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. (12 points) Consider a program that is built in a combination of a custom programming language and C. To support the custom programming language, the program's source code includes a compiler for that programming language. To build the program requires building that compiler, then using that compiler to build an object file, then linking that object file with others to build the final program.

   To build everything from scratch uses the following commands:

   ```
   clang -c compiler.c
   clang -c main.c
   clang -o compiler compiler.o
   ./compiler -c functions.customlang
   clang -o program main.o functions.o
   ```

   Fill in the blanks to complete the following Makefile to automate rebuilding the program. (You may not need all lines.)

   ```
   all: program

   main.o: main.c
       # erroneously had util.c here in the original exam
       # also accepted answers that did something to "fix" this issue below
       clang -c main.c

   compiler.o: compiler.c
       clang -c compiler.c

   compiler: compiler.o
       clang -o compiler compiler.o

   functions.o: functions.customlang compiler
       ./compiler -c functions.customlang

   program: main.o functions.o
       clang -o program main.o functions.o

   .PHONY: all
   ```

2. Consider the following C program that uses the POSIX API. (Assume all needed header files are included.)

```
1  pid_t p;
2  int g = 0;
3  void handler(int ignored) {
4      printf("%d:%d:%d\n", (int) p, (int) getpid(), g);
5      g += 1;
6  }
7
8  int main() {
9      struct sigaction sa;
10     memset(&sa, 0, sizeof(sa));
11     sa.sa_handler = handler;
12     sa.sa_flags = SA_RESTART;
13     sigaction(SIGUSR1, &sa, NULL);
14     p = getpid();
15     pid_t q = fork();
16     if (q == 0) {
17         kill(SIGUSR1, p);
18         while (1) pause();
19     } else {
20         kill(SIGUSR1, q);
21         pid_t r = fork();
22         if (r == 0) {
23             while (1) pause();
24         } else {
25             kill(SIGUSR1, r);
26             exit(0);
27         }
28     }
29 }
```

(SA_RESTART makes it so signal handlers that interrupt a system call restart that system call when they finish.)

(a) (8 points) Which of the following is a possible output of the above program? **Select all that apply.**

100:100:0	100:101:0	100:101:0	100:100:0
■ 100:101:0	■ 100:102:0	■ 100:100:0	☐ 100:100:1
100:102:1	100:100:0	100:102:1	100:100:2

3. (6 points) Consider the following C snippet that uses the POSIX API:

```
printf("Enter letter: ");
char c = getchar();
FILE *fh = fopen("dir/out.txt", "w");
if (!fh) handle_error();
fwrite(fh, &c, 1);
fclose(fh);
```

Suppose the program gets a permission denied error when it tries to open `out.txt`. Which of the following changes might fix this issue? **Select all that apply.**

- ☐ adding a new signal handler (and registering it with sigaction())
- ■ adding an appropriate entry to the access control list of the directory `dir`
- ■ adding an appropriate entry to the access control list of a file called `out.txt`, if one exists in `dir`
- ☐ adding an appropriate entry to the access control list of the program's executable
- ☐ changing the program to open for reading and writing (like with `"w+"`) instead of just for writing
- ☐ making the program dynamically linked instead of statically linked

4. Consider a system with:
   - 16384 (2^{14}) -byte pages, with 2-level page tables, where page tables at each level contain 2048 (2^{11}) entries of 8 bytes each
   - an 8-way, 64-entry TLB
   - 36-bit virtual addresses
   - 48-bit physical addresses
   - a 4-way 4096 (2^{12}) byte L1 data cache with 128 (2^{7})-byte blocks that uses physical addresses, an LRU replacement policy, and write-back and write-allocate policies
   - a 2-way 4096 (2^{12}) byte L1 instruction cache with 128-byte block that uses physical addresses and an LRU replacement policy
   - an 8-way 1048576 (2^{20}) byte L2 cache, with 128-byte blocks shared between instructions and data, that uses physical address, and has an LRU replacement policy, and write-back and write-allocate policies

(a) (4 points) How many bits are TLB tags? You may leave your answer as an unsimplified arithmetic expression.

Solution: 11+11 = 22 bit virtual page numbers - 3 index bits = 19 tag bits

(b) (6 points) **Fill in the blanks.** You may leave your answers as unsimplified arithmetic expressions.

When performing a page table lookup for virtual address `0x0 A123 4567`, if the first-level page table starts at `0x40 0000` and the second-level page table starts at physical address `0x50 0000`.

Then, to find the address of the second-level page table entry, we would take bits 14 through 24 (inclusive) of the virtual address (counting the least significant bit as bit 0) and adding 8 times that to 0x500000.

14-bit page offset, two 11-bit VPN parts; we want to use VPN part from the least significant bits for this lookup.

(c) (4 points) How much storage does the L2 cache use for tags? You may write your answer as an unsimplified arithmetic expression.

Solution: tag bits per block = (48 (address size) - 7 (offset bits) - $\log_2$(1048576/128/8) (index bits)) = 48-7-10=31; 1048576/128 (block count) times 31 (tag bits) = 8192 times 31 = 253952

(d) (6 points) Give an example of two different memory addresses that would be stored in the same set in the L1 data cache but not in the L1 instruction cache, or if this is not possible write 'impossible' and explain briefly (at most one sentence).

If you write addresses, you may write them as unsimplified arithmetic expressions if you prefer.

Solution: data cache has 7 offset bits, 3 index bits; instruction cache has 7 offset bits, 4 index bits, so the values need to differ bit 11, but have bits 7-10 be the same

(e) (6 points) Suppose the L1 data cache is initially empty, then the following operations are performed. Complete the following table, identifying how much is read/written from the L2 cache during each access. (The first row is done for you.) *-1 point per wrong, minimum 0*

	address				
operation type	tag	set index	offset	L2 read bytes	L2 write bytes
read 1 byte	`0x2345`	`0x5`	`0x02`	128	0
write 1 byte	`0x2345`	`0x5`	`0x03`	0	0
write 4 byte	`0x1234`	`0x4`	`0x44`	124 (also accept 128)	0
read 1 byte	`0x1234`	`0x4`	`0x48`	0	0
read 8 byte	`0x1043`	`0x4`	`0x00`	128	0
read 4 byte	`0x1044`	`0x4`	`0x04`	128	0

5. (4 points) Alice and Bob have a TCP connection they use to send messages to each other. One of their messages was dropped somewhere along the way and was resent. This all happened without the TCP transport layer ever behaving differently or noticing by failing to see an acknowledgment. Which of the following happened?

   - [■] Alice is connected to the internet through Wi-Fi and this link layer dropped the message, but noticed it and resent the message very quickly *this action by the WiFi layer would be invisible to the the TCP layer. It's true that TCP would be doing redundancy in addition to this, but TCP's redundancy would handle cases the link-by-ink would not.*
   - [○] At the application layer, Bob's browser saw the dropped message and resent the message very quickly *everything the browser accesses goes through the TCP layer, so it would have be involved, violating the conditions of the quesiton.*
   - [○] At the IP network layer, the packet was dropped since it was sent with the wrong destination port and was automatically detected and resent by IP with the corrected port *IP layer doesn't deal with port numbers, though could do things without the TCP layer being involved*
   - [○] The packet could have been dropped and resent at any layer since TCP never notices or changes behavior when packets are dropped. *TCP does react to drops by resending, when it detects that through a timeout.*

6. (12 points) Suppose we run a chat program similar to the signals lab on a shared, single-core system. In this design, two instances of the program run at the same time. They use shared memory to communicate message contents. Whenever a message is sent, one program sends a signal to the other. Then the other outputs that message, then modifies the shared memory region to indicate that it has received the message.

   Suppose two instances of this program are running and 'connected' to each other. Then, one message is sent from one program to the other.

   As part of sending this message, list the exceptions with their causes that must occur. Assume that initially the shared memory region is setup (such that no exceptions will occur because of it being accessed), the program sending the message has requested and but not received input, and some third program is active on the processor.

   Solution: I/O exception(s) for keyboard input; possibly timer interrupt to stop third program; system call to trigger exception, system call to show output

7. Consider the following assembly snippet:

```
1  addq %r9, %r10
2  addq %r9, %r11
3  xorq %r9, %r12
4  xorq %r10, %r11
5  xorq %r11, %r12
```

(a) (8 points) Suppose we executed the above code on a **six-stage** pipelined processor which is like the five-stage processor we discussed in lecture, but the execute stage is split into two stages. The operands for arithmetic operations are needed near the beginning of the first execute stage, and the results are available for forwarding only near the end of the second execute stage.

If the `addq` is fetched during cycle number 1, then the final `xorq` would complete its writeback during cycle _____.

```
1                    1  2  3  4  5  6  7  8  9 10 11
2  addq %r9, %r10    F  D  E1 E2 M  W
3  addq %r9, %r11       F  D  E1 E2 M  W
4  xorq %r9, %r12          F  D  E1 E2 M  W
5  xorq %r10, %r11            F  D  E1 E2 M  W
6  xorq %r11, %r12               F  D  D  E1 E2 M  W
```

Solution: one cycle of stalling between the last two xors; 6 (run first addq) + 4 (four more writebacks) + 1 (stalling) = 11

(b) (12 points) Suppose an out-of-order processor has 2 pipelined ALUs, each of which take 2 cycles for every operation but can accept a new operation each cycle. (Ignore time to load the instructions into the instruction queue, writeback their results or commit, and Assume that the result of an operation is available for forwarding immediately after it completes. that operations cannot start until after all operands are available, and that the initial values of registers are available in advance.) Fill in the table, using the instructions numbers (1–5) above, with how this instruction might be most quickly executed on the ALUs.

cycle #	1	2	3	4	5	6
ALU1 stage1	1	3	4		5	
ALU1 stage2		1	3	4		5
ALU2 stage1	2					
ALU2 stage2		2				

8. (6 points) Consider the following C code (where `...` represents missing code):

```
unsigned int secret_array[4000];
unsigned int mystery;
void MysteryFunction() {
    secret_array[mystery] += 1;
}

void ProbeFunction() {
    unsigned char probe_array[65536];
    for (int i = 0; i < array; ++i) {
        probe_array[i] += 1;
    }
    MysteryFunction();
    ... /* code to time accesses to probe_array */
}
```

Suppose we run this on a system:

- with a 64KByte (2^{16} byte) 2-way data cache with 32-byte cache blocks and an LRU replacement policy and a write-back, write-allocate policy,
- where both `secret_array` and `probe_array` start an address which is a multiple of the cache size, and
- where virtual memory is not in use
- where chars are 1 byte and ints are 4 bytes

By timing acessing to `probe_array` in ProbeFunction where `...` is placed, we can obtain information about the value of `mystery`.

Fill in the blanks below. You may use an unsimplified arithmetic expression for any numbers.

If the value of `mystery` is 102, we would expect accesses to `probe_array` index 384 through 415 *accept any in range* to be slower than accesses to most elements of `probe_array` (when this would not be the case for a majority of other possible values of mystery).

9. Acme Corp receives orders from their wholesale customers using (unwisely) system with custom cryptography (instead of something with standard TLS and the like).

   In this system, each of their customer's has access to Acme's public keys for encryption and digital signatures, and Acme has access to each of their customer's public keys for encryption and digital signatures.

   To make an order, a customer sends a message with the following parts:

   - a submessage #1, individually encrypted to Acme's public key, containing
     - an unique identifier for the order
     - the number of items in the order
   - a digital signature, made using the customer's private key, for submessage #1
   - for each item, a submessage individually encrypted to Acme's public key, containing the name of the item

   Acme Corp replies to the order with an shipping and cost information, with the following parts:

   - a submessage #1, individually encrypted to the customer's public key, containing
     - the customer's identifier for the order
     - the total cost for the order
   - a digital signature, made using Acme's private key, for submessage #1
   - for each item, a submessage individually encrypted to the customer's public key, containing the that item's shipping tracking number

(a) (5 points) Which of the following can an active machine-in-the-middle attacker do by intercepting, changing, or injecting messages without Acme being able to detect it from the messages they receive? **Select all that apply.**

- [x] change which items the customer ordered in an order
- [x] when a customer submits 4 consecutive orders, making it appear that the customer only submitted the first, second, and fourth of those orders
- [ ] when a customer submits exactly one order, making it appear that they submitted two different orders, but requesting the same list of items
- [x] when a customer submits 2 consecutive orders, the first for items A, B, and C, and the second for items D, E, and F, making it appear that they submitted two orders, the first for items A, B, and C, and the second for items A, B, and E.
- [ ] learn the order identifier the customer chose, even if the customer selected it randomly from a large set of possibilities

(b) (4 points) Assuming they know each order's identifier (perhaps because the customer uses idenitfier 1, then 2, then 3, etc.) and the items ordered, which of the following can an active machine-in-the-middle attacker do by intercepting, changing, or injecting messages without customer being able to detect it from the messages they receive? **Select all that apply.**

- [ ] change the total cost the customer's receives to something that Acme did not send
- [ ] change the total cost the customer's receives to the cost of an earlier order
- [ ] learn the shipping tracking numbers for the customer's items
- [x] change the shipping tracking numbers the customer receives to something that Acme did not send

10. Consider the following C code:

```
1  pthread_mutex_t lock;
2  int count;
3  pthread_cond_t count_increase_cv; pthread_cond_t count_decrease_cv;
```

```
4  void WaitForCountAndAdd(int start_count, int add_to_count) {
5      pthread_mutex_lock(&lock);
6      while (count != start_count) {
7          while (count < start_count) {
8              pthread_cond_wait(&count_increase_cv, &lock);
9          }
10         while (count > start_count) {
11             pthread_cond_wait(&count_decrease_cv, &lock);
12         }
13     }
14     count += add_to_count;
15     if (add_to_count > 0) {
16         pthread_cond_broadcast(&count_increase_cv);
17     } else {
18         pthread_cond_broadcast(&count_decrease_cv);
19     }
20     pthread_mutex_unlock(&lock);
21 }
```

(a) (10 points) Suppose the count is 0, and then:

- thread A calls `WaitForCountAndAdd(2, 1)`
- thread B calls `WaitForCountAndAdd(3, -1)`
- thread C calls `WaitForCountAndAdd(0, 3)`

It is possible for thread A to end up calling pthread_cond_wait for both `count_increase_cv` and `count_decrease_cv`. Complete the following timeline to show an example of how that could happen, indicating the line numbers of each call to pthread functions *added for late exams:* and including all pthread function calls until the WaitForCountAndAdd calls return. (You may not need all lines. We include the name of the function being run, but you do not need to do this.)

thread A	thread B	thread C
5 (lock)	—	—
8 (wait)	—	—
—	5 (lock)	—
—	8 (wait)	—
—	—	5 (lock)
—	—	16 (broadcast)
—	—	20 (unlock)
11 (wait)		
	18 (broadcast)	
	20 (unlock)	
16 (broadcast)		
20 (unlock)		

(b) (6 points) *original phrasing was: Suppose we have 5 threads call `WaitForCountAndAdd(x, 1)` for all `x` in 0 through 4, inclusive, with an initial count value of 0. More explicit phrasing to address an unintended interpretation we saw on the exam:* Suppose 5 threads call `WaitForCountAndAdd(x, 1)`, each using a different value of `x` ranging from 0 through 4 inclusive, and the initial value of count is 0. **Fill in the blanks.** (You may leave your answers as unsimplified arithmetic expressions.)

If there are no spurious wakeups, then the the minimum number of `pthread_cond_wait` calls is 0 and the maximum is 4+3+2+1 = 10