEM	WB			
9 ADD r11 r2 r4					IF	ID	EX	MEM	WB		
10 ADD r12 r6 r8					IF	ID	EX	MEM	WB		
11 ADD r21 r11 r12						IF	ID	EX	MEM	WB	
12 ST r21, R							IF	ID	EX	MEM	WB

5. Consider we want to execute 2 programs with 100 instructions each. The first program suffers an i-cache miss at instruction #30, and the second program another at instruction #70. Assume that

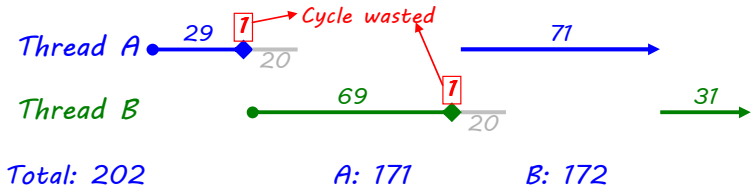
+ There is parallelism enough to execute all instructions independently (no hazards)
+ Switching threads can be done instantaneously
+ A cache miss requires 20 cycles to get the instruction to the cache.
+ The two programs would not interfere with each other's caches lines

Calculate the execution time observed by each of the programs (cycles elapsed between the execution of the first and the last instruction of that application) and the total time to execute the workload

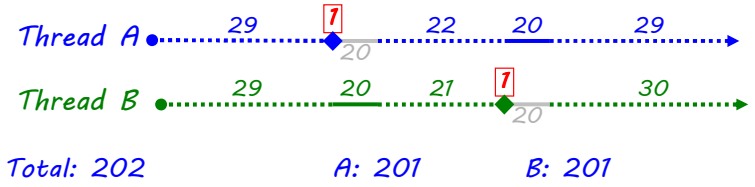
a) Sequentially (no multithreading),



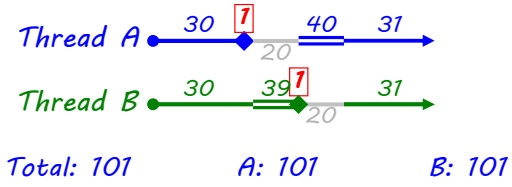
b) With coarse-grain multithreading,



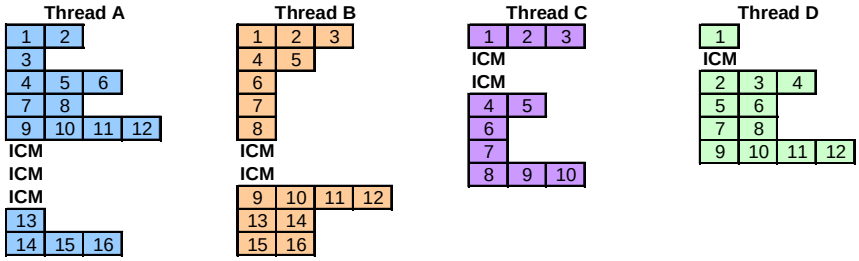
c) With fine-grain multithreading,



d) And with 2-way simultaneous multithreading.

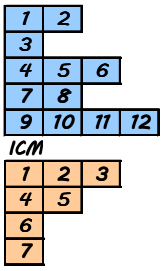


6. Consider a 4-way superscalar processor with 4-way multithreading and the 4 threads, with the instructions issued as per the diagrams below. ICM stands for instruction cache miss and means no instruction can be issued that cycle.



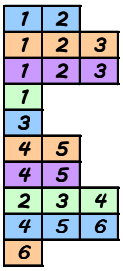
Draw a diagram showing the execution flow (issuing only) of the threads of a multithreaded processor (first 10 cycles only).
Assume the processor starts issuing instructions from the different threads in order A >> B >> C >> D >> A:

a) Coarse-grain multithreading



Swap thread upon 'expensive' operations (ICM)

b) Fine-grain multithreading



Round robin over 'ready' threads

c) Simultaneous multithreading

Issue instructions from several threads per cycle

7. Consider a modern 6-core processor with 2-way multithreading and 4-way superscalar pipelines.

a) What is this processor's peak IPC?

6 cores, each of them with a 4-way superscalar can issue/commit up to 24 instructions in a single cycle.

b) How many concurrent threads are supported by this processor?

6 cores running 2 threads each: 12 in total

c) Assume the processor has a MESI cache coherency protocol with copy back. For the following sequence of accesses to variable 'x', show the bus transactions, the cache states and the actions in main memory. Assume all the corresponding cache lines start in the 'I' state.

core0: LDR r0, x

Read miss: Cache0 sends a MEM_READ. No other cache has it, so main memory sends the value. Cache0 changes to 'E'

core1: LDR r1, x

Read miss: Cache1 sends a MEM_READ. Cache0 has the cache line, so sends the value. Cache0 and Cache1 change to 'S'

core2: STR r0, x

Write miss: Cache2 sends a RWITW. Cache0 and Cache1 change to 'I', main memory sends the value and Cache2 changes to 'M'

core3: LDR r3, x

Read miss: Cache3 sends a MEM_READ. Cache2 has the cache line, so sends the value to Cache3 and updates main memory. Cache3 and Cache2 change to 'S'

core3: STR r5, x

Write hit: Cache3 sends an INVALIDATE. Cache2 changes to 'I', Cache3 changes to 'M'

8. Consider the following 3 processors:

	Proc 1	Proc 2	Proc 3
# of cores	8	6	8
Multithreading	No	4-way fine-grain	2-way SMT
Superscalar	4-way	4-way	2-way
Clock freq.	1.5 GHz	2.5 GHz	3GHz

For each of them compute:

a) single-thread peak performance

For single-thread performance we need to multiply the peak IPC (superscalar lanes) by the clock frequency

*Proc 1: 4*1·5 = 6Ginstr/s*

*Proc 2: 4*2·5 = 10Ginstr/s*

*Proc 3: 2*3 = 6Ginstr/s*

b) peak IPC of the system (instructions per cycle)

For peak IPC, we need to multiply the number of cores times the maximum number of instructions that can be executed per cycle·

*Proc 1: 8*4 = 32 instr/cycle*

*Proc 2: 6*4 = 24 instr/cycle*

*Proc 3: 8*2 = 16 instr/cycle*

c) peak computing throughput of the system (instructions per second)

For the computing throughput we need to multiply the number of cores, the IPC and the freq·

*Proc 1: 8*4*1·5 = 48Ginstr/s*

*Proc 2: 6*4*2·5 = 60Ginstr/s*

*Proc 3: 8*2*3 = 48 Ginstr/s*

d) the total number of hardware threads supported by the system

For the number of threads, we multiply the number of cores times the threads per core

*Proc 1: 8*1 = 8 threads*

*Proc 2: 6*4 = 24 threads*

*Proc 3: 8*2 = 16 threads*