**Instructions**

**Format:** This exam consists of 6 multiple choice questions for each of the 12 lectures in the course. You may have to do some calculations in order to determine the correct answer. The motivation for having so many questions is to get a more accurate assessment of your knowledge of the material.

Be sure to circle the answers on the page titled “answers page” or your answers will not be graded. If you choose not to answer a question do not circle a letter. Feel free to detach the answers page from the remainder of the exam and only turn it in.

**Grading:** To discourage guessing, each incorrect answer will be worth \(-1/3\) point, while each correct answer is worth +1 point. I.e., if you guess randomly, the expected score is 0.

**Material:** You are allowed a calculator and one double-sided, hand-written A4 sheet of notes during the exam.

Good luck!
<table>
<thead>
<tr>
<th>Optional anonymous background questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1. A B C</td>
</tr>
<tr>
<td>0.2. A B C</td>
</tr>
<tr>
<td>0.3. A B C</td>
</tr>
<tr>
<td>1. ISA 1</td>
</tr>
<tr>
<td>1.1. A B C D</td>
</tr>
<tr>
<td>1.2. A B C D</td>
</tr>
<tr>
<td>1.3. A B C D</td>
</tr>
<tr>
<td>1.4. A B C D</td>
</tr>
<tr>
<td>1.5. A B C D</td>
</tr>
<tr>
<td>1.6. A B C D</td>
</tr>
<tr>
<td>2. ISA 2</td>
</tr>
<tr>
<td>2.1. A B C D</td>
</tr>
<tr>
<td>2.2. A B C D</td>
</tr>
<tr>
<td>2.3. A B C D</td>
</tr>
<tr>
<td>2.4. A B C D</td>
</tr>
<tr>
<td>2.5. A B C D</td>
</tr>
<tr>
<td>2.6. A B C D</td>
</tr>
<tr>
<td>3. Computer Arithmetic</td>
</tr>
<tr>
<td>3.1. A B C D</td>
</tr>
<tr>
<td>3.2. A B C D</td>
</tr>
<tr>
<td>3.3. A B C D</td>
</tr>
<tr>
<td>3.4. A B C D</td>
</tr>
<tr>
<td>3.5. A B C D</td>
</tr>
<tr>
<td>3.6. A B C D</td>
</tr>
<tr>
<td>4. Logic</td>
</tr>
<tr>
<td>4.1. A B C D</td>
</tr>
<tr>
<td>4.2. A B C D</td>
</tr>
<tr>
<td>4.3. A B C D</td>
</tr>
<tr>
<td>4.4. A B C D</td>
</tr>
<tr>
<td>4.5. A B C D</td>
</tr>
<tr>
<td>4.6. A B C D</td>
</tr>
<tr>
<td>5. Processor Control and Datapath</td>
</tr>
<tr>
<td>5.1. A B C D</td>
</tr>
<tr>
<td>5.2. A B C D</td>
</tr>
<tr>
<td>5.3. A B C D</td>
</tr>
<tr>
<td>5.4. A B C D</td>
</tr>
<tr>
<td>5.5. A B C D</td>
</tr>
<tr>
<td>5.6. A B C D</td>
</tr>
<tr>
<td>6. Pipelining</td>
</tr>
<tr>
<td>6.1. A B C D</td>
</tr>
<tr>
<td>6.2. A B C D</td>
</tr>
<tr>
<td>6.3. A B C D</td>
</tr>
<tr>
<td>6.4. A B C D</td>
</tr>
<tr>
<td>6.5. A B C D</td>
</tr>
<tr>
<td>6.6. A B C D</td>
</tr>
<tr>
<td>7. Hazards</td>
</tr>
<tr>
<td>7.1. A B C D</td>
</tr>
<tr>
<td>7.2. A B C D</td>
</tr>
<tr>
<td>7.3. A B C D</td>
</tr>
<tr>
<td>7.4. A B C D</td>
</tr>
<tr>
<td>7.5. A B C D</td>
</tr>
<tr>
<td>7.6. A B C D</td>
</tr>
<tr>
<td>8. Branch Prediction and Exceptions and Interrupts</td>
</tr>
<tr>
<td>8.1. A B C D</td>
</tr>
<tr>
<td>8.2. A B C D</td>
</tr>
<tr>
<td>8.3. A B C D</td>
</tr>
<tr>
<td>8.4. A B C D</td>
</tr>
<tr>
<td>8.5. A B C D</td>
</tr>
<tr>
<td>8.6. A B C D</td>
</tr>
<tr>
<td>9. Input/Output</td>
</tr>
<tr>
<td>9.1. A B C D</td>
</tr>
<tr>
<td>9.2. A B C D</td>
</tr>
<tr>
<td>9.3. A B C D</td>
</tr>
<tr>
<td>9.4. A B C D</td>
</tr>
<tr>
<td>9.5. A B C D</td>
</tr>
<tr>
<td>9.6. A B C D</td>
</tr>
<tr>
<td>10. Caches</td>
</tr>
<tr>
<td>10.1. A B C D</td>
</tr>
<tr>
<td>10.2. A B C D</td>
</tr>
<tr>
<td>10.3. A B C D</td>
</tr>
<tr>
<td>10.4. A B C D</td>
</tr>
<tr>
<td>10.5. A B C D</td>
</tr>
<tr>
<td>10.6. A B C D</td>
</tr>
<tr>
<td>11. Virtual Memory</td>
</tr>
<tr>
<td>11.1. A B C D</td>
</tr>
<tr>
<td>11.2. A B C D</td>
</tr>
<tr>
<td>11.3. A B C D</td>
</tr>
<tr>
<td>11.4. A B C D</td>
</tr>
<tr>
<td>11.5. A B C D</td>
</tr>
<tr>
<td>11.6. A B C D</td>
</tr>
<tr>
<td>12. Parallelism</td>
</tr>
<tr>
<td>12.1. A B C D</td>
</tr>
<tr>
<td>12.2. A B C D</td>
</tr>
<tr>
<td>12.3. A B C D</td>
</tr>
<tr>
<td>12.4. A B C D</td>
</tr>
<tr>
<td>12.5. A B C D</td>
</tr>
<tr>
<td>12.6. A B C D</td>
</tr>
</tbody>
</table>

Feel free to detach the remaining pages of the exam and turn in only your answers.
0. OPTIONAL anonymous background questions
(These are not graded and will not affect your grade on the exam. They are for me to be able to correlate what helps students prepare for the exam. Feel free to not answer these questions if you do not want to.)

1. How much of the book did you read?
   a. All
   b. Some
   c. None

2. How many of the practice sessions did you miss/turn in late?
   a. Several
   b. One
   c. None

3. How many of the previous years’ exams did you look at?
   a. All
   b. Some
   c. None

1. ISA 1

1. Which address would be an aligned address on a machine that only loads 64-bit words with a byte-addressable memory?
   a. 2
   b. 4
   c. 8
   d. 12
   
   64-bit words have 8 bytes, so for a byte-addressable memory you need to have the last 3 bits 0 to be aligned to an 8-byte boundary. 8=1000, but 12=1100.

2. How many control flow instructions are needed to implement a simple for-loop?
   a. 1
   b. 2
   c. 3
   d. Depends on whether you end the loop at 0
   
   1: one conditional branch can be used to jump back to the start of the loop until it is finished, and then you just continue past the branch when done. You need two for an if-then-else statement.

3. Why does the PC need information from the ALU?
   a. It uses the ALU to do PC+4
   b. It uses the ALU to load the instruction
   c. It uses the ALU to decide the next instruction
   d. All of the above
   
   The ALU is used to calculate whether conditional branches should be taken. The PC+4 and the instruction load are not done by the ALU.

4. How many fully usable general-purpose registers does MIPS have?
   a. 31
   b. 32
   c. 33
   d. More than 33
   
   31. There are 32 addressable registers, but R0 is not fully usable because you can’t write data to it. There are other registers (PC, floating point, multiplication, etc.) but they are not general purpose.
5. How many temporary registers are required to compute A-(B+C) if you need A, B, and C later?
   a. 0
   b. 1
   c. 2
   d. 3
   1. add t, B, C; sub t, A, t

6. From where do immediate instructions get their immediate values?
   a. The register file
   b. The instruction
   c. The ALU
   d. Depends on the instruction type
   The instruction. Immediate values are encoded in the instruction itself.

2. ISA 2
   1. Why do we only lose one register field for an I-format instruction vs. an R-format instruction?
      a. Immediate values only need as many bits as a single register field
      b. The immediate value replaces other fields and not the register fields
      c. The immediate value replaces some of the register fields and some other fields
      d. The immediate instructions have more bits
      Immediate instructions replace the rd, shmt, and funct fields. This adds up to 5+5+6=16 bits.
   2. What do we do to the immediate field for branch instructions to calculate the branch address and why?
      a. Shift to the left by 2 because instructions are word-aligned.
      b. Shift to the left by 4 because instructions are word-aligned.
      c. Add 4 because instructions are word-aligned.
      d. Sign-extend because the immediate is unsigned.
      We shift the address to the left by 2 (multiply by 4) because instructions are word-aligned. We do add 4, but that's just because we always do PC+4, and we do sign-extend, but it is because the immediate is two’s compliment.

3. How would you load the value 13 into the PC?
   a. addi $ra, $r0, 13
      jr $ra
   b. addi $pc, $r0, 13
   c. lui $pc, 0
      ori $pc, 13
   d. lui $pc, 13
   You can’t write directly into the PC, but the jr instruction will load the value from register $ra into the PC.

4. What immediate value should the conditional branch at address 12 have to jump to address 8?
   a. -1
   b. -2
   c. -4
   d. -8
   -2. The next instruction will be PC+4+(imm<<2). 8=12+4+(-2*4)

5. Why do we have a convention for callee and caller saving of registers?
   a. So we can call any function without overwriting our own registers or the function’s registers.
b. To avoid corrupting the stack.
c. To make it easier for the compiler to generate code.
d. All of the above.

The convention allows us to call other code without overwriting registers. Corrupting the stack simply requires incrementing and decrementing it correctly. The compiler would actually have an easier time if it didn’t have to follow these rules.

6. A function uses registers $s0$, $s1$ and $t3$. How much will it adjust the stack when it is called?
   a. -1
   b. -2
   c. -4
   d. -8

The callee needs to save $s1$ and $s0$ on the stack, which is two words, or 8 bytes. So the stack needs to move down by -8.

3. Computer Arithmetic

1. Why do we use two’s compliment notation?
   a. There is only one zero
   b. Subtraction is just invert addition with the carry in set to 1
   c. We can easily determine if the number is negative by looking at the MSB
d. All of the above

   All of the above.

2. How many bits are required to compute the following in two’s compliment without an overflow? -8+8=0
   a. 4
   b. 5
   c. 6
   d. 7

To hold the values 8 and -8 we need 4 bits for 8 and 5 bits for -8. Since the result is 0, we don’t need more bits for the result. Therefore we need 5 bits and get: -8+8=11000+01000 = 00000.

3. What number format gives you the largest range for a 3-bit fixed-point number?
   a. xxx.
   b. xx.x
   c. x.xx
   d. .xxx

   xxx. gives you a range of 000 to 111 = 0 to 7.

4. Why did processors switch from serial multipliers to parallel multipliers?
   a. Parallel multipliers became easier to design
   b. We had enough transistors that we didn’t care about size
   c. Serial multipliers requires less logic
   d. All of the above

   Parallel multipliers use a huge amount more logic, but as transistor sizes have shrunk we have tons of transistors to use so this is much less of an issue.

5. Which of these floating point operations is most likely to cause a divide by zero exception if there aren’t enough bits in the number representation?
   a. \( \frac{1}{(2.1\times10^{32}+5.6\times10^{5})-2.1\times10^{32}} \)
   b. \( \frac{1}{(5.6\times10^{5}+2.1\times10^{7})-5.6\times10^{5}} \)
   c. \( \frac{1}{(2.1\times10^{32}-2.1\times10^{32})-5.6\times10^{2}} \)
   d. \( \frac{1}{(5.6\times10^{2}-2.1\times10^{2})-5.6\times10^{2}} \)
(big+small) = big if we don’t have enough bits to shift the small value over to match the exponents. In that case we will have (big+small)-big = 0, and 1÷0.

6. Floating point normalization solves what problem?
   a. Exponential spacing of the values on the number line
   b. Linear spacing of the values on the number line
   c. Multiple representations for the same value
   d. Needing to shift the exponent for addition

Normalization forces a single representation for each value by insisting that the mantissa be shifted to the left as far as possible. The implied 1 before the mantissa in the IEEE standard forces the mantissa value to be shifted left until there is a leading 1.

4. Logic

1. If you choose two groups of 2 in a Karnaugh map instead of 1 group of 4, what do you get as the result?
   a. The wrong logic equation
   b. A logic equation with twice as many terms
   c. A logic equation with half as many terms
   d. A logic equation with XOR gates

The more groups you choose the more terms you have in the result. If you use two groups of two instead of 1 group of 4, then you will have two terms in the final equation instead of only 1 with the group of 4.

2. How many inputs will a 4-bit mux with a 3-bit select signal have?
   a. 3 inputs each 4 bits wide
   b. 4 inputs each 3 bits wide
   c. 8 inputs each 4 bits wide
   d. 16 inputs each 3 bits wide

The 3 bit select signal will select between \(2^3=8\) possible inputs. So there are 8 inputs, each of which is 4 bits wide.

3. Why does a decoder not have input 0 connected to the logic that generates the output?
   a. It does.
   b. Input 0 corresponds to binary output 0, which is all false, so we don’t use the input to generate any output.
   c. You can never have just input 0 active because the inputs are one hot.
   d. Zero cannot be represented with a one hot input.

When input zero is true, the output should be 000. So we do not need to use the input to generate all false outputs.

4. Why are edge-triggered flipflops better than transparent latches?
   a. They are faster because they only look at edge of the clock signal instead of the whole time it is high or low
   b. They allow the input to pass directly through to the output, so they speed up the circuit
   c. They don’t allow the input to pass directly through to the output, so they avoid feedback loops
   d. They are smaller than transparent latches because they only need two inverters

Edge-triggered flipflops use two latches back-to-back to load the input into the first latch when the clock is low and then transfer it to the second latch when the clock is high. This prevents the signal from feeding straight through and avoids feedback through the circuit.

5. Why is DRAM so much cheaper than SRAM?
   a. DRAM cells are made of two inverters so they are smaller
   b. DRAM cells are made of a single capacitor so they are smaller
c. DRAM cells are faster so you need fewer of them

D.R.A.M. cells are made of capacitors, which are much smaller than the 6 transistors needed to make a SRAM cell, so you can fit more of them into a chip.

6. Why is SRAM so much faster than DRAM?
   a. It isn’t.
   b. **SRAM cells are made of powered transistors so they can output a stronger signal, which means they can send out their data faster**
   c. SRAM cells are made of capacitors so they are smaller and their output doesn’t have to travel as far
   d. SRAM uses caches to make it faster

SRAM cells are made of inverters that feedback on each other. These inverters need power to run, which means they can output a much stronger signal than DRAM, where each cell is just a passive capacitor. When they output a stronger signal it takes less time for that signal to reach the end of the wire.

---

5. Processor Control and Datapath

![Diagram of processor control and datapath](image-url)

1. Why does our basic processor design have two memories?
   a. It doesn’t
   b. We don’t have a cache, so accessing the memory is slow
   c. Different parts of the instruction access the instruction memory and the data memory
   d. **Some instructions need to access the instruction memory and the data memory at the same time**

Load/store instructions need to both load the instruction (from the instruction memory) and access data (from the data memory) at the same time.

2. How many bits are needed for “Write register” in the figure?
   a. 1
   b. 5
   c. 16
   d. 32

5. This input chooses which register to write, so it needs 5 bits to choose from the 32 registers.
3. What is the output of the sign extender for non I-format instructions?
   a. Zero: non-I-format instructions do not have an immediate to sign-extend
   b. Nothing: non-I-format instructions do not have an immediate so the sign-extension logic is turned off
   c. Garbage: non-I-format instructions do not have an immediate so the value is whatever bits were in that part of the instruction
   d. Undefined: non-I-format instructions do not have an immediate so the sign-extension logic has unknown values coming into it and produces unknown values

Garbage. The sign extension logic sign extends the bits in the rd, shmt, and func fields of the instruction, no matter whether those are an immediate from an I-type instruction or the rd, shmt, and func fields for an R-type instruction. (the value is clearly defined, though.)

4. What determines the value of the ALUSrc MUX?
   a. The instruction type: I=Read data 2, R=Sign-extend
   b. The instruction type: I=Sign-extend, R=Read data 2
   c. The ALU Zero output: True=Read data 2, False= Sign-extend
   d. The value from Read data 2

The instruction type determines whether the ALU uses a second value from the register file (R-type) or the immediate field (I-type).

5. What are the values of ALUSrc, MemWrite, MemtoReg, and RegWrite for a lw instruction?
   a. ALUSrc=Read-data-2; MemWrite=0; MemtoReg=Read-data; RegWrite=1
   b. ALUSrc=Sign-extend; MemWrite=0; MemtoReg=Read-data; RegWrite=1
   c. ALUSrc= Sign-extend; MemWrite=0; MemtoReg=ALU-result; RegWrite=1
   d. None of these

The ALU calculates the address so it needs the immediate to add in (ALUSrc=Sign-extend); we are not writing to the memory with lw (MemWrite=0); the results are coming from the memory (MemtoReg=Read-data); and we do write the results into the register file (RegWrite=1)

6. What is the ALU operation for a sw instruction?
   a. Add
   b. Subtract
   c. Pass through
   d. Undefined (ALU results not used)

The sw instruction needs to calculate the address by adding the immediate value to the register input, so the ALU does an ADD.

6. Pipelining

1. Why did pipelining the processor help if all instructions take the same 5 pipeline stages?
   a. We can execute the instructions in parallel
   b. We can execute each instruction faster
   c. We can move data through the processor faster
   d. It didn’t: pipelining only helps when the instructions take different amounts of time

Pipelining allows us to execute instructions in parallel. (E.g., one instruction can be doing ID while the second one is doing IF.) Each individual instruction may not be faster, but we can do more at once by doing them in parallel.

2. What is the speedup for a processor when we add 10 ideal pipeline stages, but can only keep them half full with instructions?
   a. 10x
   b. 5x
   c. 2x
   d. Need more information
3. What is the change in throughput for a processor that takes 100ns per instruction unpipelined, when it is pipelined with 100 stages, where each stage needs a 2ns pipeline register? (Assume we can keep the pipeline completely full.)
   a. 1/3 (lower throughput)
   b. 3x
   c. 33x
   d. 100x
   We will get one instruction out every 100ns/100+2ns, or 3ns. So this is 33x higher throughput than the 1 every 100ns we started with.

4. What is the change in latency to complete each individual instruction for a processor that takes 100ns per instruction unpipelined, when it is pipelined with 100 stages, where each stage needs a 2ns pipeline register? (Assume we can keep the pipeline completely full.)
   a. 1/3 (lower latency)
   b. 3x
   c. 33x
   d. 100x
   Each instruction needs 100 stages, where each stage takes 100ns/100 logic and 2ns pipeline, or 3ns per stage times 100 stages is 300ns. Previously each instruction took 100ns, so it is 3x larger.

5. What happens in the MEM stage for an addi instruction?
   a. Nothing
   b. Load the value
   c. Calculate the ALU result
   d. Write back the ALU result
   Nothing. The addi instruction does not access the data memory, so it does nothing in the MEM stage.

6. What is the most important thing for getting performance from a pipelined design?
   a. Have as few pipeline registers as possible
   b. Make each stage fast
   c. Keep the pipeline full
   d. All of the above
   If you can’t keep the pipeline full then the others don’t matter.

7. Hazards
   1. What is needed to resolve the following hazard?
      sub R4, R13, R14
      add R14, R4, R13
      a. Forwarding from WB to EX
      b. Forwarding from MEM to EX
      c. Double-pumping the RF
      d. Can’t resolve: need to stall
      The second instruction needs R4 in the EX stage, but the results of R4 from the previous instruction are only in the MEM stage. So we need to forward from MEM to EX.

   2. What is needed to resolve the following hazard?
      lw R4, 12(R14)
      add R14, R4, R13
      a. Forwarding from WB to EX
      b. Forwarding from MEM to EX
c. Double-pumping the RF

d. Can’t resolve: need to stall

The second instruction needs R4 in the EX stage, but the results of R4 from the previous instruction are only available after the MEM stage. So we can’t forward them (they don’t exist yet) and have to stall.

3. Is stalling the pipeline a good way to solve hazards?

a. No, it hurts performance too much
b. No, it requires special compilers to insert stalls
c. A and B
d. None of the above

Stalls alone hurt performance too much. They don’t require a compiler, however, as you can use the same logic we use to do forwarding to detect when to stall.

4. What does the data hazard detection logic do?

a. Look at the instructions in the different pipeline stages to see if they have conflicting register file accesses
b. Look at the instructions in the different pipeline stages to see if they write to the register file
c. A and B
d. None of the above

The data hazard detection logic looks at the instructions in the pipeline to see which registers they access and which ones are writing to the registers. From this it can determine what forwarding needs to happen.

5. How many branch delay slots do we have if the branch decision is made in the MEM stage?

a. 0
b. 1
c. 2
d. 3

3. we need to wait for the IF, ID, and EX stages before we resolve a branch.

6. What does the ALU do on a beq instruction if we move the branch computation to the ID stage?

a. subtract to compute the jump condition
b. add to compute the jump condition
c. nothing
d. doesn’t matter

When we move the branch computation earlier in the pipeline we add a separate comparator to determine if the registers are equal. This means we no longer need the ALU to do that comparison, so what it does is undefined and doesn’t matter.

7. If 20% of the instructions are branches and we can only fill the branch delay slot 50% of the time, how much of a slowdown to we experience due to the branch delay slot?

a. 5%
b. 10%
c. 20%
d. 25%

20%*50% = 10% of the time we do an extra instruction, so we have a 10% slowdown.

8. Branch Prediction and Exceptions and Interrupts

1. Why do we need branch prediction on modern processors?

a. The processor runs too fast to stall the pipeline
b. The pipelines are so long that we could never fill all the branch delay slots
c. The pipeline is too long to kill instructions from a wrong branch
d. All of the above
2. What is the slowdown for processor that does one instruction per cycle and has a branch predictor that is 95% accurate with a 50 cycle misprediction cost? (Assume 20% of instructions are branches.)
   a. 2x faster
   b. **1.5x slower**
   c. 2.5x slower
   d. 3.5x slower
   
   20% of the time we will have a 1-95%=5% chance of paying a 50 cycle penalty in addition to the 1 cycle it normally takes. (0.2 branches/instruction)*(0.05 mispredictions/branch)*(50 cycles/misprediction)+(1 cyclenstruction) = 1.5 cycles per instruction, or 1.5x slower.

3. What is the reasoning behind the BTFN branch predictor?
   a. **Loops are the most common taken branches and they are usually backwards**
   b. Error code usually has forward jumps and are not frequently taken
   c. We guess 50% each way
   d. All of the above

4. What happens if you have a branch predictor with 8 entries, but your program has 9 branches?
   a. The predictor miss-predicts one of them
   b. **The predictor confuses two of them**
   c. The predictor puts one on the stack to hold the 9th branch
   d. The program crashes
   
   The predictor will put the results of two of the branches into the same place, which will cause it to confuse the two of them and probably result in a far worse prediction.

5. Why n-bit branch predictors better than 1-bit branch predictors?
   a. **You can make more miss-predictions before forgetting your previous prediction**
   b. You can predict for more branches
   c. You can predict more complicated patterns
   d. All of the above
   
   The larger number of bits allows you to keep your prediction through more miss-predictions.

6. When an exception occurs, what happens to the other instructions in the pipeline?
   a. They are all killed
   b. **They are killed if they come after the instruction with the exception**
   c. They are killed if they have a data dependency on the instruction with the exception
   d. The operating system decides which instructions to kill
   
   The instructions after the instruction with the exception have to be killed because we need to handle the exception before continuing.

9. Input/Output
   1. What is the worst-case for accessing data on a hard disk?
      a. The time it takes to move the head from the inner-most track to the outer-most track
      b. The time it takes to rotate the disk all the way around
      c. **The sum of A and B**
      d. All data takes the same amount of time to access
The worst-case is if you have to move the head all the way from the outside to the inside (or reverse) and then wait for the disk to spin all the way around to get the data under the head.

2. Rank the following technologies in terms of cost per MB: flash, DRAM, hard disk, tape
   a. (cheapest) flash < DRAM < hard disk < tape
   b. (cheapest) tape < hard disk < DRAM < flash
   c. (cheapest) tape < DRAM < flash < hard disk
   d. (cheapest) tape < hard disk < flash < DRAM

Tape is the absolute cheapest (just buy more tapes), hard disks are next cheapest, and then flash. DRAM is 10x more expensive than flash.

3. Why is it hard to make busses fast?
   a. Wires interfere with their neighbors
   b. Wires of different lengths take different amounts of time
   c. Wires get slower as they get smaller
   d. All of the above

All of the above.

4. What is the benefit of differential serial signaling over busses?
   a. Differential signals resist noise because it gets subtracted out at the end
   b. Differential signals resist noise because it gets added in at the end
   c. Serial signals are easier to build than parallel busses
   d. Serial signals use a slower clock because they send data in parallel

With differential signals noise affects both wires similarly, and by subtracting them in the end, you can subtract out the noise.

5. What is the benefit of memory-mapped IO?
   a. Faster
   b. Automatically moves data
   c. Uses regular instructions to access IO
   d. Does not pollute the address space

With memory-mapped IO you just need to know the address of the device and then do a regular load/store to access it. Memory-mapping is just a way to get the data; it doesn’t change the performance per se or make it easier to use. It does pollute the address space because you need to reserve addresses for the IO.

6. If you want to write a program with the absolute lowest possible latency to access an IO device, what approach would you use and why?
   a. Interrupts: the processor can do other work while it is waiting
   b. Interrupts: the operating system will handle telling the program when the device is ready
   c. Polling: the program will detect when the device is ready because it can’t do anything else
   d. DMA: the processor doesn’t have to do the transfer so it will be faster

In polling the processor does nothing but check if the IO device is ready. This means it will know as soon as possible when it is. With interrupts you have to wait for the interrupt handler to execute first.

10. Caches

1. Why do we want caches?
   a. DRAM is slow
   b. Processors are fast
   c. Lots of instructions access memory
   d. All of the above
2. Approximately how many processor cycles does it take to access data in DRAM? (E.g., on a cache miss.)
   a. 1
   b. 10
   c. 100
   d. 1000
   About 100. E.g., you can do 100 instructions in the time it takes to access one piece of data from DRAM.

3. How many lines will each way have in 32kB 8-way set associative cache with a 64 byte line size?
   a. 4096
   b. 512
   c. 64
   d. 8
   \[ \text{32kB} \div 64 \text{ bytes} = 512 \text{ lines total. The cache is 8-way, which means that each way will have } 512 \div 8 = 64 \text{ lines.} \]

4. Why are write-back caches better than write-through?
   a. Simpler: use a dirty bit to keep track of which data has been changed
   b. Faster: don’t have to write back data to DRAM when it is changed
   c. Smaller: don’t need a valid bit because they keep the changed data
   d. All of the above
   Write-back caches are more complicated (which is why they need a dirty bit) and not any smaller (they need a valid bit to indicate whether the line has garbage or not). They are faster, though, because you don’t have to write to DRAM on every write.

5. How many hits will the following access pattern generate in a 2-way set associative cache with 8 entries. (Assume the line size is 1 entry per line.) 4, 5, 6, 12, 13, 4, 5, 1, 12, 13, 5
   a. 2
   b. 3
   c. 4
   d. 5
   4, 5, 6 \rightarrow will put 4, 5, 6 in the first way. 12, 13 will put 12, 13 in the second way. These are all misses because the data isn’t in the cache. 4, 5 are now hits. 1 is a miss, and replaces 13 because it is least recently used. 12 now hits, but 13 misses (it was replaced by 1) and replaces 5. 5 is a miss. Total of 3 hits.

6. What happens to the average memory access time on a machine that takes 100 cycles to access DRAM, when you move from a 32kB cache (20% miss ratio, 1 cycle cache hit or miss) to a 2MB cache (5% miss ratio, 4 cycle cache hit or miss)?
   a. Goes up
   b. Stays the same
   c. Goes down
   d. Need more information
   \[ \text{32kB AMAT: } 80\% \times 1 + 20\% \times (1 + 100) = 21; \text{ 2MB AMAT: } 95\% \times 4 + 5\% \times (4 + 100) = 9. \text{ The AMAT goes down from 21 cycles to 9 cycles.} \]

11. Virtual Memory
1. Virtual memory does not do which of the following?
   a. Provide security by preventing programs from changing each other’s data
   b. Provide flexibility by letting programs use more memory than is physically available
c. Provide flexibility by letting the OS put programs anywhere in physical memory

d. Provide performance by making DRAM accesses faster

Virtual memory doesn’t make DRAM accesses faster. Indeed it makes them slower because first we need to do the virtual to physical translation.

2. How much data can a program access without a TLB miss if the TLB has 8 entries and the cache is 16kB with a 64 byte line size.
   a. 512 bytes
   b. 16kB
   c. 32kB

d. Don’t have enough information

You need to know the size of each page to calculate this. The size of the cache is irrelevant. With 8 TLB entries and a 4kB page you could access 8*4kB=32kB of data without needing to update the TLB. (Not all of them would be hits if the cache was only 16kB.) But if you had 2MB pages then you could access 8*2MB=16MB of data.

3. What is the average CPI of a program that is 33% load/store instructions, run on a 1 CPI processor with a perfect cache, and a perfect TLB that takes 2 cycles per translation?
   a. 0.66
   b. 1.66
   c. 2.66

d. 3.66

Every instruction needs 1.33 translations (1 for the instruction and 0.33 for the data). Normally an instruction takes 1 cycle, but now it takes 1*2 for the instruction TLB +0.33*2 for the data TLB +1 for the execution = 3.66.

4. Why is a virtually indexed, physically tagged cache a good design?
   a. You can do the translation and cache lookup in parallel
   b. The size of the cache is limited by the associativity and page size
   c. You don’t need the translation to determine if you have a cache hit
   d. All of the above

You can use the virtual address to look into the cache and extract the tag while you translate the address at the same time. Then when the cache is done with the lookup the translation is done, so you were able to do them in parallel.

5. Why is a virtually indexed, virtually tagged cache a good design?
   a. You can do the translation and cache lookup in parallel
   b. The size of the cache is limited by the associativity and page size
   c. You don’t need the translation to determine if you have a cache hit
   d. All of the above

Virtually-index/virtually-tagged caches let you access the cache directly with the virtual address so you don’t need to do a translation unless you have a miss.

6. Two programs want to share data. Can this be done with virtual memory?
   a. No, each program has its own virtual address space that maps to its own physical memory.
   b. Yes, they can both have pages that map to the same physical memory.
   c. Only if they use a physically-index and physically-tagged cache.
   d. Only if they use a virtually-index and virtually-tagged cache.

Programs can share data by having the OS map their virtual addresses to the same physical page. This does cause problems for the cache, but with a combination of flushing and process ID tags these can be overcome.

12. Parallelism

1. Why did clock speeds stop increasing in 2004?
   a. We can’t make faster transistors
b. The memory was too slow so there was no point

c. We couldn’t cool the chips

d. All of the above

We couldn’t cool the chips. We can make faster transistors, and caches help with the memory problem.

2. If a program is 80% parallelizable and I have 5000 processors, what is the best speedup I could achieve?

a. 5x
b. 4000x
c. 5000x
d. Need more information

You can make 80% of the program go 5000x faster. That part is therefore essentially instant, so you are left with the remaining 20%, or 1/5. That means you are 5x faster.

3. How do locks avoid data synchronization problems?

a. They prevent any other processor from changing the data while it is locked.
b. They prevent any other processor from accessing the data while it is locked.
c. They let the program keep track of whether any other processors are currently accessing the data.
d. They make copies of the data for each processor so they can update them at the same time.

The locks don’t actually prevent anything from happening. All they do is let the program know if another processor is trying to access the data. It is up to the program to be well-behaved and avoid accessing the data if another processor is.

4. What problem does cache snooping solve?

a. Prevents multiple processors from accessing the same data at the same time
b. Keeps DRAM up-to-date with changes that processors make in their own caches
c. Keeps each processor’s cache up-to-date with changes that other processors make in their own caches
d. All of the above

Snooping only makes sure that changes made in a processor’s cache are seen in the other caches as well.

5. What has happened as a result of incorrectly synchronized data accesses from parallel processors?

a. Programs crashed
b. Banks lost money
c. People died
d. All of the above

All of the above.

6. What speedup would we expect from a triple-issue pipeline that can do 1 load/store instruction and 2 other instructions at the same time for a program that is 1/3 load/store instructions?

a. Less than 3x
b. 3x
c. More than 3x, but less than 6x
d. More than 6x

Less than 3x. We know that programs have all sorts of nasty data and control dependencies that mean that even though 1 out of 3 instructions is a load/store (e.g., there are 2 non-load store instructions and 1 load/store for every three instructions) we won’t always be able to execute them at the same time. (Remember that we can’t forward from instructions that haven’t executed yet.)