CMPEN 331 – Computer Organization and Design,
Exam 2 Review Questions

1. Compare two pipeline implementations options A and B with 4 and 7 stages, respectively.
   i. The logic delays of the pipeline stages are follows:

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
Option A	250 ps	180 ps	400 ps	200 ps			
Option B	200 ps	150 ps	250 ps	250 ps	200 ps	150 ps	180 ps

What are the maximum clock rates for the two implementations?

Option A fs_max = _____ (include the unit with your result)
Option B fs_max = _____ (include the unit with your result)

   ii. The table below states the operation of each pipeline stage:

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
Option A	IF/ID	EXE	MEM	WB			
Option B	IF	ID	EXE-1	EXE-2	EXE-3	MEM	WB

Compared to the MIPS CPU, option A merges IF and ID in a single stage, while option B splits EXE over three pipeline stages. Registers and memory are written to in the first half of the cycle and read during the second half of the cycle (same as MIPS) but there are no forwarding paths. How many instructions are executed after an add and a lw instruction, respectively, before the new register values are available?

	# nops after add	# nops after lw
Option A		
Option B		

   iii. Calculate the number of instructions executed per second for each implementation for the specified fs (these are not necessarily the correct results for part 1) and CPI.

	f_s	CPI	Instructions per second
Option A	3 GHz	1.5	
Option B	5 GHz	2.0	

2. Consider a processor with the following specification:

- Standard five (5) stage (F, D, E, M, W) pipeline. o No forwarding.

- Stalls on ALL hazards.
- Non-delayed branches
- Branch comparison occurs during the second stage.
- Instructions are not fetched until branch comparison is done.
- Memory CAN be read/written on same clock cycle.
- The same register CAN be read & written on the same clock cycle (structural hazard).
- No out-of-order execution.

i. Count how many cycles will be needed to execute the code below and write out each instruction's progress through the pipeline by filling in the table below with pipeline stages (F, D, E, M, W).

ii.

```
[1] add $a0, $a0, $t1
[2] lw $a1, 0($a0)
[3] add $a1, $a1, $t1
[4] sw $a1, 0($t1)
[5] add $t1, $t1, -1
[6] bne $0, $0, end
[7] add $t9, $t9, 1
```

Cycle →	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
Inst 1	**F**																								
Inst 2																									
Inst 3																									
Inst 4																									
Inst 5																									
Inst 6																									
Inst 7																									

iii. Considering the following two changes, fill in the table again:

- Our processor now forwards values
- Delayed branches

Cycle→	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
Inst 1	**F**																								
Inst 2																									
Inst 3																									
Inst 4																									
Inst 5																									
Inst 6																									
Inst 7																									

3. Assume that we have a standard 5-stage pipelined CPU with no forwarding. Register file writes can happen before reads, in the same clock cycle. We also have comparator logic that begins at the beginning of the decode stage and calculates the next PC by the end of the decode stage. For now, assume there is no branch delay slot. The remainder of the questions pertains to the following piece of MIPS code:

	Instructions	Cycle									
		1	2	3	4	5	6	7	8	9	10
0	`start: addu $t0 $t1 $t4`	IF	D	EX	MEM	WB					
1	`addiu $t2 $t0 0`		IF	D	EX	MEM	WB				
2	`ori $t3 $t2 0xDEAD`			IF	D	EX	MEM	WB			
3	`beq $t2 $t3 label`				IF	D	EX	MEM	WB		
4	`addiu $t2 $t3 6`					IF	D	EX	MEM	WB	
5	`label: addiu $v0 $0 10`						IF	D	EX	MEM	WB
6	`syscall`										

iv. For each instruction dependency below (the line numbers are given), list the type of hazard and the length of the stall needed to resolve the hazard. If there is no hazard, write "no hazard".

```
0 → 1: addu  $t0 $t1 $t4 → addiu $t2 $t0
0 → 3: addu  $t0 $t1 $t4 → beq  $t2 $t3 label
1 → 3: addiu $t2 $t0 0 → beq $t2 $t3 label
2 → 3: ori $t3 $t2 0xDEAD → beq $t2 $t3 label
3 → 4: beq $t2 $t3 label → addiu $t2 $t3 6
```

For the following questions, assume that our CPU now has forwarding implemented as presented in class and in the book.

v. Which of these instruction dependencies would cause a pipelining hazard?

```
A. ori $t3 $t2 0xDEAD → beq $t2 $t3 label
B. ori $t3 $t2 0xDEAD → addiu $t2 $t3 6
C. ori $t3 $t2 0xDEAD → addiu $v0 $0 10
D. beq $t2 $t3 label → addiu $t2 $t3 6
E. None of the above
```

vi. If we are given a **branch delay slot**, which instruction would reduce the most amount of pipelining hazards if moved into the branch delay slot? If all instructions are equally beneficial, or no instruction removes any hazards, write "nop" as your answer.

4. The figure below illustrates three possible predictors.

- Last taken predicts taken when 1
- Up-Down (saturating counter) predicts taken when 11 and 10

Fill out the tables below for each branch predictor. The execution pattern for the branch is NTNNTTTN.

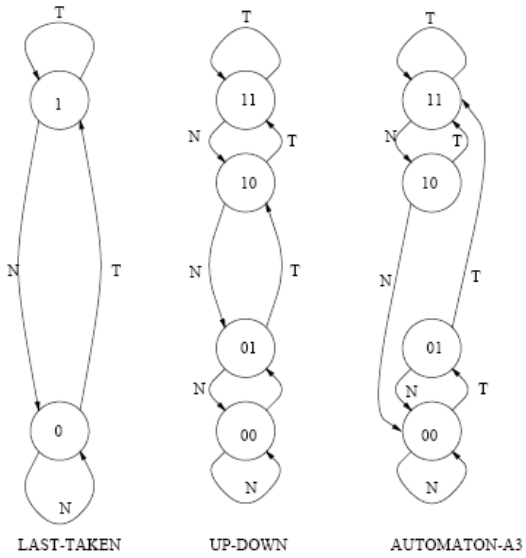

Table 1: Table for last-taken branch predictor

Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	0			
1	T				
2	N				
3	N				
4	T				
5	T				
6	T				
7	N				

Table 2: Table for saturating counter (up-down) branch predictor.

Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	01			
1	T				
2	N				
3	N				
4	T				
5	T				
6	T				
7	N				

Table 3: Table for Automata-A3 branch predictor.

Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	01			
1	T				
2	N				
3	N				
4	T				
5	T				
6	T				
7	N				

Solution

1. Compare two pipeline implementations options A and B with 4 and 7 stages, respectively.
   iv. The logic delays of the pipeline stages are follows:

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
Option A	250 ps	180 ps	400 ps	200 ps			
Option B	200 ps	150 ps	250 ps	250 ps	200 ps	150 ps	180 ps

What are the maximum clock rates for the two implementations?

Option A fs_max = _____1/0.4ns = 2.5GHz_____ (include the unit with your result)
Option B fs_max = _____1/0.25ns = 4GHz_____ (include the unit with your result)

v. The table below states the operation of each pipeline stage:

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
Option A	IF/ID	EXE	MEM	WB			
Option B	IF	ID	EXE-1	EXE-2	EXE-3	MEM	WB

Compared to the MIPS CPU, option A merges IF and ID in a single stage, while option B splits EXE over three pipeline stages. Registers and memory are written to in the first half of the cycle and read during the second half of the cycle (same as MIPS) but there are no forwarding paths. How many instructions are executed after an add and a lw instruction, respectively, before the new register values are available?

Solution

	# nops after add	# nops after lw
Option A	2	2
Option B	4	4

vi. Calculate the number of instructions executed per second for each implementation for the specified fs (these are not necessarily the correct results for part 1) and CPI.

Solution

	f_s	CPI	Instructions per second
Option A	3 GHz	1.5	1/CPI * Cycles / sec $(1/1.5)(1/((1/3)*10^{-9}))= 2*10^9$
Option B	5 GHz	2.0	$(1/2.0)(1/((1/5)*10^{-9})) = 2.5*10^9$

2. Consider a processor with the following specification:

- Standard five (5) stage (F, D, E, M, W) pipeline. o No forwarding.
- Stalls on ALL hazards.
- Non-delayed branches
- Branch comparison occurs during the second stage.
- Instructions are not fetched until branch comparison is done.
- Memory CAN be read/written on same clock cycle.
- The same register CAN be read & written on the same clock cycle (structural hazard).
- No out-of-order execution.

vii. Count how many cycles will be needed to execute the code below and write out each instruction's progress through the pipeline by filling in the table below with pipeline stages (F, D, E, M, W).

viii.

```
[1] add $a0, $a0, $t1
[2] lw $a1, 0($a0)
[3] add $a1, $a1, $t1
[4] sw $a1, 0($t1)
[5] add $t1, $t1, -1
[6] bne $0, $0, end
[7] add $t9, $t9, 1
```

Cycle →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	2
Inst 1	**F**																								
Inst 2																									
Inst 3																									
Inst 4																									
Inst 5																									
Inst 6																									
Inst 7																									

ix. Considering the following two changes, fill in the table again:

- Our processor now forwards values
- Delayed branches

Cycle→	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Inst 1	**F**																							
Inst 2																								
Inst 3																								
Inst 4																								
Inst 5																								

Inst 6																							
Inst 7																							

i)Here's the chart

1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
1	F	D	E	M	W0														Start @ 1
2		F	D	D	D0	E	M	W1											Stall for $a0 to be writte before it can be read (nc forwarding)
3			F					D1	E	M	W1								Stall for $a1 to be writte before it can be read (nc forwarding)
4								F			D1	E	M	W					Stall for $a1 to be writte before it can be read
5											F	D	E	M	W1				Stall until the above inst finishes before I can finish (no out-of-order exec!)
6												F	D	E	M	W			No hazards; proceed as normal
7														F	D	E	M	W	Stall because we have non-delayed branches but we don't know which instr to take until after the 2nd stage

ii) Here's the chart with the addition of forwarding and delayed branches
The delayed branch means that the instruction following the branch is always executed before the PC is modified to perform the branch

1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
1	F	D	E	M	W0														Start @ 1
2		F	D	E	M1	W													$a0 is forwarded from ALU->ALU
3			F		D1	E	M	W											Standard load delay stall because we need to wait for the data fror memory (to eventually be stored in $a1), which is forwarded to the ALU.
4					F	D1	E	M	W										The value of $a1 is forwarded to the memory's "data in" line
5						F	D	E	M	W									No hazards, proceed as normal
6							F	D	E	M	W								No hazards, proceed as normal
7								F	D	E	M	W							Branch-delay slot is filled, no need to stall

3. Assume that we have a standard 5-stage pipelined CPU with no forwarding. Register file writes can happen before reads, in the same clock cycle. We also have comparator logic that begins at the beginning of the decode stage and calculates the next PC by the end of the decode stage. For now, assume there is no branch delay slot. The remainder of the questions pertains to the following piece of MIPS code:

	Instructions	Cycle									
		1	2	3	4	5	6	7	8	9	10
0	**start: addu $t0 $t1 $t4**	**IF**	**D**	**EX**	**MEM**	**WB**					
1	**addiu $t2 $t0 0**		**IF**	**D**	**EX**	**MEM**	**WB**				
2	**ori $t3 $t2 0xDEAD**			**IF**	**D**	**EX**	**MEM**	**WB**			
3	**beq $t2 $t3 label**				**IF**	**D**	**EX**	**MEM**	**WB**		
4	**addiu $t2 $t3 6**					**IF**	**D**	**EX**	**MEM**	**WB**	
5	**label: addiu $v0 $0 10**						**IF**	**D**	**EX**	**MEM**	**WB**
6	**syscall**										

x. For each instruction dependency below (the line numbers are given), list the type of hazard and the length of the stall needed to resolve the hazard. If there is no hazard, write "no hazard".

```
0 → 1: addu  $t0 $t1 $t4 → addiu $t2 $t0
0 → 3: addu  $t0 $t1 $t4 → beq  $t2 $t3 label
1 → 3: addiu $t2 $t0 0 → beq $t2 $t3 label
2 → 3: ori $t3 $t2 0xDEAD → beq $t2 $t3 label
3 → 4: beq $t2 $t3 label → addiu $t2 $t3 6
```

0 → 1: data hazard, 2 cycles
0 → 3: no hazard
1. → **3: data hazard, 1 cycle**
2. → **3: data hazard, 2 cycles**
3. → **4: control hazard, 1 cycle**

For the following questions, assume that our CPU now has forwarding implemented as presented in class and in the book.

xi. Which of these instruction dependencies would cause a pipelining hazard?

A. `ori $t3 $t2 0xDEAD → beq $t2 $t3 label`

B. `ori $t3 $t2 0xDEAD → addiu $t2 $t3 6`

C. `ori $t3 $t2 0xDEAD → addiu $v0 $0 10`

D. `beq $t2 $t3 label → addiu $t2 $t3 6`

E. None of the above

A, D

xii. If we are given a **branch delay slot**, which instruction would reduce the most amount of pipelining hazards if moved into the branch delay slot? If all instructions are equally beneficial, or no instruction removes any hazards, write "nop" as your answer.

addiu $v0 $0 10

4. The figure below illustrates three possible predictors.
   - Last taken predicts taken when 1
   - Up-Down (saturating counter) predicts taken when 11 and 10

Fill out the tables below for each branch predictor. The execution pattern for the branch is NTNNTTTN.

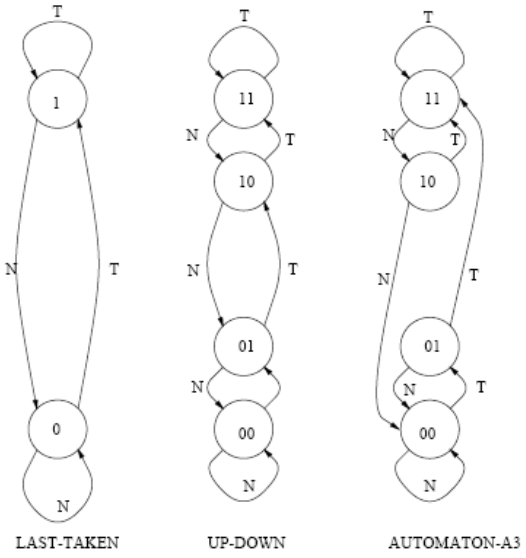

Table 1: Table for last-taken branch predictor

Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	0	NT	Correct	0
1	T	0	NT	Incorrect	1
2	N	1	T	Incorrect	0
3	N	0	NT	Correct	0
4	T	0	NT	Incorrect	1
5	T	1	T	Correct	1
6	T	1	T	Correct	1
7	N	1	T	Incorrect	0

Table 2: Table for saturating counter (up-down) branch predictor.

Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	01	NT	Correct	00
1	T	00	NT	Incorrect	01
2	N	01	NT	Correct	00
3	N	00	NT	Correct	00
4	T	00	NT	Incorrect	01
5	T	01	NT	Incorrect	10
6	T	10	T	Correct	11
7	N	11	T	Incorrect	10

Table 3: Table for Automata-A3 branch predictor.

Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	01	NT	Correct	00
1	T	00	NT	Incorrect	01
2	N	01	NT	Correct	00

3	N	00	NT	Correct	00
4	T	00	NT	Incorrect	01
5	T	01	NT	Incorrect	11
6	T	11	T	Correct	11
7	N	11	T	Incorrect	10