Name: ____

Instructions: There are nine questions with a total of 200 points. Answers must fit in the space provided. Blank pages provided at the rear of the exam can be used for scratch paper – **these will not be graded**. The exam is closed book, so no notes, calculators, phones, etc. A reference sheet for assembly instructions is provided for questions that require reading/writing assembly code. There are 3 hours (180 minutes) total for the exam. Good luck!

Question	**Pts Possible**
Q1) Kernel and Processes	25
Q2) Virtual Memory	20
Q3) Concurrency	25
Q4) IO and Persistence	20
Q5) Cross-layer	20
Q6) Datapaths and Dependencies	20
Q7) Caches and Memory	20
Q8) Larger Instruction Window Execution	25
Q9) HW Support for Parallelism and Concurrency	25
Total	**200**

Q1) Kernel and Processes

(1) [5pts] Why is the kernel stack stored separately from the user program's stack? Why does each process have its own kernel stack rather than sharing with other processes?

(2) [5pts] How does the CPU know where to find functions for handling interruptions and traps? When does the OS perform initialization to associate handlers with interrupt codes?

(3) [5pts] Does an exception (trap) always cause a user process to be terminated during its execution? If so, why? If not, provide an example.

(4) [5pts] Explain the distinction between the concepts of kernel threads in the same process and separate processes.

(5) [5pts] Consider we have four jobs with (arrival-time, runtime) pair as (0, 320), (0, 600), (100, 500), and (400, 250). What is the lowest possible average turnaround time a non-preemptive schedule can provide? What does that schedule look like? (Turnaround time of a process is defined as the time between its arrival and finish.)

Q2) Virtual Memory

(1) [7pts] Suppose we have a multi-level page table with an 8KB page size and 8-byte PTE size. How many layers should this page table have to support a 64-bit virtual address space?

(2) [7pts] Give two examples where a physical memory page (on DIMM) may be mapped into the virtual address spaces of two different processes at the same time.

(3) [6pts] Can a process cumulatively allocate more memory through `malloc` than the machine’s total physical memory? If so, how does the OS achieve that? If not, why not?

Q3) Concurrency

(1) [7pts] Explain why deadlock will not occur if all participating threads acquire locks in the same total order.

(2) [9pts] Consider the following multi-threaded C code that contains at least one bug:

```c
bool transfer(acct* src, acct* dst, unsigned int amount) {
  assert(src != dst);
  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.

(3) [9pts] Consider the following multi-threaded C code that contains at least one bug:

```
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.

Q4) I/O and Persistence

(1) [5pts] List at least two methods for the CPU to know if an I/O event happens. Explain the pros and cons for them against each other.

(2) [4pts] If a CPU issues a load instruction, does it have to always go through the memory controller on one of the DIMM slots? If yes, explain why. If not, provide an example.

(3) [6pts] Suppose all file system caches (e.g., block cache, inode cache, etc.) are cleared and all files in this file system only contain one data block. How many IOs will be triggered during the system call `open(“/foo/bar”, O_RDONLY)`? **(To get partial credits, you may describe what are each IO is for.)**

(4) [5pts] Explain the purpose of journaling in a file system.

Q5) Cross-layer

(1) [5pts] Consider RAID-0 and RAID-1 configurations of two disks. For each configuration, describe at least one advantage and one disadvantage compared to a two disk2 JBoD (Just-a-Bunch-of-Disks).

(2) [6pts] In operating systems, LRU is commonly used as the cache eviction policy. However, in hardware, almost no modern high-performance processor implements LRU in its cache. Why?

(3) [9 pts] Virtual Memory Access in HW

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-Cache use LRU policy.
- Entry for each TLB or Cache entry consists of {valid, dirty, LRU-rank (00=most recent), tag, data}

All metadata is given in binary. TLB data is in binary and cache 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-Cache:

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, 0xFEEDD0D0EEC5F00D	1, 0, 00, 00 1100, 0x1FEEDEE15DEADC0D	1, 0, 01, 00 0001, 0x0102030405060708

Given the initial contents of the DTLB and L1 D$ as shown, and the next executing instruction being

LW $1, 0x80E($0);

then

i) What physical address does this instruction request from **data** memory?

ii) fill in the register value after the instruction executes.
REG[$1] = 0x____________

Q6) Datapaths and dependencies

(1) [5pts] Control

Assuming that branch delay slots are <u>not</u> present, draw the Control Flow Graph(CFG) for the below function. Label each node in the CFG with the range of instructions contained (e.g. 1-2) in that basic block.

```
1. Aardvark:
      add $v0, $0, $0        ; set return value
2.    bne $a0, $0, Badger    ; check for null ptr
3.    jr $ra                       ; return $v0
4. Badger:
      lw $t0, 0($a0)         ; load current value
5.    bne $t0, $a1 Capybara ; don't count non-matches
6.    addi $v0, $v0, 1      ; Match, increment return value
7. Capybara:
      addi $sp, $sp, -16    ; reserve stack space
8.    sw $a0, 12($sp)       ; store argument 0
9.    sw $a1, 8($sp)        ; store argument 1
10.    sw $ra, 4($sp)       ; store return address
11.    sw $v0, 0($sp)       ; store current total
12.    lw $a0, 4($a0)       ; load left branch ptr into  arg0
13.    jal Aardvark         ; recurse left
14.    lw $t0, 0($sp)       ; restore previous total
15.    add $v0, $v0, $t0    ; sum with left branch
16.    sw $v0, 0($sp)       ; store current total
17.    lw $a0, 12($sp)      ; restore arg 0
18.    lw $a1, 8($sp)       ; restore arg 1
19.    lw $a0, 8($a0)       ; load right branch ptr into arg0
20.    jal Aardvark         ; recurse right
21.    lw $ra, 4($sp)       ; restore return address
22.    lw $t0, 0($sp)       ; retrieve previous total
23.    add $v0, $v0, $t0    ; sum right total with previous
24.    addi $sp, $sp, 16    ; release stack space
25.    jr $ra               ; return $v0
```

This page left intentionally blank – you may use it for scratch space; responses here will not be graded.

CFG answer space here:

(2) [5pts] Data dependence

Label all data dependencies and (false/anti)-dependencies, as they would be seen in a single iteration of the below code fragment (i.e. ignore loop-carried dependencies) with an arrow and their data dependency type (RAW, WAW, WAR).

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

(3) [10pts] Pipelining with data hazards and forwarding

Consider the loop consisting of the instructions below, to be 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, <u>by drawing a vertical arrow bound to a specific cycle,</u> when a value is bypassed from one instruction to another <u>in the cycle that the forwarding occurs</u>.** 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. **Schedule 2 iterations of the loop, to completion.**

CYCLES	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
FOO: lw $3, 0($5)	**E**																			
lw $2, 0($4)	**D**																			
addu $2, $2, $3	**F**																			
sw$2, 0($4)																				
lw $4, 4($4)																				
addi $5, $5, 4																				
bne $4, $0, FOO																				

Q7) Caches and memory

(1) [5pts] Cache basics

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 P bytes, the cache has S sets, W ways, and a block size of B bytes.

i) What is the capacity of the cache?

ii) How many bits are needed for each of the tag, index, and block offset fields?

iii) Assuming that each tag array entry also stores a valid bit, a dirty bit, and two coherence state bits, how many total bits are needed for the entire tag array for the entire cache?

(2) [6pts] Cache parameter optimization

For each of the following, list **both**: 1) which one of the "3 Cs" (Compulsory, Capacity, Conflict) of cache misses is being addressed by the proposed change to a cache, and 2) what the dominant cost (negative considerations) of applying the proposed change is likely to be. Assume that any parameters not mentioned are not modified.

i) Associativity doubles, number of sets halves

ii) Number of sets quadruples, associativity halves

iii) Block size doubles, associativity halves

(3) [9pts] Cache performance

Assume that the base CPI (cycles per instruction) of a system, **excluding memory stalls**, is **C**. Assume that no memory stalls overlap with each other or any other hazard included in **C**.

Loads and stores collectively constitute **LS**% of all instructions
Accessing the L1 data cache takes **A** 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 **M** cycles.
The L1 D-cache miss rate/access is **D**. The L1 I-cache miss rate/access is **I**.

i) What is the CPI of the above system, **including memory behavior**?

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

Q8) Larger Instruction Window Execution

(1) [5pts] Branch prediction basics

For the below, ignore all PC_i->PC_j aliasing for all i != j.

i) Give an example of a code pattern for which a bimodal predictor with 2-bit counters will have a high (95+%) prediction accuracy?

ii) Give an example of a code pattern that a two-level predictor with per-branch history and 2-bit counters can accurately predict that a 2-bit bimodal predictor cannot.

iii) Give an example of a code pattern that a two-level predictor with global history can and 2-bit counters accurately predict that a two-level predictor with per-branch history cannot.

(2) [5pts] Branch performance impacts

Consider two pipelines:

FOO: a scalar, in-order F|D|E|M|W 5-stage pipeline with branch resolution in D

BAR: a scalar, in-order F1|F2|Q1|D|Q2|E|M1|M2|W 9-stage pipeline; branch resolution in D

i) If branches are 20% of instructions and branch prediction accuracy is 95%, what is the CPI impact of branch misprediction (i.e., branch misprediction stalls) for each pipeline?

ii) Would the same approach to measuring CPI impact apply to deeper, Out-of-order pipelines? Why, why not?

(3) [15pts] Instruction level parallelism

i) [3pts] OoO HW employs a form of internal resource virtualization to elide false dependencies and anti-dependencies while preserving true dependencies. What is this process called, and what resource is being virtualized?

ii) [4pts]One key limiter in adopting OoO execution in microprocessors was how to continue to support precise exceptions. What mechanism in a modern OoO processor is primarily responsible for precise exception support?

iii) [8pts] Superscalar In-order Processors

Consider a 6-stage, in-order FDHXMW pipeline, extended to be 2-wide, with full forwarding and bypass support, and all functional units are replicated 2x, including data memory ports. Assume fetch in F, decode in D, hazard detection and branch resolution in H, execute in X, Memory access in M, and Writeback in W. Schedule the below instructions for this pipeline, assuming that **no other instruction can advance into a pipeline stage where any instruction is currently stalled**. Ignore any instructions after the branch.

Indicate all forwarding of values with <u>a vertical arrow in the cycle it occurs.</u>

CYCLES	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Label: lw $4, 0($2)	**F**																		
lw $5, 0($4)	**F**																		
add $4, $4, $5																			
sw $4, 0($2)																			
lw $2, 4($2)																			
sltiu $4, $2, 1																			
nor $4, $4, $5																			
bne $0, $4, **Label**																			

Q9) HW support for parallelism and concurrency

(1) [5pts] Coherence

Describe a scenario (a set of parallel program behaviors) where an MSI protocol will generate substantially more coherence traffic than an MESI protocol. Specifically describe why the addition of the E state provides benefits in this scenario.

(2) [10pts] Amdahl’s law

i) Consider the following scenario: An initially serial application, with a larger than L2 but smaller than L3 working set, is parallelized across N cores. Of the serial application time T, P seconds are parallelizable. Is a parallel latency of (non-trivially) less than $(T-P) + P/N$ **possible**, and, if so, how/why? If not, why not?

ii) Consider the following scenario: An initially serial application, with S time serial and $4S$ time parallelizable, is parallelized across M cores. However, to support parallelization, new code had to be added that accounts for a one time cost of Y seconds. How large can Y be, in terms of S, for there to be at least a 2x speedup when $M = 16$?

(3) [10pts] HW atomics and support for synchronization/concurrency

Using the LL (load-linked) and SC (store-conditional) atomic primitives, implement, in assembly, (i.e. using a sequence of 32-bit RISC-style instructions from the provided note sheet at the back of the exam) a blocking mutex acquire function that takes one argument (the address of the mutex state) and returns no value, with busy-loop (i.e. empty loop doing nothing) based exponential backoff on non-acquisition. For simplicity in implementation, you may assume that your busy loop will never need to iterate more than 2^{30} times (i.e. you may safely ignore counter overflow issues with a 32-bit counter)

// assume lock stateis defined as 0:= unlocked; 1:=locked

Acq: // label for initial function entry; function argument is in register $a0

jr $ra // return

(SCRATCH PAPER – **NOT GRADED)**

(SCRATCH PAPER – **NOT GRADED)**

MIPS-32 instructions, derived from P&H COD and the MARS help information. MIPS has 32 (mostly) general purpose registers – see next page for details and naming conventions. “imm$_X$” indicates a signed immediate represented in x bits and “imm$_{xz}$” represents an unsigned immediate. Updated registers, *if any*, are indicated in bold.

INSTRUCTION	Description
add \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] + REG[\$rt]
addi \$**rt**, \$rs, imm$_{16}$	REG[\$rt] = REG[\$rs] + imm
addiu \$**rt**, \$rs, imm$_{16}$	REG[\$rt] = REG[\$rs] + imm; ! ovrflw
addu \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] + REG[\$rt]; !ovr
sub \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] - REG[\$rt]
subu \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] - REG[\$rt]; !ovr
mul \$**rd**, \$rs, \$rt	REG[\$rd] = (REG[\$rs]*REG[\$rt]}[31:0]
mult \$rs, \$rt	HI|LO = REG[\$rs] * REG[\$rt]
multu \$rs, \$rt	HI|LO = us(REG[\$rs]) * us(REG[\$rt])
div \$rs, \$rt	\$rs/\$rt; HI = quot; LO = rem
divu \$rs, \$rt	us(\$rs)/us(\$rt); HI = quot; LO = rem
and \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] & REG[\$rt]
andi \$**rt**, \$rs, imm$_{16z}$	REG[\$rt] = REG[\$rs] & imm
lui \$**rt**, imm$_{16z}$	REG[\$rt][31:16]=imm;[15:0]=0
nor \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] NOR REG[\$rt]
or \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] OR REG[\$rt]
ori \$**rt**, \$rs, imm$_{16z}$	REG[\$rt] = REG[\$rs] OR imm
xor \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] ^ REG[\$rt]
xori \$**rt**, \$rs, imm$_{16z}$	REG[\$rt] = REG[\$rs] ^ imm
sll \$**rd**, \$rs, SHAMT	REG[\$rd] = REG[\$rs] << SHAMT
sra \$**rd**, \$rs, SHAMT	REG[\$rd] = REG[\$rs] >>> SHAMT
srl \$**rd**, \$rs, SHAMT	REG[\$rd] = REG[\$rs] >> SHAMT
lb \$**rt**, imm$_{16}$(\$rs)	Load byte from MEM[REG[\$rs]+imm]
lbu \$**rt**, imm$_{16}$(\$rs)	Load us(byte) from MEM[REG[\$rs]+imm]
lw \$**rt**, imm$_{16}$(\$rs)	Load 32bits from MEM[REG[\$rs]+imm]
sb \$rt, imm$_{16}$(\$rs)	Store REG[\$rt][7:0] to MEM[REG[\$rs]+imm]
sw \$rt, imm$_{16}$(\$rs)	Store REG[\$rt]to MEM[REG[\$rs]+imm]
slt \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] < REG[\$rt]
slti \$**rt**, \$rs, imm$_{16}$	REG[\$rt] = REG[\$rs] < REG[\$rt]
sltiu \$**rt**, \$rs, imm$_{16}$	REG[\$rt] = REG[\$rs] us(<) imm
sltu \$**rd**, \$rs, \$rt	REG[\$rd] = REG[\$rs] us(<) REG[\$rt]
beq \$rs, \$rt, imm$_{16}$	Branch to label if REG[\$rs]=REG[\$rt]
bne \$rs, \$rt, imm$_{16}$	Branch to label if REG[\$rs]!=REG[\$rt]
j imm$_{26z}$	Jump to label
jal imm$_{26z}$	Jump to label and set \$31 = ret. addr
-jalr \$rs, \$rd	Jump to REG[\$rs] and set \$rd to ret
jr \$rs	Jump to REG[\$rs]
ll \$**rt**, imm$_{16}$(\$rs)	As LW, but set linked status for line

sc $**rt**, imm$_{16}$($rs)	As SW, **but set $rt to 1 if succeeded, 0 if failed;**

$zero	0	Constant 0
$at	1	Reserved for assembler
$v0, $v1	2, 3	Function return values
$a0 - $a3	4 – 7	Function argument values
$t0 - $t7	8 – 15	Temporary (caller saved)
$s0 - $s7	16 – 23	Temporary (callee saved)
$t8, $t9	24, 25	Temporary (caller saved)
$k0, $k1	26, 27	Reserved for OS Kernel
$gp	28	Pointer to Global Area
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address