Name: _____

Write your name and computing ID above. Write your computing ID at the top of each page in case pages get separated. Sign the honor pledge below.

Generally, we will not answer questions about the exam during the exam time. If you think a question is unclear and requires additional information to answer, please explain how in your answer. For multiple choice questions, write a * next to the relevant option(s) along with your explanation.

On my honor as a student I have neither given nor received aid on this exam.

_____

1. Consider the following program that uses the pthreads API:

```
1  void *f1(void *ignored_arg) {
2      write(STDOUT_FILENO, "A", 1);
3      /* (1) */
4      return NULL;
5  }
6
7  int main() {
8      pthread_t t;
9      pthread_create(&t, NULL, f1, NULL);
10     write(STDOUT_FILENO, "B", 1);
11     /* (2) */
12     write(STDOUT_FILENO, "C", 1);
13     pthread_join(t, &p);
14     return 0;
15 }
```

(Assume all needed header files are `#included`.)

(a) (12 points) Suppose the output of this snippet is `BAC`.

If this snippet ran on a single-core processor when this happened and no other programs were running, complete the following table of exceptions that likely occurred, including which line of the above code was active, their type and whether a context switch occurred while handling them. For exceptions that occur as a result of `f1` or `main` returning, indicate line 5 or 15, respectively.

**You may not need all rows of the table.** If there are multiple plausible sequences of exceptions, we will accept any one that is consistent with output given. *not all correct answers are shown below; note that a correct sequence needs to explain how line 2 executed in between lines 10 and 12; while we preferred that the sequence of exceptions was listed in the order of occurence, that was not actually required in the question*

| line # | | cause (what triggered exception) | | | | |
|---|---|---|---|---|---|---|
| 9 | | ■ system call | ○ I/O device | ○ timer | ○ other | |
| 10 | | ■ system call | ○ I/O device | ○ timer | ○ other | |
| 10 | | ○ system call | ○ I/O device | ■ timer | ○ other | |
| 2 | | ■ system call | ○ I/O device | ○ timer | ○ other | |
| 5 | | ■ system call | ○ I/O device | ○ timer | ○ other | |
| 12 | | ■ system call | ○ I/O device | ○ timer | ○ other | |
| 14 | *no deduction for omitting* | ■ system call | ○ I/O device | ○ timer | ○ other | |
| | | ○ system call | ○ I/O device | ○ timer | ○ other | |
| | | ○ system call | ○ I/O device | ○ timer | ○ other | |
| | | ○ system call | ○ I/O device | ○ timer | ○ other | |

(b) (4 points) Consider modifying the code as follows:

- add a global `pthread_barrier_t barrier`;
- initialize `barrier` initialized it at the beginning of `main()` with `pthread_barrier_init(&barrier, NULL, 2);`; and
- insert `pthread_barrier_wait(&barrier);` in place of the comments with `(1)` and `(2)`

After these changes, give an example of an output that would **not** be possible, but would have been possible before these changes.

BCA (only possible answer)

2. Suppose a chat program works by having a single server that multiple chat clients (run on user's own machines) connect to using TCP. Each client maintains an permanent connection to the chat server, and uses this connection messages for other users to and through the chat server.

   To send a chat message "Hello" from user A to user B, user A's chat client sends the following to the chat server over TCP:

   ```
   SENDTO B: Hello
   ```

   Then the chat server sends the following to user B over TCP:

   ```
   FROM A: Hello
   ```

   Assume user A and user B are both connected to the same local wifi network. This local wifi network is connected to an ISP's network via wired Ethernet via a router. The ISP's network is then connected to a datacenter network via another router over wired Ethernet. On that datacenter network, the chat server is connected via wired Ethernet.

(a) (6 points) Consider A sending 'Hello' to B using the chat server.

To accomplish this, multiple messages will likely be sent over the **local wifi network**, including some acknowledgments. In the table below, identify the sender, receiver of each message sent over the wifi network and whether or not the message is an acknowledgment.

Assume no messages are lost and resent and that data is not split across multiple messages due to message size limitations.

(You may not need all lines.)

| | from | to | ack? |
|---|---|---|---|
| 1. | A | local router / chat server (either acceptable) | ☐ |
| 2. | local router / chat server (either acceptable) | A | ■ |
| 3. | local router / chat server (either acceptable) | B | ☐ |
| 4. | B | local router / chat server (either acceptable) | ■ |
| 5. | | | ☐ |
| 6. | | | ☐ |

(b) (4 points) Suppose instead of having messages go through the remote server, one wanted user A and user B's clients to communicate directly with each other, avoiding having the server handle the messages. To enable this, it would be useful if _____. **Select all that apply.**

- ☐ the clients communicated with the chat server using UDP instead of TCP *does not really change anything*
- ■ the clients supported multiple TCP connections at a time *to have TCP connection between user A and user B's clients while also still having one to the server or other users*
- ☐ the chat clients sent two copies of each message *doesnot really change anything*
- ■ the chat server sent the clients the IP addresses of the other clients *to send messages directly from user A to user B*

(c) To prevent the chat server from interfering with user A and user B's communications, the chat clients could use cryptography: in advance, user A and user B receive public keys from each other (for both encryption/decryption and signing/signature verification). Then, when sending a message from user A to user B, user A takes the text of the message, encrypts it with user B's public encryption key, then signs the encrypted message text and a timestamp with user A's private signing key. When user B is sending a message back to user A, they do the same, except swapping user A and user B's keys.

i. (4 points) It is important that user A and user B receive each other's public keys securely. Assuming they both trust a common certificate authority, one way they can do this is with *certificates*. (A trusted *certificate authority* signs a message containing a public key and a corresponding a identity. The message with the signature is called a *certificate*.)
When user A obtained a certificate for their public keys signed by a certificate authority, they would _____, then use the private keys corresponding to the public keys in the certificate to communicate further with user B.

- ○ create a new message containing their public keys, the certificate authority's public keys, and then generate a signature using user A's private key over this new message, and send the message and signature to user B
- ○ edit the identity information in the certificate from specifying user A to specify user B instead, then send the modified copy to user B
- ■ send a copy of the certificate to user B without modifying it
- ○ verify the signature on the certificate, then copy the public keys and identity information in the certificate and, send that copy to user B along with a new signature created using user A's private key
- ○ none of these — explain: _____

ii. (4 points) Assuming A and B both obtain each other's public keys securely and A and B communicate through the chat server, what can the chat server do in spite of the cryptography? **Select all that apply.**

- ☐ the chat server can modify the text of a message from user A to user B without user B being able to detect that it was modified *signatures allow detection*
- ■ the chat server can discard some but not all messages from user A to user B without user B being able to detect that messages were missing *nothing in scheme tracks how many messages were actually sent, etc.*
- ☐ the chat server can record a message and resend it much later without user B being able to detect that user A did not intend to send the message then *prevented by timestamp*
- ☐ the chat server can take a message intended to be sent from user A to user B and send it back to user A without user A as if it were a message from user B being able to know that it wasn't sent by user B *encryption/signature would be using wrong key*

3. (8 points) Consider the following code:

```
unsigned char table1[4096];
...
unsigned char table2[4096];
int TableLookup(unsigned int x, unsigned int y) {
    if (x < 4096 && y < 2048) {
        return table2[table1[x] + y];
    } else {
        return 0;
    }
}
```

Suppose this code is run as part of a Spectre-style attack and:

- no virtual memory is used (so all addresses are physical);
- the system running it has a 16384-byte ($2^{14}$ byte) direct-mapped data cache with 256-byte blocks;
- `table1` is located at address `0x10 0000` (so `table1[0]`'s address has set index 0, cache offset 0)
- `table2` is located at address `0x12 0000` (so `table2[0]`'s address also has set index 0, cache offset 0)
- the processor's branch predictor predicts `x < 4096 && y < 2048` to be true
- an attacker sets x to `0x10 0000`, so when `table1[x]` is partially run as a result of branch prediction, `table1[x]` refers to the value at address `0x20 0000`

After setting `x` as above and `y` to 0, the attacker discovers that the access to `table1` evicts from cache set index 0 and the access to `table2` evicts from cache set index 0*was 4 as printed*. They also discover that when setting `x` as above and `y` to 128 that the access to `table1` still evicts from cache set index 0 and the access to `table2` evicts from cache set index 1*was 5 as printed*. Based on these results, what is a possible value of memory at `0x20 0000`?

You may leave your answer as an unsimplified arithmetic expression.

**Solution:** as printed: not possible given unsigned char or 256 * 4 + 128 through 256 * 4 + 255; half credit for 256 * 4; with corrections above: 128 through 255 inclusive

4. For the following questions, consider the assembly function foo (with //-comments describing each instruction):

```
.global foo
foo:
   movq    (%rsi), %rax  // RAX <- MEMORY[RSI]
   movq    (%rdi), %rcx  // RCX <- MEMORY[RDI]
   cmpq    %rcx, %rax    // compare RCX and RAX
   je      end_foo       // if (RCX==RAX) goto end_foo
   addq    %rcx, %rax    // RAX <- RCX + RAX
   movq    %rax, (%rdi)  // MEMORY[RDI] <- RAX
   addq    $8, %rsi      // RSI <- RSI + 8
   jmp     foo           // goto foo
end_foo:
   ret                   // return from function
```

*as printed on the exam, the comment for the addq of 8 used RDI instead of RSI* which could have been generated from the C code:

```
void foo(long *a, long *b) {
    while (*a != *b) {
        *a += *b;
        b += 1;
    }
}
```

(a) Suppose the above assembly snippet runs on a *seven-stage* pipelined processor where the pipeline stages are:
fetch; decode (including reading registers); execute part 1; execute part 2; memory part 1; memory part 2; and writeback.

Assume:

- the processor uses forwarding when possible without dramatically increasing cycle time, and
- the split execute and memory stages
  - perform the same operations as the single execute and memory stages in the 5-stage processor we discussed
  - need their inputs near the beginning of the part 1 stage, and
  - produce their outputs near the end of the part 2 stages

i. (6 points) Given this design, in an iteration of the loop of `foo`, the `cmpq %rcx, %rax` instruction will complete its writeback stage _____ cycles after the `movq (%rsi), %rax` instruction. Show any work.

**Solution:** 5 (align decode stage of cmpq with memory 2 stage of movq from %rdi)

ii. (4 points) (Information from previous page reproduced here for convenience.
Assembly snippet:

```
.global foo
foo:
   movq    (%rsi), %rax  // RAX <- MEMORY[RSI]
   movq    (%rdi), %rcx  // RCX <- MEMORY[RDI]
   cmpq    %rcx, %rax    // compare RCX and RAX
   je      end_foo       // if (RCX==RAX) goto end_foo
   addq    %rcx, %rax    // RAX <- RCX + RAX
   movq    %rax, (%rdi)  // MEMORY[RDI] <- RAX
   addq    $8, %rsi      // RSI <- RSI + 8
   jmp     foo           // goto foo
end_foo:
   ret                   // return from function
```

Pipeline design: fetch, decode, execute part 1, execute part 2, memory part 1, memory part 2, writeback.)
Given the seven-stage design , during the **second** iteration of the loop in foo, the `movq (%rsi), %rax` will obtain the value of %rsi by _____.

- ■ reading it from the register file *comment as written on exam*
- ○ reading it from the register file and adding 8 in the movq's execute stages
- ○ forwarding it from the previous iteration's `movq (%rsi), %rax`
- ■ forwarding it from the previous iteration's `addq $8, %rsi` *assembly as written*
- ○ forwarding it from the previous iteration's `jmp foo`
- ○ forwarding it from some instruction run before `foo` was called or reading it from the register file, depending on the code that calls foo
- ○ something else, explain: _____

(b) (7 points) Suppose the above code executes on an out-of-order processor similar to what we described in lecture.

Based on the dependencies between instructions and assuming perfect perfect branch prediction, an out-of-order processor could perform the addition calculation for `addq %rcx, %rax` at the same time as _____ performs its addition calculation, data cache access, or comparison.

Not all possible overlapping instructions are listed. Ignore any restrictions on what instructions can execute at the same time that aren't due to one instruction needing the results, directly or indirectly, of another to execute. *This question was miskeyed before approx 6p 15 December. Sorry for the error*
**Select all that apply.**

- ■ `cmpq %rcx, %rax` from the previous iteration of the loop *Note that thee addq into %rax's result affects the next %rax through a store and load to memory. Because of branch prediction, it's not a problem that we haven't determined whether the loop continues or not yet for certain. It's not a problem that the %rax, %rcx registers are reused because of register renaming — they'll actually be able to use independent physical registers (assuming enough available, renaming happens fast enough, etc.)*
- ■ `cmpq %rcx, %rax` from the same iteration of the loop *values of %rax, %rcx are computed before addq runs, so don't need to wait to addq to complete to get them, and addq doesn't use the results of the cmpq itself (though the cmpq is used to check that the branch prediction that ran the addq was correct)*
- ☐ `cmpq %rcx, %rax` from the next iteration of the loop *since need to read new value of %rax*
- ☐ `addq %rcx, %rax` from the previous iteration of the loop
- ☐ `addq %rcx, %rax` from the next iteration of the loop

- ☐ `movq %rax, (%rdi)` from the previous iteration of the loop *since stored value needs to be loaded again*
- ☐ `movq %rax, (%rdi)` from the same iteration of the loop*requires result of addq to store in data cache*

5. Suppose a system has:
   - a 2048-byte ($2^{11}$ byte) 2-way L1 data cache with 16-byte blocks and an LRU replacement policy
   - a 16384-byte ($2^{14}$ byte) *corrected — wrote 8192/$2^{13}$ on exam as printed* 2-way L2 data cache with 16-byte blocks and an LRU replacement policy
   - caches that always use physical addresses (any translation from virtual to physical addresses occurs before a cache acces)
   - 4096-byte ($2^{12}$ byte) pages
   - 3-level page tables with 1024 entries ($2^{10}$ entries) in tables at each level (so 10 bits of the virtual page number are used for each lookup in a page table)
   - page table entries stored as 4-byte integer, where the least significant bit represents the valid bit, and the most significant 20 bits represent the physical page number
   - a 4-entry fully-associative (4-way) data TLB with an LRU replacement policy

   Assume that the system ensures that data cached in the L1 data cache is also cached in the L2 data cache.

(a) (4 points) How large in bits are virtual addresses on this system? (You may leave your answer as an unsimplified arithmetic expression.)

10 * 3 + 12

(b) (4 points) If `first_level_pte` represents the **value** of a valid first-level page table entry (as a 32-bit integer) and `va` represents a virtual address being accessed, then the **address** of the second-level page table entry would be returned by which *one* of the following C snippets, assuming `A`, `B`, `C`, and `D` are appropriate integer constants?

- ○ `return (((first_level_pte & A) >> B) * C) | (va & D)`
- ■ `return (first_level_pte & A) + (((va & B) >> C) * D)` *e.g., A = 0xFFFFF000, B = 0x3FF000, C = 12, D = 4; gives physical address*
- ○ `return (first_level_pte + ((va >> A) * B)) & C`
- ○ `unsigned *p = (unsigned *)first_level_pte;return &p[(va & A) >> B]`

(c) (4 points) If a process on this system has assigned to it:
   - 1 first-level page table
   - 1 second-level page tables
   - 2 third-level page tables

   then what is the maximum number of distinct bytes of physical memory it can access successfully without additional page tables being allocated for it? (You may leave your answer as an unsimplified arithmetic expression.)

4096 * 1024 * 2 (each third-level table points up to 1024 data pages that each contain 4096 bytes)

(d) (Information reproduced from previous page for convenience:

- a 2048-byte ($2^{11}$ byte) 2-way L1 data cache with 16-byte blocks and an LRU replacement policy
- a 16384-byte ($2^{14}$ byte) *corrected — wrote 8192/$2^{13}$ on exam as printed* 2-way L2 data cache with 16-byte blocks and an LRU replacement policy
- caches that always use physical addresses (any translation from virtual to physical addresses occurs before a cache acces)
- 4096-byte ($2^{12}$ byte) pages
- 3-level page tables with 1024 entries ($2^{10}$ entries) in tables at each level (so 10 bits of the virtual page number are used for each lookup in a page table)
- page table entries stored as 4-byte integer, where the least significant bit represents the valid bit, and the most significant 20 bits represent the physical page number
- a 4-entry fully-associative (4-way) data TLB with an LRU replacement policy

)

Suppose a program accesses 1 byte from virtual address 0x12348, and that adddress corresponds to physical address 0x45348.

i. (4 points) Immediately after the access to 0x12348, it's possible for a following access to be a TLB hit but a miss in both the L1 and L2 data caches. Give an example of a *virtual* address which would have this property.
0x12xxx where xxx is not between 340 and 34f. Should have been more explicit that the TLB hit was to the sa

ii. (6 points) After the access to 0x12348, it's possible after two more accesses for the cache block containing 0x12348 to be evicted from the L1 data cache. Give an example of *virtual* addresses these two accesses could have.

> **Solution:** two addresses in different cache blocks (than 0x12340 and each other) ending in [37bf]4[0-f] when written in hex

iii. (6 points) In the L2 cache, block offsets are 4 bits and set indices are 9 bits.
The last 4 bits of 0x12348 are 1000 and the previous 9 bits are 0 0011 0100 (or 0x034 in hexadecimal).
In spite of this it's possible that when the L2 cache is accessed as part of reading 0x12348, that the cache accesses the set with 1 0011 0100 (or 0x134 in hexadecimal).
Briefly explain what must be true for this to happen.

> **Solution:** This question was dropped. With corrections above — either physical address 0x12348 maps to has a 1 in the 13th least significant bit (unlike virtual addresss) or L1 cache has a writeback policy and an eviction was triggered from the L1 cache which had that set index as interpreted for the L2 cache. Without corrections above, there is contradictory information about the L2 set index size, and if the set index is actually 8 bits, then the physical address explanation is not possible.

6. (18 points) Suppose we are implementing an office-hour queue system. In this system, we run one thread to represent each student and each teaching assistant. In our course, teaching assistants specialize in particular types of questions, so when students add themselves to the queue, they indicate the question type they have and teaching assistants indicate the type of question they can handle.

   We represent this with an API as follows:

   - `TAID WaitForTA(StudentID student_id, QuestionType qType)` — wait for a TA to be available to handle a question of type QuestionType
   - `StudentID WaitForStudent(TAID ta_id, QuestionType qType)` — wait for a student to have a question of one of the question types in qTypes

   On the next page, complete the code where `AddToLinkedList` and `RemoveFromLinkedList` are functions that add a WaitingStudent struct to or remove from one from the linked list represented with the pointers head and tail, updating the head and tail pointers as necessary.

   (There are three blanks to fill in.)

```
pthread_mutex_t lock; pthread_cond_t ta_cv;
struct WaitingStudent {
    StudentID id; TAID ta;
    QuestionType qType;
    pthread_cond_t cv;
    int waiting;

    struct WaitingStudent *next;
    struct WaitingStudent *prev;
};
struct WaitingStudent *head;
struct WaitingStudent *tail;

void WaitForTA(StudentID student_id, QuestionType qType) {
    struct WaitingStudent student;
    student.student_id = id;
    student.qType = qType;
    student.waiting = 1;
    pthread_cond_init(&student.cv);
    pthread_mutex_lock(&lock);
    AddToLinkedList(&student, &head, &tail);
    pthread_cond_broadcast(&cv);
    while (student.waiting) {
        pthread_cond_wait(&student.cv, &lock);
    }
    pthread_cond_destroy(&student.cv);
    pthread_mutex_unlock(&lock);
    return student.ta;
}

struct WaitingStudent *FindStudentMatching(QuestionType qType) {
    struct WaitingStudent *student_pointer;
    student_pointer = head;
    while (student_pointer) {
```

```
        if (student_pointer->qType == qType) {
            return student_pointer;
        }
        student_pointer = student_pointer->next;
    }
    return NULL;
}

StudentID WaitForStudent(TAId ta_id, QuestionType qType) {
    pthread_mutex_lock(&lock);
    while (FindStudentMatching(qType) == NULL) {
        pthread_cond_wait(&ta_cv, &lock);
    }
    struct WaitingStudent *student_pointer;
    student_pointer = FindStudentMatching(qType);
    RemoveFromLinkedlist(student_pointer, &head, &tail);
    student_pointer->waiting = 0; student_pointer->ta = ta_id;
    StudentID student_id = student_pointer->student_id;
    pthread_cond_signal(&student_pointer->cv);
    pthread_mutex_unlock(&lock);
    return student_id;
}
```

7. (4 points) An executable file '`1.exe`' and a text file '`2.txt`' have access control lists (ACLs) as follows:

ACL for 1.exe
```
user:foo:rwx
user:bar:r-x
group:quux:r-x
other:---
```

ACL for 2.txt
```
user:foo:rw-
user:bar:rw-
group:baz:r--
other:---
```

In these ACLs:

- '`r`' represents read permission; '`w`' write permission; and '`x`' execute permission;
- a user line takes precedence over any group line, and the other line only applies when no user or group line matches

Based on these access control lists, which of the following are true? **Select all that apply.**

- ☐ at most three distinct users can run '1.exe' from their shells *could have more users whose processes run in group quux*
- ■ if a program can modify '1.exe' by overwriting it, then it can also modify '2.txt' by overwriting it *programs that overwrite 1.exe must be running as user foo (or superuser)*
- ☐ when the executable '1.exe' is run, it will be able to read '2.txt' *user ID executable runs as is not based on owner/etc. normally, and 1.exe could be run by process in group quux that is not in group baz or running as user foo or bar*
- ☐ if a process is not running as the user 'foo' or the user 'bar', then it can only read one of '1.exe' and '2.txt' *same process could be in group quux and baz; or could be running as superuser*

8. Consider the following program that uses the POSIX API:

```
1  int main() {
2      pid_t p;
3      int fds[2];
4      pipe(fds);
5      p = fork();
6      if (p == 0) {
7          dup2(fds[1], STDOUT_FILENO);
8          close(fds[1]); close(fds[0]);
9          char *args[] = {"/bin/mystery", NULL};
10         execv("/bin/mystery", args);
11     } else {
12         close(fds[1]);
13         char c;
14         while (read(fds[0], &c, 1) == 1) {
15             if (c != 'A') {
16                 write(STDOUT_FILENO, &c, 1);
17             }
18         }
19         waitpid(p, NULL, 0);
20     }
21 }
```

(a) (8 points) Assuming `write`, `fork`, `read`, `pipe`, `waitpid`, and `execv` do not fail, **briefly** describe what the above code outputs to stdout. (The program's output depends on what the program `/bin/mystery` does.)

**Solution:** the output of /bin/mystery with any capital As omitted

(b) (4 points) If the `waitpid()` were moved just before the while loop, then _____ (even though it did not without this change). *was not marked select all that apply properly on exam as printed* **Select all that apply.**

- ☐ the execv call could fail
- ■ the program could hang at the waitpid() call
- ☐ the program could hang at the read() call
- ☐ the program could segfault

9. (5 points) Suppose a Makefile that contains the following:

```
all: application main.o utility.o

main.o: main.c utility.h
        clang -g -Og -Wall -c main.c -o main.o

utility.o: utility.c utility.h
        clang -g -Og -Wall -c utility.c -o utility.o

application: main.c utility.c
        clang -g -Og -Wall -o application main.o utility.o
```

This Makefile erroneously has the wrong dependencies for the `application` rule.

Which are problems that this error could cause to occur while running the command `make all`? **Select all that apply.**

- ■ although `utility.h` was modified, make does not rebuild `application`
- ☐ although `utility.h` was modified, make does not rebuild `main.o`
- ☐ although `utility.c` was modified, make does not rebuild `utility.o` *dropped — on exam as printed "not" was not written*
- ☐ there is a file not found error for `main.c` *if this occurs, would also occur with dependencies fixed*
- ■ there is a file not found error for `main.o` *application built before main.o built since no dependency to tell make they need to happen in a particular order*