Problem M3.5: Fetch Pipelines [? Hours]

PC	PC Generation
F1	ICache Access
F2	
D1	Instruction Decode
D2	
RN	Rename/Reorder
RF	Register File Read
EX	Integer Execute

Problem M3.5.A — Pipelining Subroutine Returns

Immediately after what pipeline stage does the processor know that it is executing a subroutine return instruction?
D2

Immediately after what pipeline stage does the processor know the subroutine return address?
RF

How many pipeline bubbles are required when executing a subroutine return?
6

Problem M3.5.B — Adding a BTB

A subroutine can be called from many different locations and thus a single subroutine return can return to different locations. A BTB holds only the address of the last caller.

Problem M3.5.C — Adding a Return Stack

Normally, instruction fetch needs to wait until the return instruction finishes the RF stage before the return address is known. With the return stack, as soon as the return instruction is decoded in D2, instruction fetch can begin fetching from the return address. This saves 2 cycles.

A return address is pushed after a JAL/JALR instruction is decoded in D2. A return address is popped after a JR r31 instruction is decoded in D2.

Problem M3.5.D Return Stack Operation

```
A: JAL B
A+1:
A+2:
...

B: JR r31
B+1:
B+2:
...
```

instruction					time→													
A	PC	F1	F2	D1	D2	RN	RF	EX										
A+1		PC	F1	F2	D1	~~D2~~	~~RN~~	~~RF~~	~~EX~~									
A+2			PC	F1	F2	~~D1~~	~~D2~~	~~RN~~	~~RF~~	~~EX~~								
A+3				PC	F1	~~F2~~	~~D1~~	~~D2~~	~~RN~~	~~RF~~	~~EX~~							
A+4					PC	~~F1~~	~~F2~~	~~D1~~	~~D2~~	~~RN~~	~~RF~~	~~EX~~						
B						PC	F1	F2	D1	D2	RN	RF	EX					
B+1							PC	F1	F2	D1	~~D2~~	~~RN~~	~~RF~~	~~EX~~				
B+2								PC	F1	F2	~~D1~~	~~D2~~	~~RN~~	~~RF~~	~~EX~~			
B+3									PC	F1	~~F2~~	~~D1~~	~~D2~~	~~RN~~	~~RF~~	~~EX~~		
B+4										PC	~~F1~~	~~F2~~	~~D1~~	~~D2~~	~~RN~~	~~RF~~	~~EX~~	
A+1											PC	F1	F2	D1	D2	RN	RF	EX

Problem M3.5.E Handling Return Address Mispredicts

When a value is popped off the return stack after D2, it is saved for two cycles as part of the pipeline state. After the RF stage of the return instruction, the actual r31 is compared against the predicted return address. If the addresses match, then we are done. Otherwise we mux in the correct program counter at the PC stage and kill the instructions in F1 and F2. Depending on how fast the address comparison is assumed to be, you might also kill the instruction in D1. So there is an additional 2 or 3 cycles lost on a return mispredict.

Problem M3.5.F Further Improving Performance

Ben should add a cache of the most recently encountered return instruction addresses. During F1, the contents of the cache are looked up to see if any entries match the current program counter. If so, then by the end of F1 (instead of D2) we know that we have a return instruction. We can then use the return stack to supply the return address.

Problem M3.6: Managing Out-of-order Execution

Problem M3.6.A

Rename Table

R1	~~P1~~	~~P4~~	**P7**
R2	~~P2~~	~~P5~~	**P8**
R3	~~P3~~	P6	
R4	P0		

Physical Regs

P0	8016	p
P1	~~6823~~	~~p~~
P2	~~8000~~	~~p~~
P3	7	p
P4	**0**	**p**
P5	**8004**	**p**
P6	**8**	**p**
P7		
P8		
P9		

Free List

~~P4~~
~~P5~~
~~P6~~
~~P7~~
~~P8~~
P9
P1
P2
⋮

Reorder Buffer (ROB)

	use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
next to commit	~~x~~	**x**	lw	p	P2			r1	P1	P4
	~~x~~	**x**	addi	p	P2			r2	P2	P5
	x		beqz	**p**	P4					
	x	**x**	addi	p	P3			r3	P3	P6
next available	x		bne	**p**	P5	p	P0			
	x		**lw**	**p**	**P5**			**r1**	**P4**	**P7**
	x		**addi**	**p**	**P5**			**r2**	**P5**	**P8**
	x		**beqz**		**P7**					

Problem M3.6.B

Rename Table

R1	**P4**
R2	**P5**
R3	**P3**
R4	**P0**

Physical Regs

P0	**8016**	**p**
P1		
P2		
P3	**7**	**p**
P4	**0**	**p**
P5	**8004**	**p**
P6		
P7		
P8		
P9		

Free List

P9
P1
P2
P6
P7
P8
⋮

Reorder Buffer (ROB)

	use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
next to commit	**x**		**bne**	**p**	**P5**	**p**	**P0**			
next available										

Problem M3.6.C

Under what conditions, if any, might the loop execute at a faster rate on the in-order processor compared to the out-of-order processor?

If the out-of-order processor frequently mispredicts either of the branches, it is likely to execute the loop slower than the in-order processor. For this to be true, we must also assume that the branch misprediction penalty of the out-of-order processor is sufficiently longer than the branch resolution delay of the in-order processor, as is likely to be the case. The mispredictions may be due to deficiencies in the out-of-order processor's branch predictor, or the data-dependent branch may be fundamentally unpredictable in nature.

Under what conditions, if any, might the loop execute at a faster rate on the out-of-order processor compared to the in-order processor?

If the out-of-order processor predicts the branches with high enough accuracy, it can execute more than one instruction per cycle, and thereby execute the loop at a faster rate than the in-order processor.

Problem M3.7: Exceptions and Register Renaming

Problem M3.7.A Recovering from Exceptions

By the definition of a precise exception, an exception that occurs in the middle of an x86 instruction should cause the machine state to revert to the state that previously existed right before the excepting instruction started executing. Thus a strategy to determine a precise state would be to take snapshots of the RAT only on x86 instruction boundaries (either when the last µop of an x86 instruction commits or right before the first µop of an x86 instruction is renamed).

Problem M3.7.B Minimizing Snapshots

Ben is correct. Since an exception causes the machine to revert to the state found on an x86 instruction boundary, all the temporary state used by the µops does not need to be kept. Thus, the RAT only has to hold the rename mappings for the architectural registers, and not for T0-T7.

Problem M3.7.C Renaming Registers

There must be at least 17 physical registers for the Bentium 4 to work properly. 16 registers are needed to hold the state of the machine at any given point in time (architectural and temporary register values), and an extra one is needed to rename an additional register using the given renaming algorithm to allow forward progress.