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!

Note: Please take this exam seriously and pass it so you do not have to take future makeup exams! While it is essential to give students appropriate chances to demonstrate their knowledge, producing and grading an exam consumes significant teaching resources that could be spent otherwise improving the course. For example, this exam will cost the IT department roughly 10,000SEK in teaching time to prepare, administer, and grade. For that much effort we could have revised a whole lab.
0. Optional anonymous background questions
0.1. A B C
0.2. A B C
0.3. A B C D
0.4. Program: __________________

1. ISA 1
1.1. A B C D
1.2. A B C D
1.3. A B C D
1.4. A B C D
1.5. A B C D
1.6. A B C D

2. ISA 2
2.1. A B C D
2.2. A B C D
2.3. A B C D
2.4. A B C D
2.5. A B C D
2.6. A B C D

3. Computer Arithmetic
3.1. A B C D
3.2. A B C D
3.3. A B C D
3.4. A B C D
3.5. A B C D
3.6. A B C D

4. Logic
4.1. A B C D
4.2. A B C D
4.3. A B C D
4.4. A B C D
4.5. A B C D
4.6. A B C D

5. Processor Control and Datapath
5.1. A B C D
5.2. A B C D
5.3. A B C D
5.4. A B C D
5.5. A B C D
5.6. A B C D

6. Pipelining
6.1. A B C D
6.2. A B C D
6.3. A B C D
6.4. A B C D
6.5. A B C D
6.6. A B C D

7. Hazards
7.1. A B C D
7.2. A B C D
7.3. A B C D
7.4. A B C D
7.5. A B C D
7.6. A B C D

8. Branch Prediction and Exceptions and Interrupts
8.1. A B C D
8.2. A B C D
8.3. A B C D
8.4. A B C D
8.5. A B C D
8.6. A B C D

9. Input/Output
9.1. A B C D
9.2. A B C D
9.3. A B C D
9.4. A B C D
9.5. A B C D
9.6. A B C D

10. Caches
10.1. A B C D
10.2. A B C D
10.3. A B C D
10.4. A B C D
10.5. A B C D
10.6. A B C D

11. Virtual Memory
11.1. A B C D
11.2. A B C D
11.3. A B C D
11.4. A B C D
11.5. A B C D
11.6. A B C D

12. Parallelism
12.1. A B C D
12.2. A B C D
12.3. A B C D
12.4. A B C D
12.5. A B C D
12.6. A B C D

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.

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

2. How many practice exams did you do?
   - a. None
   - b. One
   - c. Several

3. How much time did you spend studying for this exam?
   - a. None
   - b. <2 hours
   - c. 2-8 hours
   - d. >8 hours

1. **ISA 1**

   1. Which of the following addresses is word-aligned?
      - a. 0110 1010 0110 1000 0110 0110 1001
      - b. 0000 0100 1000 1011 0110 1000 1100 1100
      - c. 0000 0010 1000 1010 0110 1000 1110 1111
      - d. 0000 0000 0000 1010 0110 1000 1010 1110

      *Word-aligned addresses must be multiples of 4, which means the last two bits are zero.*

   2. Why do we need both a register read and an ALU operation to write to memory?
      - a. We don’t
      - b. The register provides the write data and the ALU calculates the address
      - c. The ALU always does something regardless of whether we use the results
      - d. The register provides the address and the ALU proves the write data

      *The sw instruction uses the ALU to calculate the address (adding the immediate offset to one of the registers) and uses the other register for the data to write.*

   3. What address does the following code load?

      ```
      addi R9, R0, 24
      addi R1, R9, 12
      addi R9, R0, 12
      lw R9, 12(R1)
      ```

      - a. 12
      - b. 24
      - c. 36
      - d. **48**

      \[R9=24, R1=12+24=36, R9=12, \text{load address} = R1+12=48\]

   4. Why do branch labels (e.g., “skip:”) not take up any space in the program code?
      - a. We do not need a constant to jump to a label
      - b. The label is included in instruction after the label
      - c. **We just use the label to know what constant to put in the branch/jump**
      - d. They do take up space!

      *Branch labels are not part of the code. When the code is assembled, the assembler uses the label to know what constant to put into the jump offset. (E.g., if the label “skip” is 4 instructions after the branch where it is called, then the assembler will use the constant +4 as the offset to get to the instruction.)*
5. What value should be put in X to load the word starting at address 25 into R8?

\[
\begin{align*}
\text{lw} & \quad R2, 24(R0) \\
\text{sll} & \quad R2, R2, 8 \\
\text{lw} & \quad R3, 28(R0) \\
\text{slr} & \quad R3, R3, X \\
\text{or} & \quad R8, R2, R3 \\
\end{align*}
\]

\begin{itemize}
\item a. 8
\item b. 16
\item c. 24
\item d. 32
\end{itemize}

\text{R2 first loads bytes [24,25,26,27] and is then needs to be shifted left by 8 bits to be [25, 26, 27, 0]. R3 then loads [28,29,30,31] and needs to be shifted right by 24 bits to be [0,0,0,28]. The OR then creates [25,26,27,28].}

6. Why would we ever execute an add with R0 as a source register?

\begin{itemize}
\item a. To move data between registers
\item b. To create a NOP
\item c. \textbf{A and B}
\item d. We wouldn’t: it adds 0 so it is pointless
\end{itemize}

\text{We use add dst, src, R0 to move data between registers and an add to R0 will have no effect so it can be used as a NOP.}

\section*{2. ISA 2}

1. Why is the instruction “bne R9, 1, loop” not possible in the MIPS ISA encoding?

\begin{itemize}
\item a. Because R-type instructions only have enough immediate field bits for one constant
\item b. \textbf{Because I-type instructions don’t have enough immediate field bits for two constant}
\item c. Because I-type instructions need a constant offset, not a string like “loop”
\item d. Because J-type instructions don’t have any bits for registers
\end{itemize}

\text{The I-type instructions use X bits for the opcode, and from the remaining \( Y \) bits, \( Z \) are for the register, and there is only \( XX \) left for the constants.}

2. How would you load the value 9 into the PC?

\begin{itemize}
\item a. lui $pc, 0
\item b. addi $pc, $r0, 9
\item \textbf{jr} $pc
\item c. lui $pc, 9
\item d. \textbf{addi $ra, $r0, 9}
\item \textbf{jr $ra}
\end{itemize}

\text{You can’t write directly into the PC, but the jr instruction will load the value from register $ra into the PC.}

3. What happens with the immediate for branch instructions, and why?

\begin{itemize}
\item a. It is shifted right by 8 bits because the lower byte is not necessary
\item b. It is shifted right because instruction addresses are always multiples of 4
\item c. \textbf{It is shifted left because instructions are word-aligned}
\item d. It is not modified
\end{itemize}

\text{A and b are really the same thing since a word is 4 bytes.}

4. How can the processor know if an instruction is I-type and not R-type?

\begin{itemize}
\item a. \textbf{It looks at the opcode}
\item b. It looks if there is an immediate field
\item c. It looks if the destination register is in the right place
\item d. All of the above
5. This code will never finish. Why?

```assembly
main:
    addi R1, R0, 4
    addi R2, R0, 0
    jal sum_from_next_address
    j finish
sum_from_next_address:
    addi R1, R1, 4
    jal load_data
    add R2, R2, R3
    jr
load_data:
    lw R3, 0(R1)
    jr
```

a. It doesn’t save temporary and saved registers
b. **It doesn’t save the return address register**
c. It doesn’t follow the register convention for argument and return registers
d. All of the above

This code doesn’t save temporary and save registers nor follow the register convention, but it doesn’t need to do what it says. However, because it doesn’t save the return address when calling jal load_data, it will never return back to main.

6. Why do we decrement the stack pointer when we store data to the stack?

a. **Because the ISA defines it that way**
b. Because memory addresses go down
c. To avoid running out of space if we increment it
d. All of the above

This is just an arbitrary choice for the ISA: the stack grows down (and the heap grows up). As long as all methods use the same standard everything works together.

3. Computer Arithmetic

1. If we changed our 32-bit processor to have 256 registers, have the same size instructions, and keep similar I-, R-, and J-formats, how many bits would we waste if we loaded a 32-bit constant?
   a. 0
   b. 2
   c. 4
   d. 8

With 256 registers we need 8 bits per register. This means the I-type instructions will need 16 bits for the register values, which is 6 more than the 10 bits they have with 32 registers. This would reduce the size of their immediate field from 16 bits to 10 bits, which would require 4 immediate constants to fill up a 32 bit constant. With 4 10-bit immediates we would load 40 bits, which means we would waste 8 bits for our 32 bit constant.

2. Which of the following statements is **not** true about two’s compliment notation?
   a. We can easily determine if the number is negative by looking at the MSB
   b. **Addition is a non-inverted addition with carry set to 1**
   c. Subtraction inverts one input and does an addition with carry in set to 1
   d. There is only one zero
To subtract we need to set carry to 1 and add the inverse of the number we want to subtract.

3. For an 8-bit floating-point format: \((-1)^S*(.FFFF)*(2^EEE)\) where FFFF is unsigned and EEE is two’s complement, what is the largest positive value you can represent?
   a. 125
   b. 31.75
   c. 15.875
   d. 7.5
   The largest number is given by \((-1)^0*(.1111)*(2^011) = .1111<<3 = 111.1 = 7.5

4. Which of these is not a valid interpretation of the bits 1010 according to the representations we’ve discussed?
   a. -6
   b. -2
   c. 2.75
   d. 10
   Two’s compliment: -6, Signed-magnitude: -2, Unsigned: 10, 2.2 unsigned: 2.5.

5. How are binary numbers distributed along the number line for the fixed-point number format?
   a. Linearly
   b. Exponentially
   c. Depends on the binary point location
   d. Depends on the exponent
   Fixed-point counts just like regular binary, but everything is divided by the binary point shift. E.g., .22 counts 0,1,2,3, etc, but everything is divided by 4 (shifted right by two decimal places) so the interpretation is 0.0, 0.25, 0.5, 0.75, etc. Only floating point has exponentially distributed numbers.

6. Which is not a benefit of floating point over fixed-point?
   a. Floating point can represent larger numbers
   b. Floating point can represent smaller numbers
   c. Floating point addition is simpler
   d. Floating point does not require more bits
   Addition is much more complex because you have to shift to make the exponents match first. You don’t need more bits to do floating point; you just use them differently.

4. Logic
   1. How many output bits does an adder have if it is adding two 8-bit numbers?
      a. 5
      b. 8
      c. 9
      d. Depends on whether the numbers are two’s compliment or fixed-point
      8 bits plus 1 carry out, for 9 bits. This is the same for two’s compliment.
   2. (!A AND B) OR (!B AND A) is the equivalent of which of the following?
      a. A AND B
      b. A OR B
      c. A XOR B
      d. A XNOR B
      The logic equation says that A and B are different, which is XOR.
   3. Why are transparent latches worse than edge-triggered flipflops?
      a. They are slower. They look if the clock signal is high or low the whole time, instead of just looking at the edge of the clock
      b. They don’t avoid feedback loops as they allow the input to pass directly through to the output
c. They don’t allow the input to pass directly through to the output, so they slow down the circuit
d. They are bigger because they need more than 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.*

4. Which one of these circuits passes its current input to the output whenever the clock goes high?

a. ![Circuit A]

b. ![Circuit B]

c. ![Circuit C]

d. ![Circuit D]

The single latch with clock as the input control will get new data in whenever the clock goes high, and will constantly output that result.

5. Why is DRAM slower than SRAM?

a. There is a lot more of it, so it takes longer to get data from it to the processor
b. *It’s built of capacitors which have less charge and take longer to produce output*

c. It’s built of feedback loops which need external power to output their data
d. The wires are longer so it takes longer to charge up the wire and read the output

*DRAM stores charge in capacitors and they are tiny, so there is only a tiny amount of charge. Detecting the change in output from this tiny amount of charge is slow. SRAM uses feedback loops, which do require power, but this means they can drive their output much faster. The wire lengths are not dramatically different.*

6. How would the output of this circuit change if we removed the inverters at the end of each feedback loop? (One is labeled A, the other is directly before the Out. Ignore the B label.)

a. The output would be inverted, but faster
b. The output would be inverted, but slower
c. **The output would not change, but be faster**
d. The output would not change, but be slower
There are two inverters in a row, so they cancel each other out. Removing them will not change the output, but it will make the circuit faster since there are fewer gates for the signal to go through.

5. Processor Control and Datapath

1. Which part of the processor is storing state?
   a. PC, registers, ALU
   b. PC, registers, ALU, instruction memory
   c. PC, registers, ALU, instruction memory, sign extension
   d. **PC, registers, instruction memory**

   The ALU is just combinational logic. It takes whatever its inputs are and generates outputs. The other components all store values and update them on the clock signal.

2. Why did our single-cycle processor design have two memories?
   a. It doesn’t, we just treat them as two different memories to understand better
   b. Different parts of the instruction access the instruction memory and the data memory
   c. Some instructions need to access the instruction memory and the data memory at the same time
   d. We don’t have a cache so it is faster if we have two memories

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

3. What do we need to add to the processor shown above to support the \\textit{jr} instruction?
   a. Another read and another write inputs to the Register File
   b. Path from the PC to the Register File and another ALU function to the ALU control
   c. **Path from the Register File to the PC**
   d. All of the above

   We need to be able to send the PC that was stored with the \\textit{jal} instruction in the register file back to the PC.

4. Which type of branch and jump instructions are supported by the datapath shown above?
   a. j
   b. \textit{beq, j}
c. bne, j

5. What would happen if the RegDst MUX was removed from the datapath and it’s 1 input was wired directly?
   a. No instructions could write back to the register file
   b. I-type instructions could not write back to the register file
   c. I-type instructions would write back to the wrong register
   d. The processor would be slower due to hazards

   The RegDst MUX is used to chose the right destination register bits to specify which register to write into. This is needed because I instructions use different bits for their destination than R instructions. If this was removed, the I instructions would end up writing to registers defined by some of the bits in their immediate field.

6. Why do we not have hazards in the single-cycle processor?
   a. Every instruction finishes before the next one starts so the register file always has the right data
   b. No instructions are executing in parallel so there are no conflicts between instructions
   c. Each instruction has access to the whole processor while it is executing so there is no fighting for resources
   d. All of the above

   The pipelined register has problems because these things are not true!

---

6. Pipelining

1. What is the key thing that makes pipeline processors faster?
   a. They can execute instructions in parallel
   b. Instructions can take different numbers of cycles
   c. Instructions are split into smaller steps
   d. All of the above

   Executing instructions in parallel is the key to pipeline performance. We use each part of the processor for a different instruction at the same time.

2. What value will be in R9 after this code finishes executing on the pipeline shown above?
   R1=1, R4=1, R9=1
   beq R1, R4, skip
   addi R9, R0, 0
   addi R9, R9, 2
   addi R9, R9, 4
   skip:
add  R0, R0, 0  
end:
  a.  0  
  b.  1  
  c.  6  
  d.  12  

The pipeline as drawn has three branch delay slots, so we will execute all three addis before we notice that we should have jumped to skip. We will get \( R9 = 1 \) (beq) 0, 2, 6.

3. What value will be in \( R9 \) after this code finishes executing on the pipeline shown above?

\[
\begin{align*}
R1 &= 1, \ R4 = 1, \ R9 = 1 \\
\text{beq} \ R1, \ R4, \text{skip} \\
\text{add} \ R9, \ R0, \ 0 \\
\text{skip:} \\
\text{add} \ R9, \ R9, \ 2 \\
\text{add} \ R9, \ R9, \ 4 \\
\text{add} \ R0, \ R0, \ 0 \\
\end{align*}
\]

a. 0  
 b. 1  
 c. 6  
 d. 12  

The pipeline as drawn has three branch delay slots, so we will execute all three addis before we notice that we should have jumped to skip. We will then jump to skip and do the last two addis again! We will get \( R9 = 1 \) (beq) 0, 2, 6, 8, 12.

4. A processor takes 280ns for the longest instruction. The processor is pipelined with 14 (equal) stages using pipeline registers that take 10ns. What percentage of the resulting cycle time is used for computation?

a. 33%  
 b. 66%  
 c. 96%  
 d. 100%  

Each stage is \( 280/14 = 20 \text{ns} \) plus \( 10 \text{ns} \) register overhead, or \( 30 \text{ns} \). \( 20/30 \text{ns} \) is used for computation, or 66%.

5. In the pipeline above, why do we need to store the ALU’s Zero output in the EX/MEM stage pipeline register? (Note the wire that would answer this question is not shown in the diagram.)

a. We don’t  
 b. We need it in the MEM stage  
 c. We need it in the EX stage  
 d. We need it in the IF stage  

The Zero value is stored so that the IF stage can use it on the cycle after the EX stage to choose between PC+4 and the branch PC.

6. Does the pipelined processor keep the ISA’s promise of in-order and atomic execution?

a. Atomic: yes, In-order: yes  
 b. Atomic: yes, In-order: no  
 c. Atomic: no, In-order: yes  
 d. Atomic: no, In-order: no  

The pipeline executes instructions in neither an atomic nor in-order manner. The instructions now happen in 5 separate pieces (not atomically) and don’t happen in-order (e.g., one instruction’s EX may happen before another instruction’s MEM).

7. Hazards

1. Double-pumping the register file fixes which of the following problems?

a. Needing to read the register file early for branches  
 b. Needing to get data to the register file early for all dependent instructions
c. **Needing to read and write the register file at the same time**
   
   Double pumping just allowed us to read and write in the same cycle from different pipeline stages. It did not fix all of the dependent instructions.

2. What is needed to resolve the following hazard?

   ```
   add R14, R1, R13
   lw R1, 4(R14)
   add R14, R15, R15
   addi R14, R12, 0
   ```

   a. **Forwarding from MEM to EX**
   
   b. Forwarding from WB to MEM
   
   c. Can’t resolve, need to stall
   
   d. There is no hazard to resolve

   The second instruction needs R14 in the EX stage to calculate the address and the result is in the MEM stage, so we need to forward from the MEM stage to the EX stage.

3. What is needed to resolve the following hazard?

   ```
   lw R1, 4(R14)
   add R14, R1, R13
   add R14, R15, R15
   addi R14, R12, 0
   ```

   a. Forwarding from MEM to EX
   
   b. Forwarding from WB to MEM
   
   c. **Can’t resolve, need to stall**
   
   d. There is no hazard to resolve

   The second instruction needs R1 in the EX stage, but the results of R1 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.

4. Which of the following options contain only structural hazards?

   a. Calculating the branch condition and address at the same time; reading and writing the RF at the same time; writing the data read by the previous instruction.
   
   b. Reading the instructions and data at the same time; writing the data read by the previous instruction.
   
   c. Calculating the branch condition and address at the same time; reading the instructions and data at the same time; writing the data read by the previous instruction.
   
   d. **Calculating the branch condition and address at the same time; reading and writing the RF at the same time; reading instructions and data at the same time.**

   Writing the data read by the previous instruction is a data hazard and just requires a new forwarding path. The others all require extra hardware to do multiple things at the same time, which is a structural hazard.

5. A processor is ideally pipelined to 10 stages, but due to hazards, 10% of the instructions are NOPs. What is the speedup over the original processor?

   a. 10%
   
   b. 90%
   
   c. **9x**
   
   d. 10x

   *If the pipeline was full, an ideally pipelined processor with 10 stages will be 10x faster. But we waste 10% of our instructions, so we only realize 90% of the speedup, or 90%*10=9x.*

6. Which of these does not address a structural hazard?

   a. Having independent data and instruction memories
   
   b. Reading and writing the register file in the same cycle
   
   c. Extra comparator for branch decisions
   
   d. **Forwarding from a later stage**
Data forwarding addresses a data hazard (the data is somewhere else). All the others add hardware or modify hardware to allow multiple instructions to execute at the same time.

8. Branch Prediction and Exceptions and Interrupts

1. What happens if you have fewer entries in your branch predictor than you have branches in your program?
   a. The branch predictor can’t make a prediction so it will turn off
   b. The branch predictor will always predict incorrectly
   c. **The branch predictor will be very likely to predict less accurately**
   d. You won’t be able to run your program
      
      _Multiple branch addresses will point to the same entry in the branch predictor (just like multiple addresses point to the same place in a cache) and they will share predictors, which is very likely to result in worse predictions._

2. Which processor would you rather have, assuming the other characteristics of the processor are the same? Processor A with a 100 cycle branch misprediction penalty and a 99.9% accurate branch predictor or Processor B with a 10 cycle branch misprediction penalty and a 99% accurate branch predictor?
   a. A
   b. B
   c. **They’re both just as good**
   d. Need more information
      
      _For each branch, A will give you a penalty of 100*.001=0.1 and B will give you 10*.01=0.1. You don’t need to know the branch frequency since it would have the same impact on both._

3. Which processor would you rather have, assuming the other characteristics of the processor are the same, and the pipelining speedup is only limited by branch mispredictions? Processor A with a 100 cycle pipeline, a 100 cycle branch misprediction penalty and a 99.9% accurate branch predictor or Processor B with a 10 cycle pipeline, a 10 cycle branch misprediction penalty and a 99% accurate branch predictor?
   a. A
   b. B
   c. They’re both just as good
   d. Need more information
      
      _For each branch, A will give you a penalty of 100*.001=0.1 and B will give you 10*.01=0.1. However, A will have a 100x pipelining speedup and B will only have a 10x speedup. Both will be limited because of branch mispredictions, but since they have the same penalty, this will affect them in the same way._

4. For which kind of application would a BTFN branch predictor work best?
   a. Applications with plenty of code that jumps forward over error handling code
   b. **Applications with plenty of loops that are usually taken backwards**
   c. Applications with loops that start out being taken all the time and then switch to not being taken later in the program
   d. All of the above
      
      _Loops are the most common code and they typically go backwards._

5. Why is a 2-bit predictor often better than a 1-bit predictor?
   a. It learns faster
   b. **It changes more slowly**
   c. It uses less logic
   d. It predicts more accurately
      
      _The 2-bit predictor can be better because it changes its prediction more slowly, so it is less likely to be confused by an occasional misprediction._
6. If an interrupt occurs which causes the operating system to switch to another program. Whose responsibility is it to save the current program’s registers to the stack?
   a. The current program
   b. The caller
   c. Don’t need to
   d. The operating system

   The current program doesn’t know it is being interrupted (since it can’t control interrupts) so it wouldn’t know what registers to save. The caller is part of the current program. The new program doesn’t know what the other program is doing either. Only the operating system, which handles the program switch, knows to save and restore the program’s registers.

9. Input/Output

1. Why would you choose to use interrupts instead of polling?
   a. Polling has lower latency than interrupts
   b. Polling uses up less processor time
   c. Interrupts use up less processor time
   d. Interrupts have lower latency

   Interrupts may take longer (since you have to go through the interrupt handler) but they allow the processor to do other things while it is waiting for I/O. Polling, however, will much more quickly let you know when data is available since you don’t have to go through the interrupt handler, but you can’t do anything else while you are polling.

2. Which of the following is true when it comes to making busses fast?
   a. Wires interfere with their neighbors
   b. Wires of different lengths take different amounts of time
   c. To send more bits at the same time you need more wires in a bus
   d. All of the above

   Busses need many wires working together to send bits in parallel, and these wires interfere with the ones next to them, and take different amounts of time to send data if they are different lengths.

3. Why does reading large chunks of contiguous data from a spinning hard disk give enormously higher throughput than reading small chunks of data from random locations?
   a. The small chunks are stored on the inside track of the disk which is slower
   b. Moving the read head between each small chunk takes a long time
   c. The controller has to process more to find many small chunks than one large chunk
   d. All of the above

   The overhead is due to having to move around for each chunk vs. moving once to the data and then just reading it as the disk moves by. The small chunks are stored randomly on the disk so they are not all in the slowest inside track. Processing many small chunks is not much harder than a large one since the large one is made up of many sectors internally.

4. What does Ethernet not do to avoid errors?
   a. Detect data collisions
   b. Use separate wires for each sender
   c. Retry sending the data
   d. Use a checksum

   Ethernet uses a checksum to detect if the data has been corrupted by collisions and then retries sending it later. It does not use separate wires for each sender, but relies on a shared medium.

5. How do DMA and memory-mapping interact?
   a. DMA cannot be used if the device is memory-mapped
b. DMA uses the device’s interrupt to move data

c. DMA moves data by accessing the device directly

d. **DMA moves data using the device’s memory-mapped addresses**

*DMA just copies data using the same memory-mapped addresses as regular code might. The DMA avoids having to have a program running on the CPU take care of the copy.*

6. How does DMA make our computers faster?

a. It transfers data faster than the CPU can

b. It avoids the overhead of interrupts when data is ready

c. **It frees up the CPU from having to do the transfer**

d. All of the above

*DMA just relieves the CPU of the need to do the transfer itself so it can do other work.*

10. Caches

1. Why do we want caches?

a. **DRAM is slow**

b. Disks are slower than DRAM

c. Virtual memory requires it

d. All of the above

*We use caches because accessing data in DRAM is too slow. We use DRAM to cache data from disks, and virtual memory does not require a cache to operate.*

2. Which of the following assumptions is required for caches to be effective?

a. Recently used data is not likely to be used again soon

b. The amount of data being used at any one time is big

c. **Recently used data is likely to be used again soon**

d. Data is accessed in regular patterns

*Data doesn’t have to be accessed in regular patterns. For caches to work the recently used data needs to be reused and it needs to be small enough to fit in the cache so it will still be there when it is reused.*

3. What data will be in a 4-entry, direct-mapped, cache with a least-recently-used replacement policy and one byte per line after the following memory accesses? address: 3, 2, 1, 0, 4, 5, 3, 1, 2

a. 1, 2, 3, 5

b. **1, 2, 3, 4**

c. 1, 3, 4, 5

d. 1, 2, 4, 5

*We fill up to 3, 2, 1, 0, then we replace 0→4, 1→5, 3 hit, 5→1, 2 hit*

4. What data will be in a 4-entry, fully-associative, cache with a least-recently-used replacement policy and one byte per line after the following memory accesses? address: 3, 2, 1, 0, 4, 5, 3, 1, 2

a. **1, 2, 3, 5**

b. 1, 2, 3, 4

c. 1, 3, 4, 5

d. 1, 2, 4, 5

*The fully-associative cache can keep any 4 values, and the LRU replacement policy will replace the oldest one. This will result in the last 4 distinct values being in the cache, which are 5, 3, 1 and 2.*

5. Caches give programmers the illusion of what?

a. Atomic load/store instructions in a pipeline

b. A full 32-bit address space for each application

c. **A large and fast memory**

d. All of the above
The cache provides the illusion of making memory look both fast and large by storing the right data in a small, fast memory.

6. What is the difference between a direct-mapped cache with a least-recently-used (LRU) replacement policy and a direct-mapped cache with a most-recently-used (MRU) replacement policy?
   a. The MRU policy will work less well, leading to a lower hit ratio
   b. The MRU policy will work less well, leading to a higher hit ratio
   c. The MRU policy will effectively have only one entry that gets used
   d. The caches will behave the same

Direct-mapped caches only have one place to put every address, so there is no choice in terms of what you evict (or remove) when you install new data. Even using an incredibly stupid replacement policy like MRU will not have any impact. (For a fully-associative cache, an MRU policy would mean you always replace the last item you inserted, so you would only use one location.)

11. Virtual Memory

1. How much data can a program access without a TLB miss on a machine with a 64 entry, 8-way set associative TLB where each page is 4kB, and a 16kB cache with a 64 byte line size.
   a. 16kB
   b. 256kB
   c. 1MB
   d. Don’t have enough information

   With 64 TLB entries in total and a 4kB page you could access 64*4kB=256kB of data without needing to update the TLB. (Not all of them would be hits if the cache was only 16kB, though.)

2. What happens before the actual write to DRAM when a program with the following page table tries to write to address 14?

   VA\rightarrow PA on disk access bits
   2\rightarrow 8 0 read/copy on write
   1\rightarrow 9 0 read/write
   14\rightarrow 7 0 read/write
   16\rightarrow 14 1 read/write
   13\rightarrow 6 1 read only

   a. VA\rightarrow PA translation, then the data is loaded from the disk
   b. VA\rightarrow PA translation, then a copy on write
   c. VA\rightarrow PA translation, then a memory protection exception is generated
   d. VA\rightarrow PA translation

   The TLB has an entry for virtual address 14, so the TLB translates VA 14 to PA 7 before writing to physical address 14.

3. Writing to which address with the following page table would result in a page being loaded from disk?

   VA\rightarrow PA on disk access bits
   2\rightarrow 8 0 read/copy on write
   1\rightarrow 9 0 read/write
   14\rightarrow 7 0 read/write
   16\rightarrow 14 1 read/write
   13\rightarrow 6 1 read only

   a. 8
   b. 13
   c. 14
   d. 16
Writing to address 8 would result in a memory protection exception/segmentation fault since that address is not in the page table. Address 13 would result in a segmentation fault (since you don’t have permission), address 14 would result in a direct write to DRAM since the page is in memory, but address 16 requires that the data be loaded from disk before it can be written.

4. What can virtual memory do for you if you have an ISA with a 64-bit address space and 16GB of physical memory?
   a. Nothing: a 64-bit address space can address a lot of memory already so it is not useful
   b. Not needed: a 64-bit address space can have multiple programs access more than 16GB of memory without needing virtual memory
   c. A little: you can use the physical memory beyond 16GB instead of paging to disk for a single application
   d. A lot: you need virtual memory to keep multiple applications from crashing each other.

   Virtual memory allows us to have multiple applications each with their own private memory space with protection guarantees from the OS.

5. Why is a Virtually Tagged, Physically Indexed cache bad?
   a. Slow: have to do the translation before you can access the cache.
   b. Insecure: no way to prevent different programs from sharing data in the cache.
   c. Both A and B
   d. It’s not: it is both fast and secure

   A VTPI cache requires a translation before you can index into and then you get back a virtual tag, which doesn’t help you keep applications’ virtual address spaces separate.

6. The TLB operates like a cache for page table entries. Which of the following does a TLB have that a data cache does not?
   a. A replacement policy
   b. Dirty bits
   c. Permission bits
   d. Tag bits

The TLB needs to have permission bits to keep track of what the program is allowed to do with each page. Regular caches allow the program to do anything it wants with the data it finds, so it does not need permission bits. Both need dirty bits to know what to do with the data (cache) or page (TLB) if it is evicted.

12. Parallelism

1. Why do we do parallelism?
   a. To reduce power
   b. To get better performance
   c. To save power
   d. To get better performance, save power and overcome Moore’s law

Parallelism can be more power-efficient, but it doesn’t in itself reduce power. We only do it for performance. We wouldn’t do parallelism if it helped overcoming Moore’s Law but not improving performance.

2. What is a lock?
   a. It is a mechanism to protect data so only one processor can access it at a time
   b. It is a mechanism to let a program check if another processor is using the data
   c. It is a mechanism to synchronize data so that different caches see the same value
   d. All of the above

All locks do is provide a way for a program to check if another processor doing something. They don’t actually fix any synchronization, but they allow you to write a program that is synchronized.
3. How much faster can my program run if I have 9000 cores and 20% of the program cannot be parallelized?
   a. 5x
   b. 1800x
   c. 7200x
   d. 9000x
   
   *The best we can do is to make the 80% take zero time, which would be 5x faster.*

4. Which of the following used parallelism to improve performance?
   a. Associative caches
   b. Address and branch calculations
   c. Register file access
   d. All of the above
   
   *Caches use parallel address comparisons and lookups; the pipeline does the address and branch comparison in parallel with different ALUs; and the register file supports reading and writing 3 registers in parallel.*

5. The Intel Haswell Processor (CPU) with the Iris Pro 5100 integrated Graphics Processor (GPU) has the following characteristics: 40 GPU cores that can each execute 20.8B floating point operations per second per core and 4 CPU cores that can each execute 56B floating point operations per second per core. Which is better for a program that is 50% parallel?
   a. Run on the CPU cores
   b. Run on the GPU cores
   c. Same
   d. Need more information
   
   *For the CPU, we can run 50% of the program at 4x56B and 50% at 56B, for an average speed of 140B instructions per cycle. For the GPU, we can run 50% at 40*20.8B and 50% at 20.8B for an average speed of 426B instructions per cycle.*

6. The Intel Haswell Processor (CPU) with the Iris Pro 5100 integrated Graphics Processor (GPU) has the following characteristics: 40 GPU cores that can each execute 20.8B floating point operations per second per core and 4 CPU cores that can each execute 56B floating point operations per second per core. How much faster is running the parallel part on the GPU and the serial part on the CPU than running the whole thing on just the GPU?
   a. 4% faster
   b. 1.4x faster
   c. 2.8x faster
   d. Need more information

   *The whole thing on the GPU is 50% at 40*20.8B plus 50% at 20.8B, for an average of 426B instructions per cycle. Putting the serial part on the CPU gives us 50% at 40*20.8B plus 50% at 56B, or 444B, which is 4% faster.*

---

Make sure you circled the answers on the answer page. Answers on other pages will NOT be graded.