Practice Operating Systems/Architecture Candidacy Exam

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) Kernel (10 pts)	
Q4) Storage (10 pts)	
Q5) CPU design (15 pts)	
Q6) Pipelining (15 pts)	
Q7) Memory hierarchy (10 pts)	
Q8) Multiprocessors (10 pts)	
Total	

Q1) Virtual memory

(1pt) What is the purpose of a TLB?

(2pts) Suppose we're using an **8-bit** system with 16-byte pages and a set-associative TLB with a total of 8 sets. How many bits are there in the TLB tag?

(1pt) What is the purpose of a page table?

(2pts) Suppose we're using a **16-bit** system with 2-byte pointers. Suppose the system has 64-byte pages with a single-level page table. What is the size of a page table in bytes?

(1pt) What is the purpose of making a page table multi-level?

(3pts) Describe how Copy-On-Write (COW) affects the page tables in the parent and child process in fork().

(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 = 0xca. What is *p, assuming the process's base page table register (e.g., cr3) contains 0x80?

Q2) Concurrency

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

```
bool send_money(acct* src, acct* dst, unsigned int amount) {
  bool sent = false;
  mutex_lock(&src->mutex);
  if (src->value >= amount) {
    src->value -= amount;
    mutex_lock(&dst->mutex);
    dst->value += amount;
    mutex_unlock(&dst->mutex);
    sent = true;
  }
  mutex_unlock(&src->mutex);
  return sent;
}
```

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

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

```
mutex_t queueMutex;
int queueMutexInit = 0;
int insertQueueThreadSafe(queue_t* q, void* data) {
  if (queueMutexInit == 0) {
    queueMutexInit = 1;
    mutex_init(&queueMutex);
  }
  mutex_lock(&queueMutex);
  int ret = insertQueue(q, data);
  mutex_unlock(&queueMutex);
  return ret;
}
```

Describe any bug(s) you see, if any.

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

```
void push(void* data) {
  mutex_lock(&mutex);
  queue.insert(data);
  cond_signal(&cv);
  mutex_unlock(&mutex);
}
void* pop() {
  mutex_lock(&mutex);
  if (queue.empty()) {
    cond_wait(&cv, &mutex);
  }
  void* data = queue.pop();
  mutex_unlock(&mutex);
  return data;
}
```

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

Q3) Kernel

(1pt) Give an example where a page fault **results** in an application crash.

(1pt) Give an example where a page fault **doesn't result** in an application crash.

(1pt) Why is the kernel stack separate from the user program's stack?

(3pts) List three (3) events that generate different interrupts.

(2pts) To implement a context switch, what data needs to be saved?

(2pts) Consider the following code in the kernel space for swapping the contents of two non-overlapping 4kb arrays:

```
void swap4kb(void* a, void* b) {
  char tmp[4096];
  assert(a != NULL);
  assert(b != NULL);
  memcpy(tmp, a, 4096);
  memcpy(a, b, 4096);
  memcpy(b, tmp, 4096);
}
```

Describe any bug(s) you see, if any.

Q4) Storage

(1pt) Briefly describe the data that is stored in the page cache.

(1pt) Briefly describe the data that is stored in the disk cache.

(1pt) Briefly describe the purpose of the swap partition.

(1pt) Briefly describe one reason why SSDs are better than HDDs.

(1pt) Briefly describe one reason why HDDs are better than SSDs.

(1pt) What is the difference between RAID4 and RAID5, which is better, and why?

(2pts) What is the purpose of journaling in filesystems?

(2pts) What is the purpose of log structuring in filesystems?

Q5) Understanding processor performance (15 pts)

(4pts) Assume that you are moving from a single-cycle MIPS processor design with frequency F and CPI C_{base} to a 5-stage MIPS processor design with frequency 3F. At what CPI would your new processor have twice the performance of the original?

(4pts) Assume that your processor has no cache, and that your memory takes a fixed amount of time independent of cycle time to access, stalling the pipeline while memory is being accessed. If memory stalls currently account for K% of CPI, what is the maximum speedup that can be achieved by increasing the clock frequency by a factor of G?

(7pts) Assume that a parallelization optimization across two cores for an originally serial program provides an improvement factor of K for the X% of execution that it applies to, but requires an additional Y% of the original execution worth of computation to be performed in order to apply the optimization. Assuming that the tradeoff is worthwhile (i.e. speedup >1) what are the improvements in **throughput** and **latency** over the original serial code if two copies of the two-way parallelized program are run across four cores? **Express your throughput improvement as a function of your latency improvement.**

Q6) Pipelining with data hazards (15 pts)

(a) (4) Label all data dependencies and (false/anti)-dependencies in the below code fragment with an arrow and their data dependency type.

```
1 |  FOO: lw      $2, 0($4)
2 |  lw           $3, 0($5)
3 |  addu $2, $2, $3
4 |  sw           $2, 0($4)
5 |  lw           $4, 4($4)
6 |  addi  $5, $5, 4
7 |  bne     $4, $0, F00
```

(b) (7) Consider the instructions from (a) scheduled on a five stage, scalar MIPS pipeline (FDEMW) **with full forwarding and bypass networks and hazard detection and branch resolution in D**. In the table below, for each cycle, indicate the cycle an instruction completes a stage with a capital letter (FDEMW) and indicate stalls with a lowercase letter (fdemw). **Indicate by drawing a vertical arrow when a value is bypassed from one instruction to another in the cycle that the forwarding occurs**. For simplicity, omit W→D bypassing arrows. Assume all loads/stores are 1-cycle hits, that there are no exceptions, and that all branches are perfectly predicted.

CYCLE	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
FOO: lw $2, 0($4)																			
lw $3, 0($5)																			
addu $2, $2, $3																			
sw $2, 0($4)																			
lw $4, 4($4)																			
addi $5, $5, 4																			
bne $4, $0, FOO																			

(c) (4) Given the schedule below, list what forwarding is demonstrated as being supported (and list the pair of instructions that demonstrate this, for each pipeline stage pair) and where

hazard detection and branch resolution occur.

CYCLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
FOO: lw $2, 0($4)	F	D	E	M	W														
lw $3, 0($5)		F	D	E	M	W													
addu $2, $2, $3			F	D	e	E	M	W											
sw $2, 0($4)				F	d	D	e	E	M	W									
lw $4, 4($4)					f	F	d	D	E	M	W								
addi $5, $5, 4							f	F	D	E	M	W							
bne $4, $0, FOO									F	D	E	M	W						

Q7) Caching (10 pts)

(a) (4) The base CPI of a system, **excluding memory stalls**, is C

Loads and stores collectively constitute LS% of all instructions
Accessing the L1 data cache takes K cycles (accounted for in base CPI).
Accessing the L1 instruction cache takes P cycles (accounted for in base CPI).
Misses to main memory take an average of D cycles.
The L1 D-cache miss rate/access is M_D. The L1 I-cache miss rate/access is M_I.

Your team is considering adding an L2 cache <u>for data accesses only</u>. **Assuming that the L2 access time is 4K, how high can the L2 miss rate be and still improve AMAT?**

(b) (6) 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 the replacement policy would not have significant effects on performance and that FOO, BAR, and BAZ do not vary wildly in performance when increasing associativity beyond two, 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.

a) 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).

b) 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.

Q8) Multiprocessors and topologies (10 pts)

(2pts) What is the intuitive meaning of bisection bandwidth?

(8pts) Using LL/SC, safely insert an element into a concurrently shared list that may have an arbitrary number of readers and writers. You should assume that no reader or writer will delete nodes from the list and that there is an invariant header node that is always valid. Only provide the insertion code below, ignoring how the appropriate insertion point was located. Specify any necessary invariants.