Very Short Answer:

(1) (2) What is the relationship between speculation and power consumption?

(2) (2) If you are mainly worried about program size, what type of instruction set would you use? What if performance was your primary concern?

(3) (1) Which is more effective, dynamic or static branch prediction?

(4) (1) Do benchmarks remain valid indefinitely?

(5) (2) Issuing multiple instructions per cycle puts tremendous pressure on what two parts of the machine?

(6) (2) In class we mentioned VLIW and Superscalar as two ways to circumvent the Flynn Limit of 1. We also talked about two other approaches - what were they?

(7) (2) Out of Order completion makes supporting what very difficult?

(8) (2) Are wire delays or transistors more likely to be the most significant limit on clock frequency in the future? Why?

(9) (2) What does SMT stand for?

(10) (2) Decoupled architectures split a program into two streams. What are they?

(11) (2) What is Amdahl's law (in words)?

(12) (2) What is the primary difference between Scoreboarding and Tomasulo's algorithm?

Short Answers:

(10) (4) Why are there multiple dies per silicon wafer? Why not just fabricate one huge die per wafer?

(11) (3) Write down the 3-term CPU performance equation developed in class.

(12) (3) The book lists several things that limit the amount of achievable ILP. List 3 of them.

(13) (4)What is a predicated instruction? What are the advantages to using predicated instructions? When would you *not* want to use one?

(14) (4) What is the definition of a basic block? Why is there a desire to create larger ones?

(15) (3) There are at least two types of control flow changes that standard dynamic branch predictors have trouble with. There is a technique that works well for one of these types ... name the two types of branches, and the technique used to successfully deal with one of them.

(16) (4) Supporting precise interrupts in machines that allow out of order completion is a challenge. Briefly explain why, and give three different techniques that can be used to provide precise interrupts.

(17) (5) Why is branch prediction important? What performance enhancing techniques have made it so? List 3 examples of existing Branch Prediction strategies in order of (average) increasing effectiveness.

(18) (12) Draw a basic high-level picture of what tomasulo's hardware looks like, when the ROB is included. (In other words, sketch out all the hardware involved, and how things are connected.) The emphasis is on conveying knowledge - do not worry about how pretty it is, but do make sure I can read it and understand what you have done.

(10) The contents of the table represent the state of the machine at time 6. Your job is to fill in the table at time 7.

Time = 6

Instruction Status

Instruction		Issue Inst	Execute Underway	Execute Completes	Write Result
LF	F2,34(R3)	X	X	X	**Load2**
MULTF	F0,F2,F4	X			
SUBF	F8,F6,F2	X			
DIVF	F10,F0,F6	X			
ADDF	F6,F8,F2	**X**			

Reservation Stations

Tag	Name	Busy	Fm(op)	Val.j(Vj)	Val.k(Vk)	Tag.j(Qj)	Tag.k(Qk)
Add1	Add/Sub	Yes	Sub	**Mem[]**	0		Load2
Add2	Add/Sub	**Yes**	**Add**	0	0	**Add1**	**Load2**
Add3	Add/Sub						
Mult1	Mult/Div	Yes	Mult	0	Reg[F4]	Load2	
Mult2	Mult/Div	Yes	Div	0	**Mem[]**	Mult1	

Register Status

	F0	F2	F4	F6	F8	F10	12	...	F30
Tag.i(Qi)	Mult1	Load2		**Add2**	Add1	Mult2			

Time = 7

Instruction Status

Instruction		Issue Inst	Execute Underway	Execute Completes	Write Result
MULTF	F0,F2,F4				
SUBF	F8,F6,F2				
DIVF	F10,F0,F6				
ADDF	F6,F8,F2				

Reservation Stations

Tag	Name	Busy	Fm(op)	Val.j(Vj)	Val.k(Vk)	Tag.j(Qj)	Tag.k(Qk)
Add1	Add/Sub						
Add2	Add/Sub						
Add3	Add/Sub						
Mult1	Mult/Div						
Mult2	Mult/Div						

Register Status

	F0	F2	F4	F6	F8	F10	12	...	F30
Tag.i(Qi)									

(19) (14) You are given the following code sequence:

```
ADDF    F1,F1,F4
SUBF    F2,F2,F1
MULTF   F2,F2,F3
DIVF    F1,F3,F1
```

Assume there are 8 logical and 16 physical registers. On the left below is the register mapping upon entering the code sequence. Your job is to fill in the mappings after the execution of the DIVF instruction, including what is on the free list. (Assume that during the execution of this code, no registers are released - in other words, the free list will be shorter at the end than at the beginning.)

BEFORE	
Logical	Physical
0	3
1	5
2	6
3	8
4	11
5	12
6	1
7	7

Free Pool: 0,2,4,9,10,13,14,15

AFTER	
Logical	Physical
0	
1	
2	
3	
4	
5	
6	
7	

Free Pool:

Now, rewrite the code sequence below using the actual physical register names instead of the logical ones.

```
ADDF   P__,P__,P__
SUBF   P__,P__,P__
MULTF  P__,P__,P__
DIVF   P__,P__,P__
```

(20) (6) Suppose floating point instructions are responsible for 50% of the execution time of a particular benchmark, and the floating point square root (FPSQ) by itself is responsible for 23% of the execution time. You have a choice between speeding up the FPSQ instruction by a factor of 10, or making all floating point instructions run faster by a factor of 1.65. Which should you do? (Show how you set up the equations, although you do not need to actually solve them.)

(21) (6) In class, we talked about the cycle by cycle steps that occur on different interrupts. For example, here is what happens if there is an illegal operand interrupt generated by instruction i+1:

	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB	<- Interrupt detected		
i+2			IF	ID	EX	MEM	WB	<- Instruction Squashed	
i+3				IF	ID	EX	MEM	WB	<- Trap Handler fetched
i+4					IF	ID	EX	MEM	WB

Fill out the following table if instruction i+1 experiences a fault in the Mem stage (page fault, for example):

	1	2	3	4	5	6	7	8	9	10
i	IF	ID	EX	MEM	WB					
i+1		IF	ID	EX	MEM	WB				
i+2			IF	ID	EX	MEM	WB			
i+3				IF	ID	EX	MEM	WB		
i+4					IF	ID	EX	MEM	WB	
i+5						IF	ID	EX	MEM	WB

What happens in this case?

	1	2	3	4	5	6	7	8	9	10
i	IF	ID	EX	MEM	WB	<- Data write causes Page Fault				
i+1		IF	ID	EX	MEM	WB	<- Illegal Opcode			
i+2			IF	ID	EX	MEM	WB			
i+3				IF	ID	EX	MEM	WB		
i+4					IF	ID	EX	MEM	WB	
i+5						IF	ID	EX	MEM	WB