Spring 2019 Operating Systems/Architecture Candidacy Exam Exam # __________

Name: ____

Instructions: There are eight (8) questions with a total of 100 points. You may use the backs of the pages for additional scratch paper. The exam is closed book, so no notes, calculators, phones, etc. There are 3 hours (180 minutes) total for the exam. Good luck!

Question	**Score**
Q1) Virtual Memory (15pts)	
Q2) Concurrency (15pts)	
Q3) Kernel (10pts)	
Q4) Storage (10pts)	
Q5) Role of Branch Prediction in CPU Design (10pts)	
Q6) VLIW and Pipelining (10pts)	
Q7) Virtual Memory and Memory Hierarchy (15pts)	
Q8) Protocol-Behavior Interactions on Multi-core (15pts)	
Total	

Q1) Virtual Memory (15pts)

(1pt) Does a page table A) always contain virtual addresses, B) always contain physical addresses, or C) sometimes contain virtual or physical addresses depending on how the system is configured?

(2pts) Suppose we're using a **16-bit** system with 2-byte pointers. Suppose the system has 16-byte pages with a two-level page table that is evenly split between the levels. What is the size of the top level page table in bytes?

(2pts) Give an example why two different processes may have virtual addresses that map to the same physical address.

(1pt) What is the purpose of address space layout randomization (e.g., stack randomization)?

(2pts) List two (2) cases where a page fault handler would be triggered.

(2pts) Describe how swapping is related to page tables.

(5pts) Suppose we're using an **8-bit** system with 16-byte pages and a two-level page table. Assume the two-level split is such that each table contains the same number of entries. Suppose the following is a dump of **physical memory**:

```
00:  50 10 20 20 10 90 f0 50  e0 40 20 90 00 50 80 10
10:  90 00 c0 50 30 e0 30 00  f0 f0 60 20 10 d0 50 f0
20:  40 e0 40 f0 f0 00 f0 f0  a0 30 a0 f0 50 e0 40 f0
30:  50 00 f0 f0 90 e0 90 40  f0 10 d0 20 10 90 f0 50
40:  80 40 00 f0 40 90 e0 10  d0 90 30 00 f0 f0 90 e0
50:  60 40 f0 10 20 20 10 90  f0 30 b0 10 20 40 00 f0
60:  50 70 e0 50 f0 50 80 f0  60 20 90 d0 50 f0 80 40
70:  50 00 f0 d0 c0 f0 20 10  c0 f0 f0 60 60 30 50 40
80:  50 40 10 20 c0 50 f0 00  f0 f0 c0 90 20 30 f0 30
90:  70 50 f0 60 90 e0 90 00  60 20 50 50 00 00 70 c0
a0:  40 20 30 f0 20 e0 20 e0  50 00 10 40 40 30 30 50
b0:  60 00 20 10 40 f0 30 f0  40 50 00 f0 90 40 d0 f0
c0:  a0 50 20 50 70 90 30 40  50 20 40 d0 30 c0 f0 e0
d0:  60 40 10 20 c0 50 00 30  70 90 80 30 80 40 20 c0
e0:  00 60 10 30 40 f0 20 90  10 c0 70 90 60 f0 50 80
f0:  b0 d0 00 c0 50 00 30 40  40 90 e0 00 00 70 c0 90
```

Suppose we declare char* p = 0xba. What is *p, assuming the process's base page table register (e.g., cr3) contains 0x30?

Q2) Concurrency (15pts)

(5pts) Consider the following multi-threaded C code:

```
// Barrier waits for n threads to call barrier_wait before
// allowing the threads to continue.
void barrier_init(barrier_t* b, unsigned int n) {
  b->n = n;
  b->count = 0;
  sem_init(&b->semaphore, 0 /*not shared*/, n /*init val*/);
}
void barrier_wait(barrier_t* b) {
  int new_count = atomic_increment(&b->count);
  if(new_count == b->n) {
    sem_post(&b->semaphore);
  }
  sem_wait(&b->semaphore);
  sem_post(&b->semaphore);
}
```

Describe and **fix** any bug(s) you see, if any.

(5pts) Consider the following multi-threaded C code:

```
void removeFront(list_t* l) {
  mutex_lock(&l->writeMutex);
  if(l->head) {
    free(l->head);
    l->head = l->head->next;
  }
  mutex_unlock(&l->writeMutex);
}
int getFront(list_t* l) {
  mutex_lock(&l->readMutex);
  int data = (l->head) ? (l->head->data) : 0;
  mutex_unlock(&l->readMutex);
  return data;
}
```

Describe and **fix** any bug(s) you see, if any.

(5pts) Consider the following multi-threaded C code:

```
int joinQueue(server_t* servers, int numServers, job_t* j) {
  int r1 = rand() % numServers;
  int r2 = (r1 + 1 + (rand() % (numServers-1)) % numServers;
  return joinBestQueue(&servers[r1], &servers[r2], j);
}
int joinBestQueue(server_t* s1, server_t* s2, job_t* j) {
  mutex_lock(&s1->mutex);
  mutex_lock(&s2->mutex);
  server_t* best = (s1->numJobs < s2->numJobs) ? s1 : s2;
  insertQueue(best->queue, j);
  int numJobs = ++best->numJobs;
  mutex_unlock(&s1->mutex);
  mutex_unlock(&s2->mutex);
  return numJobs;
}
```

Describe and **fix** any bug(s) you see, if any.

Q3) Kernel (10pts)

(1pt) What is the difference between a process and a thread?

(1pt) Suppose the OS is currently running Process A. What does the OS do differently when context switching to Process B vs. context switching to another Process A thread?

(1pt) What happens when a user program reads a kernel memory address?

(1pt) Suppose I'm designing a new kernel that reuses the user stack rather than creating a separate kernel stack. What is a design flaw in doing this?

(1pt) What is the difference between an interrupt and an exception?

(1pt) Give an example of an interrupt.

(1pt) Give an example of an exception.

(3pts) Page-locking a piece of memory prevents it from being swapped out to disk. Give an example where it is necessary to page-lock memory for correctness.

Q4) Storage (10pts)

(1pt) What is "seek time" in the context of HDDs?

(1pt) Briefly describe one reason why HDDs are better than SSDs. Do not use generic terms such as "faster".

(1pt) Briefly describe one reason why SSDs are better than HDDs. Do not use generic terms such as "faster".

(1pt) What is "write amplification" in the context of SSDs?

(1pt) Give an example of "write amplification" in the context of SSDs.

(1pt) What is the purpose of RAID0 (striping)?

(2pts) List two (2) fields contained within an inode.

(1pt) What is filesystem journaling?

(1pt) Give an example where filesystem journaling is beneficial.

Q5) Role of Branch Prediction in CPU Design (10pts)

(1pt) Why is stalling, used for data dependencies, insufficient to also solve control dependencies?

(1pt) Consider a 2-wide ARM processor design where branch resolution is very late in a deep pipeline. If 25% of your instructions are branches, branch resolution is in the 4th pipeline stage, and your prediction is 75% accurate, what is the expected (additive) increase in CPI due to branch mispredictions (over perfect prediction)?

(3pts) Assume that a given static branch has a repeating pattern of TTTN. Going from a 1-bit bimodal predictor to a 2-bit bimodal predictor increases prediction accuracy by 25% (steady state). What are the gains for this pattern from using a 3-bit bimodal predictor? What common feature of all three prediction schemes described by 2-level branch predictors (like those described by Yeh & Patt) is fundamental in achieving higher accuracy for the above scenario with their predictors than with a bimodal predictor?

(5pts) 2-bit bimodal predictors are generally better than 1-bit bimodal predictors, but both can be subject to pathological behaviors. Given a 1-bit and a 2-bit bimodal predictor, both starting in their weakest not-taken state, fill in all (22) empty boxes to show a length 8 sequence of branch outcomes, and the associated predictions, wherein each predictor mispredicts only the circled entries, resulting in the 1-bit predictor having a higher accuracy than the 2-bit predictor.

Actual Branch Outcomes (N,T) →								
1-bit prediction state (N,T)	N	○			○			○
2-bit prediction state (N,n,t,T)	n	○	○		○	○		○

Q6) VLIW and Pipelining (10pts)

Consider a 2-wide single bundle VLIW pipeline of the form F-D-X-M-W

Assuming that the VLIW architecture can perform **1 memory and 1 non-memory operation in one bundle**, and that:

- There are 64 general purpose registers in the VLIW ISA,
- That all loads and stores hit
- There is **NO (data) hazard detection** and **ONLY forwarding between W and D**
- Branches resolve in X

(4pts) **Mark all W→ D forwarding** in the below VLIW code that has been scheduled so that there are no data dependence stalls (necessary, since the pipeline has no hazard detection!)

CYCLE → **Instruction**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
LW $4, 0 ($8)	F	D	X	M	W														
ADDI $8, $8, 4	F	D	X	M	W														
LW $3, 0 ($7)		F	D	X	M	W													
ADDI $7, $7, 4		F	D	X	M	W													
LW $2, 0 ($6)			F	D	X	M	W												
ADDI $6, $6, 4			F	D	X	M	W												
NOP				F	D	X	M	W											
NOP				F	D	X	M	W											
NOP					F	D	X	M	W										
SUB $4, $4, $3					F	D	X	M	W										
NOP						F	D	X	M	W									
NOP						F	D	X	M	W									
NOP							F	D	X	M	W								
NOP							F	D	X	M	W								
NOP								F	D	X	M	W							
ADDI $4, $4, 2								F	D	X	M	W							
NOP									F	D	X	M	W						
NOP									F	D	X	M	W						
NOP										F	D	X	M	W					
NOP										F	D	X	M	W					
SW $4, -4 ($8)											F	D	X	M	W				
BEQ $4, $0, FOO											F	D	X	M	W				

(6pts) Imagine that the poor efficiency in the previous part inspired a redesign. The pipeline has been modified to provide hazard detection, but at the cost of an additional cycle, producing a F-H-D-X-M-W pipeline with hazard detection in H, and register read and other decode stage functionality in D. Rewrite and reschedule the VLIW code for the new pipeline.

CYCLE → **Instruction**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	F	H	D	X	M	W													
	F	H	D	X	M	W													

Q7) Virtual Memory and Memory Hierarchy (15pts)

Assume that you have a system with the following properties and configuration:

- The system is byte-addressable with 16-bit words
- Physical address space: 10 bits; Virtual address space: 16 bits; Page size: 2^4 bytes
- VIPT L1 D-cache = 48 bytes, 3-way associative, with 8-byte blocks; write-allocate/write-back
- Fully associative 3 entry DTLB; both DTLB and L1 D$ use LRU policy.
- Entry for each TLB or $ entry consists of {valid, dirty, LRU-rank(00=most recent), tag, data}
  All metadata is given in binary. TLB data is in binary and $ data is in hexadecimal.
- Endianness: If the data block containing address 0x0006 was 0x0123456789ABCDEF, the word loaded from 0x0006 would have integer value = 0xCDEF.

DTLB:

1,0,00, 0000 1000 0000, 00 1100	1,0,10, 1101 0000 1101, 00 0001	1,0,01, 0001 1101 0000, 01 1111

L1 D$:

SET 0	1, 0, 01, 00 1100, 0x1234567887654321	1, 0, 00, 00 0001, 0xCD9CEF0990FEDCBA	0, 0, 10, 01 1001, 0xBAD1BAD2BAD3BAD4
SET 1	1, 0, 10, 01 1111, 0xFEEDEEC5D0D0F00D	1, 0, 00, 00 1100, 0x1FEEDEE15DEADC0D	1, 0, 01, 00 0001, 0x0102030405060708

(10pts) Given the initial contents of the DTLB and L1 D$ as shown above, fill in the register value blanks after each instruction executes:

```
LW      $1, 0x1D0C($0) ;       $1=0x__________
LW      $2, 0 ($1) ;           $2=0x__________
ADDI    $3, $2, 0x1111;        $3=0x__________
SW      $3, 2 ($1) ;
```

(5pts) Fill in the contents of the DTLB and L1 D$ in the table below, after **all instructions** above have executed:

DTLB (after):

___,___,___, _______________, __________	___,___,___, _______________, __________	___,___,___, _______________, __________

L1 D$ (after):

___,___,___, _______________, 0x_______________	___,___,___, _______________, 0x_______________	___,___,___, _______________, 0x_______________
___,___,___, _______________, 0x_______________	___,___,___, _______________, 0x_______________	___,___,___, _______________, 0x_______________

Q8) Protocol-Behavior Interactions on Multi-core (15pts)

Assume that two cores are about to run the same spin-lock protected code, with assumptions given in comments, and that they execute no instructions beyond OP13, and the cache is initially empty:

```
Check:  ADDI   $1, $0, 1       #OP1- set $1 to 1 (1 locked, 0 unlocked)
        LL     $2, 0($3)       #OP2- $3 contains pointer to lock
        BNE    $2, $0, Check   #OP3- repeat if locked
        SC     $1, 0($3)       #OP4- attempt to lock
        BEQ    $1, $0, Check   #OP5- repeat if failed
        LW     $1, 0($4)       #OP6- critical section – assume that the pointers stored
        LW     $2, 0($5)       #OP7- in $3, $4, and $5 point to separate cache blocks
        ADDU   $1, $2, $1      #OP8- and that the associativity of the cache is > 3
        ADDU   $1, $1, $6      #OP9- where the value of $6 may differ per thread
        ADDU   $2, $2, $7      #OP10- where the value of $7 may differ per thread
        SW     $1, 0($4)       #OP11- update protected value
        SW     $2, 0($5)       #OP12- update protected value
        SW     $0, 0($3)       #OP13- release lock
```

Assume that each core can execute 1 instruction per cycle, including (uncontended) bus transactions, that core 0 always wins when two bus operations are requested in the same cycle, and core 0 first executes OP1 in cycle 0 and core 1 first executes OP1 in cycle 1. Assume MSI coherence.

(8pts) For every dynamic instance of an instruction executed by core 1, list the cycle it completes in.

(3pts) Would MESI improve performance? Why/why not? What if there were more than two threads?

(4pts) Would VI hurt performance? Why/why not? What if there were more than two threads?