Operating Systems/Architecture Candidacy Exam FA 2019

Name: _____

Instructions: There are a total of eight (8) questions with a total of 100 points. The exam is closed book, so no notes, calculators, phones, etc. There are 180 minutes total for the exam. Good luck!

Question	**Score**
Q1) Virtual Memory (15 pts)	
Q2) Concurrency (15 pts)	
Q3) Interrupts, Traps, and System Calls (10 pts)	
Q4) File Systems (10 pts)	
Q5) Caching (15 pts)	
Q6) Instruction scheduling (15 pts)	
Q7) Performance and efficiency modeling (10 pts)	
Q8) Multiprocessors (10 pts)	
Total	

Q1) Virtual memory (15 pts)

Given below is a page table for a system with a virtual address space of 8192 bytes, a physical memory space of 4096 bytes and a page size of 1024 bytes. NOTE THAT ALL THE VIRTUAL ADDRESSES GIVEN BELOW ARE **DECIMAL**.

	Present / Valid	Physical Frame No.	Referenced	Modified
0	0	2	0	0
1	1	1	1	0
2	0	2	0	0
3	1	0	0	0
4	0	3	1	0
5	1	2	1	1
6	1	3	0	1
7	0	0	0	0

a) (6) Is there a page fault when a program generates address 3010 (DECIMAL)? If not, what physical address (in BINARY) is sent to the memory?

b) (6) Is there a page fault when a program generates address 7095 (DECIMAL)? If not, what physical address (IN BINARY) is sent to the memory?

c) (3) Explain why not all invalid memory accesses result in a segmentation fault in a paging-based virtual memory system.

Q2) Concurrency (15 pts)

A group of students is studying for the systems qualifying exam. The students can study only while eating pizza. Each student executes the following loop:

```
while (true) {
  pickUpSlice (); // pick up a slice of pizza
  study (); // study while eating slice;
}
```

If a student finds that the pizza is gone, the student goes to sleep until another pizza arrives! The first student to discover that the group is out of pizza phones Brothers Pizza to order another pizza before going to sleep. Each pizza has **S** slices. Write code to synchronize the student threads and the pizza delivery thread. Your solution should avoid deadlock and phone Brothers Pizza (i.e., wake up the delivery thread) exactly once each time a pizza is exhausted. No piece of pizza may be consumed by more than one student!

Q3) Interrupts, traps, and system calls (10 pts)

a) (3) How does the CPU know the addresses of the operating system's interrupt and trap handlers? When does the CPU become aware of these addresses and where are they kept?

b) (3) True or False: `read` system call may result in a disk write operation. Explain your answer.

c) (4) Some architectures (e.g. x86) default to using a hardware page table walker for TLB misses, while others (e.g. MIPS) trap to the kernel to perform TLB management in software. Discuss the tradeoffs between having a dedicated hardware unit versus an OS trap for managing the TLB miss event.

Q4) File Systems (10 pts)

Most file systems support pretty large files. In this question, we'll see how big a file various file systems support. Assume for all questions below that file system blocks are 4KB.

a) (2.5) Assume you have a really simple file system, *directfs*, where each inode only has 16 direct pointers, each of which can point to a single file block. Direct pointers are 32 bits in size. What is the maximum file size for *directfs*?

b) (2.5) A new file system uses direct pointers but also adds indirect pointers and double-indirect pointers; we call it *indirectfs*. Specifically, an inode within *indirectfs* has one direct pointer, one indirect pointer, and one double-indirect pointer field. Pointers, as before, are 4 bytes in size. Wat is the maximum file size for *indirectfs*?

c) (2.5) A compact file system, called *compactfs*, tries to save as much space as possible within the inode. Thus, to point to files, it stores only a single pointer to the first block of the file. However, blocks within *compactfs* store 4KB of user data and a next field, much like a linked list. What is the maximum file size for *compactfs*?

Journaling is a technique used in file systems to ensure that the file system returns to a consistent state after a crash.

a) (2.5) Data journaling can be used to ensure that file writes are atomic, i.e., either all or none of the write is visible after a crash. What are the steps involved in ensuring that the write operation is atomic?

Q5) Caching (15 pts)

a) (7) Consider a cache for a system that does not support virtual memory (i.e. no paging and no translation). The byte-addressable address space consists of Z bytes, the cache has S sets, A ways, and a block size of K bytes.

i) (2) If we want to address compulsory misses, and we do not change the capacity of the cache, what parameters will we change and how? Will the bit length of the tag become larger, smaller, or stay the same? Why?

ii) (2) If we want to address conflict misses, and we do not change the capacity of the cache, what parameters will we change and how? Will the bit length of the tag become larger, smaller, or stay the same? Why?

iii) (3) If block size decreases by 2x, associativity triples, and cache capacity goes up by 12x, how many bits are in each of the tag, index and block offset fields, in terms of the given parameters of the original cache?

b) (8) There are three programs, FOO, BAR, and BAZ, a reference microbenchmark "ALWAYS-HITS" and an architecture SHODAN-N where each processor in the SHODAN-N series varies only by the total size of its (highly associative) single-level cache. The size of the cache doubles with each increment of N (i.e. SHODAN-3 has 8 times as much cache as SHODAN-0). Assume that the access patterns of all three programs are **not pathological with respect to cache parameters** (i.e. assume as a simplifying assumption that changing among reasonable replacement policies would not have significant effects on performance, nor would the relative behaviors of the programs change significantly if the associativity or block size were modestly increased or decreased, etc.)

As FOO, BAR, and BAZ are run across the SHODAN series of processors in order from a SHODAN-0 to a SHODAN-7, the following behaviors are observed:

The performance of FOO is very good (similar to ALWAYS-HITS) on a SHODAN-0, but gets progressively worse by SHODAN-7, although it still performs reasonably well.

The performance of BAR is initially poor, and then increases greatly between SHODAN-2 to SHODAN-3, and then declines.

The performance of BAZ is initially poor, and gets progressively better from SHODAN-0 to SHODAN-7, but the rate of improvement decreases with every step.

i.) (6) Describe the properties of FOO, BAR, and BAZ that would produce these respective behaviors (you may assume that each program has uniform behavior if it simplifies your answer).

ii.) (2) Assume that you are designing a follow-on to the SHODAN series, the POLITO processors. Describe a cache memory system that will effectively serve all three programs and justify your answers. Note any compromises in the design where one program suffers at the expense of the others, and how.

Q6) Scheduling, Dependencies, and Hazards (15 pts)

Consider the following translation of

```
node * lists[256];
int sum [256];
. . . // assume above structures are initialized appropriately
do {
  node * somePtr = NULL;
  for (int i = 0; i < 256; ++i){
    node * currentPtr = lists[i];
    somePtr |= currentPtr;
    if( currentPtr ){
      sum[i] += currentPtr->val;
      lists[i] = currentPtr->next;
    }
  }
} while (somePtr);
```

into a RISC-style (pre-link, pseudo-)assembly code (in this case, MIPS):

```
1 |       LA   $s0, lists          ; //$s0 ← &lists
2 |       LA   $s1, sum            ; //$s1 ← &sum
3 |       ADDI $s2, $s0, 1024      ; //$s2 ← &lists[256]
4 |DW0:   XOR  $t1, $0, $0         ; do {
                                       node * somePtr = NULL;
5 |FL0:   XOR  $t2, $0, $0         ;   for (int i = 0; i < N; ++i){
6 |       ADDU $t3, $s0, $0        ; //&lists[i=0]
7 |       ADDU $t4, $s1, $0        ; //&sum[i=0]
8 |FL1:   LW   $t5, 0($t3)         ;     node * currentPtr = lists[i];
9 |       OR   $t1, $t1, $t5       ;     somePtr |= currentPtr;
10|       BEQ  $t5, $0, SKIP       ;     if( currentPtr ){
11|       LW   $t6, 0($t5)         ;       sum[i] += currentPtr->val;
12|       LW   $t7, 0($t4)         ;
13|       ADDU $t6, $t7, $t6       ;
14|       SW   $t6, 0($t4)         ;
15|       LW   $t6, 4($t5)         ;       lists[i] = currentPtr->next;
16|       SW   $t6, 0($t3)         ;
17|SKIP:  ADDI $t3, $t3, 4         ;     }
18|       ADDI $t4, $t4, 4         ;
19|       BNE  $t3, $s2, FL1       ;   }
20|       BNE  $t1, $0, DW0        ; } while (somePtr);
```

Assuming a) a 5-stage (Fetch-Decode-eXecute-Memory-Writeback) 2-wide in-order superscalar processor with all forwarding paths; b) hazard detection and branch resolution in D; c) even-aligned instruction fetch on 2-instruction (e.g. (4,5) but not (11,12)) boundaries; d) no branches will be taken, but inst 10 (BEQ) will be mispredicted as taken --

Schedule the execution, starting at the Fetch of instruction 4 through its completion at the Writeback of instruction 20, in the format below, using lower-case letters to indicate stalls (e.g. F d D X M W for an instruction stalled for one cycle in D). Show misfetched instructions and indicate, with a vertical arrow between dependent instructions, all forwarding (except for W→D) that occurs in your schedule.

CYCLE → Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
`4:  XOR  $t1, $0, $0`	F	D	X	M	W														
`5:  XOR  $t2, $0, $0`	F	D																	
`6: ADDU $t3, $s0, $0`		F																	
`7: ADDU $t4, $s1, $0`		F																	

Q7) Understanding and Modeling Performance and Efficiency (10 pts)

a) (6) The base CPI of a system, excluding memory stalls, is QUUX
Loads and stores collectively constitute A% of all instructions
Accessing the L1 data cache takes P cycles (accounted for in base CPI).
Accessing the L1 instruction cache takes D cycles (accounted for in base CPI).
Misses to main memory take an average of K cycles.
The L1 D-cache miss rate/access is MD. The L1 I-cache miss rate/access is MI.

Your team is considering either i) adding a unified L2 cache for both data and instruction accesses or ii) doubling the size of the existing L1 caches at a penalty of 1 extra cycle per access. From an AMAT optimization perspective, and assuming that the L2 miss rate for instructions is half that of the miss rate for data, what relationship would have to hold true for the L2 cache to be a better option?

b) (4) Assume an originally serial program for which A% of the execution is parallelizable across N cores (where N is a power of 2) and B% is not. Assume that, in the parallel version of the code, there is an additional overhead of $C\%*(1+\lg_2(N))$ that cannot be parallelized. If the serial and parallel versions have identical execution times for N=4, what is C?

Q8) (10 pts) Protocol-Behavior interactions on multi-core

a) (3) Consider two possible implementations of designs that support two threads:

i) A single core, 4-issue, 2-way simultaneous multi-threading dynamically scheduled (OoO) processor

ii) A multiprocessor with two 2-issue single-threaded OoO cores

Assume that both designs have a two-level cache hierarchy, and that the L1 cache size per core is identical.

Describe a two-threaded workload (either multi-threaded or multi-process) where each thread has a fixed amount of work to perform that would be expected to reach a point where both threads have completed execution significantly faster on i) than ii) and explain why.

Describe a two-threaded workload (either multi-threaded or multi-process) where each thread has a fixed amount of work to perform that would be expected to reach a point where both threads have completed execution significantly faster on ii) than i) and explain why.

b) (3) ARM's big.LITTLE processors feature pairs of A15 (3-wide OoO) and A7 (2-wide in-order) cores. Both cores use the same ISA.

Assume the following:
An A15 operates at 5 watts.
An A7 operates at 1 watt.
An A15 is has a cycle time that is 2/3 that of the A7.
An A15 has a CPI that is ¾ that of the A7.
Zero overhead for switching between cores (not true in reality).
100% efficient power-gating.

If you have to maintain an asymptotic average power budget of 3 watts, what is the maximum sustained speedup of a big.LITTLE system for a single-threaded program compared to a system with just an A7?

c) (4) Assume that your processor ISA's only support for synchronization is via an `atomic_increment` memory primitive that returns the current value `V` of a memory location `L` and writes `V+1` to `MEMORY[L]` as a single atomic action and an `MFENCE_ALL` instruction that orders both loads and stores within a single thread with respect to the `MFENCE_ALL` instruction. Using these primitives, provide the pseudo-code for an N-way barrier among `N` threads where `N` is a compile-time constant.