University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2017 Instructors: Randy Katz, Krste Asanovic 2017-12-14

CS61C FINAL

After the exam, indicate on the line above where you fall in the emotion spectrum between "sad" & "smiley"...

Last Name	
First Name	
Student ID Number	
CS61C Login	**cs61c-**
*The name of your **SECTION** TA and time*	
Name of the person to your LEFT	
Name of the person to your RIGHT	
*All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who have not taken it yet. **(please sign)***	

Instructions (Read Me!)

- This booklet contains 19 numbered pages including the cover page and an answer sheet.
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats & headphones. Place your backpacks, laptops and jackets under your seat.
- You have 170 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use three handwritten 8.5"x11" page (front and back) crib sheet in addition to the RISC-V Green Sheet, which we will provide.
- **Please write all of your answers on the answer sheet provided. Only the answer sheet will be graded.** When we provide a blank, please fit your answer within the space provided. **When we provide a multiple choice question, please bubble in your choice(s) on the answer sheet. You will lose credit if you fail to do so.**

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Total
Points Possible														TBD

Q1: A bit of C here and there

1. What, if anything, is wrong with this code? Write your answer in 10 words or less. (Hint: Recall Project 2-1 and consider *all* possible values of m.)

   ```
   int floor_divide_by_two(int m) { return m >> 1; }
   ```

   Implementation doesn't work with negative numbers.

2. Please fill in `power_of_16` to calculate whether the given integer is a power of 16.

   Be sure to only use bitwise operators, == and != and up to 1 subtraction. You may introduce constants but not any new variables.

   Return 0 if it is not a power of 16, or 1 if it is. (**Note: 0 is not a power of 16**).

   ```
   // sizeof(int) == 4
   int power_of_16(int m) {
         # check sign bit & # of 1-bit in binary representation
         if (0 != ((m >> 31) & 1) || (m & (m-1)) != 0) {
               return 0;
         } else {
               return m != 0 & (m & 0xEEEEEEEE) == 0;
         }
   }
   ```

Q2: MapReduce

We want to provide more insight into the Yelp dataset used in Project 4. **Fill in the blanks for the Python code in the answer sheet** to return a review with the highest word count per star label using Spark (Multiple reviews could have the same word count. If so, the code should return just one of them). A sample of input and output is given:

Sample Input (in yelp_data.txt): (review_id, num_stars, review_text)

```
7xGHiLP1vAaGmX6srC_XXw 5 Great place
KpRwKYyQ93ypyDSdA7IXfw 3 Okay place
ZWlXWc9LHPLiOksrp-enyw 1 Bad place fine service
```

Sample Output: (num_stars, (review_id, num_words))

```
(5, (7xGHiLP1vAaGmX6srC_XXw, 2))
(3, (KpRwKYyQ93ypyDSdA7IXfw, 2))
(1, (ZWlXWc9LHPLiOksrp-enyw, 4))
```

```
def parseLine(line):
    tokens = line.split(" ")
    review_text = tokens[2:]
    return (tokens[1], (tokens[0], len(review_text)))


def findMax(value1, value2):
    if value1[1] > value2[1]:
        return value1
    else:
        return value2

if __name__ == '__main__':
    # Assume you have a Spark Context set up
    sc = SparkContext()
    raw_reviews = sc.textFile("yelp_data.txt")
    results = raw_reviews\
                .map(parseLine).reduceByKey(findMax)
```

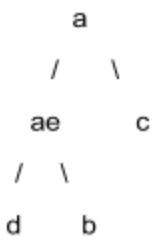
Q3: Now you C me, now you don't

Let's build a basic min-heap. Recall that a min-heap is similar to a binary tree where each child is greater than or equal to its parent. We will be using a single dimensional array to represent our min-heap, which other than `create`, supports `insert` and `free_heap` only.

Here's an example. Given the array,

```
char* arr[] = { "d", "b", "c", "ae", "a" };
```

Our min-heap will transform this array into
`["a", "ae", "c", "d", "b"]`



Which, diagrammatically, would be represented as on the right:

Here's the data structure we will be using, with its corresponding create function.

```
#define INIT_CAP 10

struct min_heap {

      char** data;

      size_t size; // current number of elements in heap

      size_t capacity; // max number of elements heap can handle

};

struct min_heap* create(char** arr, size_t size) {
      struct min_heap* m = malloc(sizeof(struct min_heap));
      m->size = m->capacity = 0;
      for(size_t i = 0; i < size; i++)
            insert(m, arr[i]);
      return m;
}
```

Please fill in the blanks to complete the functionality for the `free_heap` and `insert` functions on your **answer sheet**. As a reminder, to insert an element into a min heap, you will insert the element at the back of the array and then bubble up that element (this has already been implemented for you). You may or may not need all lines.

For the purposes of this problem, assume

- Access to the entire standard library
- `data` should contain copies of strings; memory allocation always succeeds.
- String comparisons are done lexicographically.
- For an empty heap, allocate `INIT_CAP` elements initially and every time afterwards, if we run out of space, double its size. Do not declare any new variables or introduce new conditional statements or loops.

```
#define INIT_CAP 10
#define RESIZE_FACTOR 2
void free_heap(struct min_heap* m) {

      for (size_t i = 0; i < m->size; i++)
            free(m->data[i]);
      free(m->data);
      free(m);

}

void insert(struct min_heap* m, char* elem) {

        if(m->capacity == 0) {

                m->data = malloc(sizeof(char*) * INIT_CAP);

                m->capacity = INIT_CAP;

        } else if(m->size >= m->capacity) { #check if we need to resize

                m->data = realloc(m->data, sizeof(char*)*(m->capacity*2));

                m->capacity = m->capacity * 2;

         }

         char* temp = malloc((strlen(elem) + 1) * sizeof(char));

         strcpy(temp, elem);

         int iter = m->size++;

      // Bubble up inserted element to correct position in heap
      while(iter != 0 && strcmp(temp, m->data[(iter - 1) / 2]) < 0)) {

                  m->data[iter] = m->data[(iter - 1) / 2];

                  iter = (iter - 1) / 2;

      }
      m->data[iter] = temp;

}
```

Q4: This question might be a RISC

Consider the following RISC code. The function **read_input** will prompt the user to provide a 32-bit integer and stores it in a0. As a reminder, the **ecall** instruction will call an OS function (determined by the ecall number stored in a0), with the value stored in a1 as the function's argument. **ecall numbers are as follows: 1 = print integer, 4 = print string, 10 = exit.**

```
1.  .data
2.  Boom:  .asciiz "Ayy, man's not dumb." # strlen(this string) == 20
3.  Skraa: .asciiz "The ting goes skkkraaa." # strlen(this string) == 23
4.
5.  .text
6.  MAGIC:          # prologue
7.                  la s0, Risc-tery
8.                  la s1, Boom
9.                  addi s2, x0, 0x61C
10. Get:            jal read_input  # provide either 0 or 1 (USER_IN_1)
11.                 beq a0, x0, Default
12. Risc-tery:      jal read_input  # provide any integer (USER_IN_2)
13.                 beq a0, x0, QuickMaths # Q2
14.                 addi t0, x0, 9
15.                 slli t0, t0, 2
16.                 add s0, s0, t0
17.                 lw t1, 0(s0)
18.                 slli a0, a0, 20   # shift user input by 20
19.                 add t1, t1, a0
20.                 sw t1, 0(s0)
21. QuickMaths:     addi a1, s1, 0
22.                 addi a0, x0, 4
23.                 ecall
24.                 j Done
25. Default:        addi a0, x0, 1
26.                 add a1, s2, x0
27.                 ecall
28. Done:           # epilogue
29.                 jalr ra
```

1. Consider the function `MAGIC`. The prologue and epilogue for this function are missing. Which registers should be saved/restored on the stack? Select all that apply.
   `s0, s1, s2, ra`
2. Assume the assembler has been run. What machine code is the line commented Q2 `(beq a0, x0, QuickMaths)` converted to? Please write your answer in the table provided on your answer sheet.

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
0b____0000000__	0b___00000__	0b_01010__	000	0b__10000_	1100011

3. Assume you call `MAGIC`, providing the input 0 (to the call to `read_input` on line 10, commented `USER_IN_1`). What does the program print? Write your answer in the blank provided.
   `0x61C or 1564`

4. Say that you call `MAGIC` again. What input values to the calls to `read_input` will print "Ayy, man's not dumb"? Remember each call to `read_input` takes in a single input value. If you select option D for part (b), then please write the exact value that the input should be.

   a) USER_IN_1:
   A. 0 B. 1 C. Not Possible

   b) USER_IN_2:
   A. Any integer B. Any nonzero integer C. 0 D. Exact value _____

5. Assume we can both read and write to **any** valid memory address. Please specify the input values to `read_input` such that calling `MAGIC` prints out "The ting goes skkkraaa."
   a) USER_IN_1:
   A. 0 B. 1 C. Not Possible

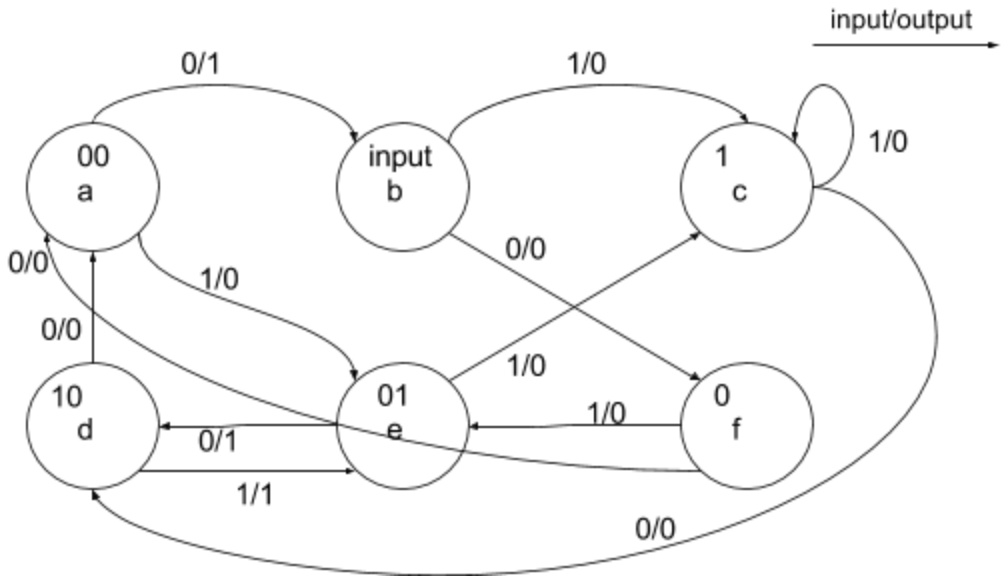
   b) USER_IN_2:
   A. Any integer B. Any nonzero integer C. 0 D. Exact value ___21_____
   `strlen(string at Boom) + 1 (null terminator) = 21`

Q5: SDS + FSM

1. Lisa is trying to create a FSM that outputs 1 if it has seen a sandwich of three values (e.g. 010, 101) ending at the current value, and 0 otherwise. Also, **after observing the last 0 value in a series of 3 0's (000), the system will output 1 and reset (i.e. returns to the START state)**. A series of 3 1's (111) is not a sandwich and should output 0. For example, an input of 001010001010 will output 000111010011.

Lisa created the FSM below. Unfortunately, the FSM file was corrupted: All states are provided, but **three arrows and several "input/output" pairs** are missing. Help Lisa add back the missing arrows. **For each arrow to add in, the starting state is given. Select the letter corresponding to the ending state on your answer sheet.**

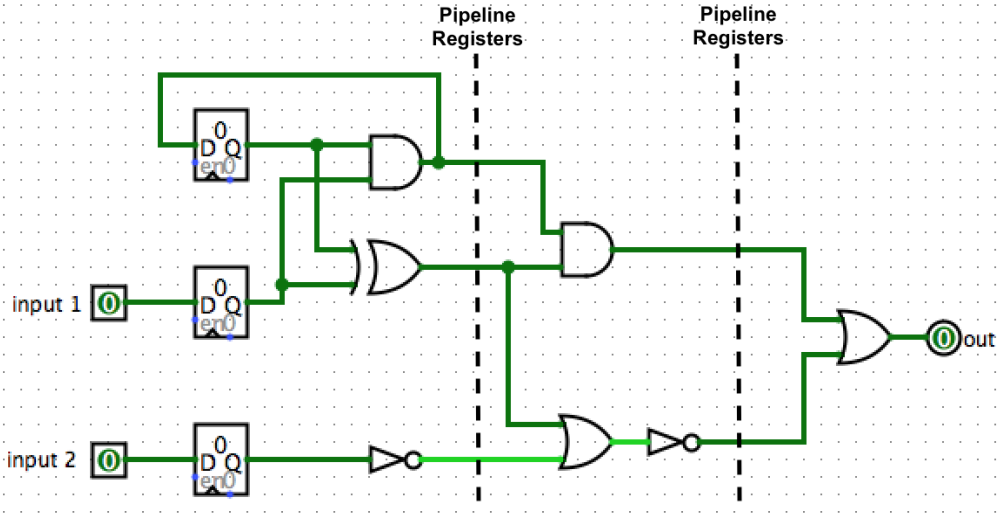
Hint: Lisa remembers that the START state is among the top row of states.



I.	Arrow from state **c** to state:	A. a	B. b	C. c	D. d	E. e	F. f
II.	Arrow from state **d** to state:	A. a	B. b	C. c	D. d	E. e	F. f
III.	Arrow from state **f** to state:	A. a	B. b	C. c	D. d	E. e	F. f

2. Consider the non-pipelined circuit diagram below with the following timings. Assume the registers have a clk-to-q delay of 5 ns and a setup time of 0 ns. The NOT gate has a propagation delay of 5 ns and other gates each have a delay of 15 ns.

What is the largest possible hold time (in ns) for the **top flip flop** such that there will be no violation with the given constraints above?

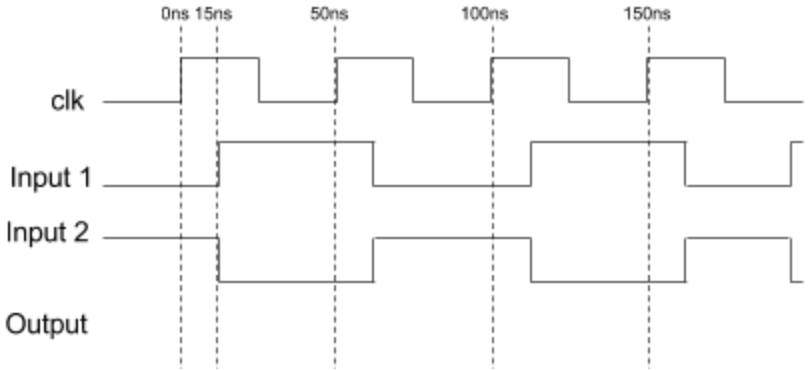


A. 0 B. 5 C. 10 D. 15 E. 20 F. 25 G. 30 H. 35

`Hold time <= clk-to-q delay + gate delay + setup time = 5+15+0 = 20 ns`

3. Considering again the non-pipelined circuit diagram, compute the output values at the given timesteps. Use the given time signals below. Assume the top register begins initially with 0.

I. T = 50 ns: 1 II. T = 100 ns: 0 III. T = 150 ns: 1



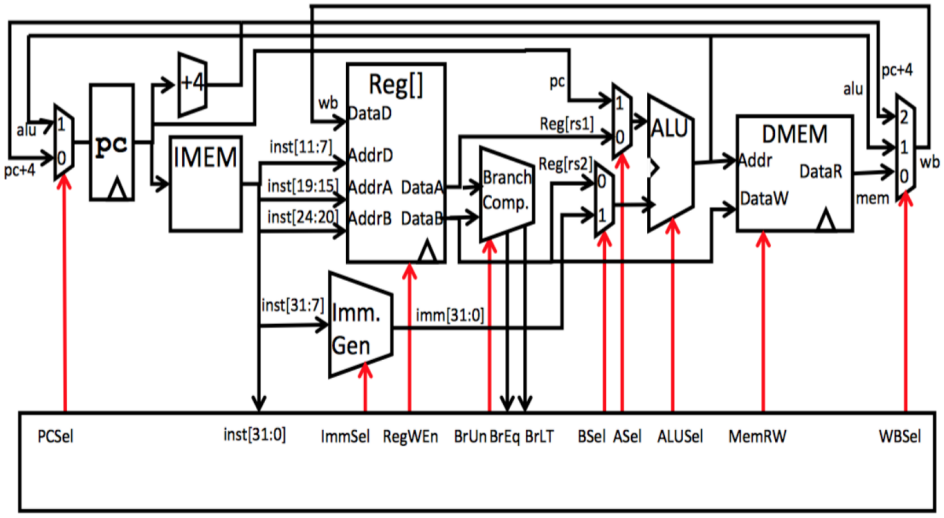
4. Now, pipeline registers are added to the circuit along the dotted lines labeled “Pipeline Registers”. Assume the hold time of all registers is 10 ns (which may or may not agree with your answer from part 2). Inputs 1 and 2 resolve 15 ns after each rising edge. What will be the minimum clock period (in ns) the new circuit can run at?

A. 20 B. 25 C. 30 D. 35 E. 40 F. 45 G. 50 H. 55

Three stage pipeline: You can get a critical path of 5 (clk-q) + 15 (logic) + 5 (NOT) = **25 ns**.

Q6: Game of Signals

The following canonical single-cycle datapath for a RISC-V architecture supports the base RISC-V ISA (RV32I).



For the following questions, we want to expand our instruction set to support some new instructions. For each of the proposed instructions below, choose ONE of the options below.

A. Can be implemented without changing datapath wiring, only changes in control signals are needed.(i.e. change existing control signals to recognize the new instruction)
B. Can be implemented, but needs changes in datapath wiring, only additional wiring, logical gates and muxes are needed.
C. Can be implemented, but needs change in datapath wiring, and additional arithmetic units are needed (e.g. comparators, adders, shifters etc.).
D. Cannot be implemented.

Note that the options from A to D gradually add complexity; thus, selecting B implies that A is not sufficient. You should select the option that changes the datapath the least (e.g. do not select C if B is sufficient). You can assume that necessary changes in the control signals will be made if the datapath wiring is changed.

1) Allowing software to deal with 2's complement is very prone to error. Instead, we want to implement the negate instruction, neg rd,rs1, which puts -R[rs1] in R[rd].
   Ans: A. This is a tricky question! Notice `neg` doesn't use all available bits, so we could make `neg rd, rs1` into a special R-type instruction `neg rd, x0, rs1`, such that the instruction does `R[rd]= x0 - R[rs1]`. Notice that subtraction is supported by our default datapath. So we only need to add the new control signal `neg`, which will produce the same ALUsel, Asel, Bsel, … signals as `sub` does.
2) Sometimes, it is necessary to allow a program to self-destruct. Implement `segfault rs1, offset(rs2)`. This instruction compares the value in `R[rs1]` and the value in `MEM[R[rs2]+offset]`. If the two values are equal, write 0 into the PC; otherwise treat this instruction as a NOP.
   Ans: C. Need to 1) Add a comparator after memory unit and wire the output to PCSel. 2) Add a zero wired to mux before PC. 3) Change corresponding PCSel signal width.

Q7: Caches

For this question, we will analyze the hit rates of 4 different caches:

- Cache A, a direct-mapped cache
- Cache B, a 4 way set-associative cache with LRU replacement
- Cache C, a fully associative cache with LRU replacement
- Cache D, an N-way set-associative cache with LRU replacement

No matter its type, each cache will have 64B cache blocks with a total capacity of 2 KiB, and have a **write through, no write allocate policy**. We will work with a **32-bit word-aligned**, physical memory space. Integers and floats are **1 word** long. Consider the following program:

```
struct Student {
      int sid;
      float epa;
      char login[3];
      char* ta;
};

#define LEN 128*32
int main() {
      int i;
      struct Student arr[LEN]; // initialized correctly with LEN
                                  Student structs
      // PART I
      for (i = 0; i < LEN; i += 128) {
            arr[i].epa += 10;
            arr[i + 2].epa += 10;
      }
      // PART II
      for (i = 0; i < LEN; i += 128) {
            arr[i + 1].epa = 15;
            arr[i + 3].epa = 15;
      }
}
```

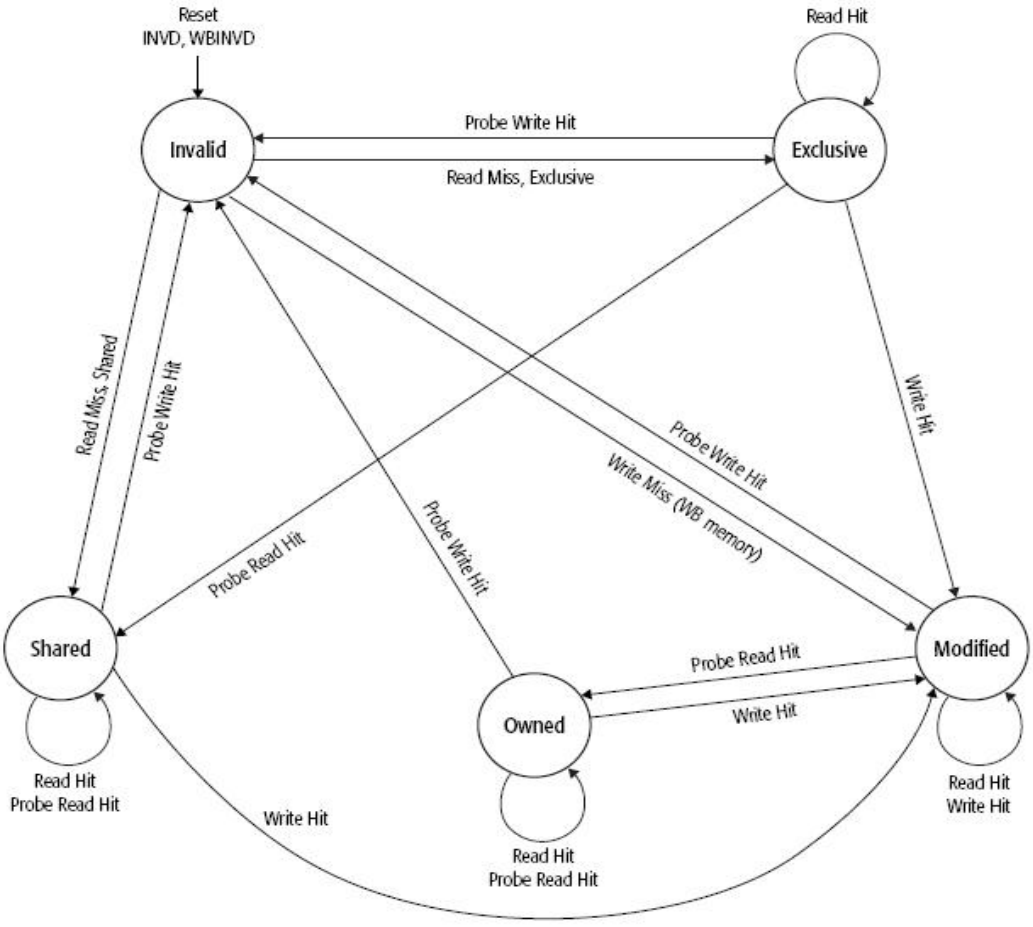
1. How many bytes is the `Student` struct when compiled for the RV32 ISA?
   A. 4 B. 7 C. 8 D. 12 E.15 F. 16
2. If we were to run the for loop, PART I, starting with cold caches, what would be the hit rate of the loop if we used cache A? How about if we used cache B instead?
   `75%; 75%`
3. If we were to run the for loop, PART II, starting with cold caches, what would be the hit rate of the loop using cache C? Write your answer in your answer packet.
   `50%`
4. If we were to use Cache D to run both PART 1 and PART 2 sequentially, starting with a cold cache before PART 1, what value of N would maximize the hit rate? `32. There are only 32 unique blocks being accessed and the cache can only have a maximum of 32 cache blocks.`

Q8: Cache Coherency

We are running code on a **dual-core** (core A & core B) processor with **Write Back, Write Allocate** caches implementing the MOESI protocol. The definitions and transition diagram of the states are given below.

1. `Invalid`: This block is not valid; it must be fetched to satisfy any attempted access.
2. `Shared`: This line is one of several copies in the system. This cache does not have permission to modify the copy.
3. `Modified`: This cache has the only valid copy of the cache line, and has made changes to that copy.
4. `Exclusive`: This cache has the only copy of the line, but the line is clean (unmodified).
5. `Owner`: This cache is one of several with a valid copy of the cache line, but has the exclusive right to make changes to it.

   Note: “Probe Write” is equivalent to saying “another core writes”



Both core A and core B are running the accesses below:

(1) Read from address 0x61C
(2) Write to address 0x61C
(3) Read from address 0x61C
(4) Write to address 0x61C
(5) Read from address 0x61C

Do not optimize or re-order the code. Assume the system guarantees cache coherency and executes in the following sequence:

A.1, A.2, B.1, B.2, A.3, B.3, B.4, A.4, A.5, B.5.

For all of the following questions, please select from the following answer choices:

Ⓐ Invalid Ⓑ Shared Ⓒ Modified Ⓓ Exclusive Ⓔ Owner

1. Fill in the table below with the states of the cache lines **after** every step.

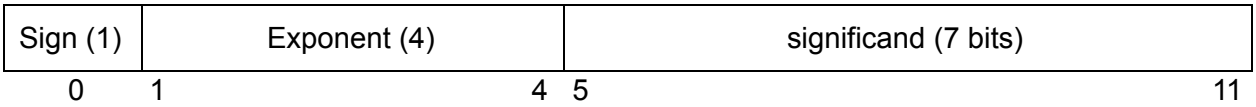
Core: Instruction	Core A, 0x61C state	Core B, 0x61C state	Is Memory up-to-update?
Initiating A & B	Invalid	Invalid	yes
A: read 0x61C	Exclusive	Invalid	yes
A: write 0x61C	Modified	Invalid	no
B: read 0x61C	Owned	Shared	no
B: write 0x61C	Invalid	Modified	no
A: read 0x61C	Shared	Owned	no
B: read 0x61C	Shared	Owned	no
B: write 0x61C	Invalid	Modified	no
A: write 0x61C	Modified	Invalid	no
A: read 0x61C	Modified	Invalid	no
B: read 0x61C	Owned	Shared	no

2. MOESI is extended from MESI. Using the possible MOESI states, please fill in the blanks to explain the advantage of MOESI over MESI:

"In MOESI, the (a) Owner stage has the advantage that one cache can write a block and still allow other caches with that block to be in the (b) Shared state. In MESI, one cache write would cause all other caches with that block to transition to the (c) Invalid state, and cause future cache misses."

Q9: Floating Point

Some of the 61C TAs get tired of having to convert floating-point values into 32 bits. As a result they propose the following smaller floating-point representation.
It consists of a total of 12 bits as shown below:



- The largest exp remains reserved as in traditional floating point
- The smallest exp follows the same denormalized formula as traditional floating point
- **Numbers are rounded to the closest representable value. Any numbers that have 2 equidistant representations are rounded down towards zero.**

All of the parts of this question refer to this floating-point scheme.

1. What is the smallest nonzero positive value that can be represented? Write your answer as a numerical expression in the answer packet.

2^{-13}

2. Consider some **positive normalized** floating-point number p where p is described as p = 2^y * 1.significand. What is the distance (i.e. the difference) between p and the next-largest number after p that can be represented? Write your answer as a numerical expression.

2^{y-7}

3. Now instead let p be a positive denormalized number described as p = 2^y * 0.significand. What is the distance between p and the next-largest number after p that can be represented?

2^{-13}

Q10: If only you could parallelize taking this test…

The following code snippets count the number of nonzero elements in the array.

```
int omp1(double arr[], int size) {
        int nonzero = 0;
        #pragma omp parallel for reduction(+:nonzero)
        for (int i = 0; i < size; i++) {
            if (arr[i] != 0)
                nonzero++;
        }
        return nonzero;
}
```

```
int omp2(double arr[], int size) {
        int nonzero = 0;
        #pragma omp parallel for private(nonzero)
        for (int i = 0; i < size; i++) {
            if (arr[i] != 0)
                nonzero++;
        }
        return nonzero;
}
```

```
int omp3(double arr[], int size) {
        int nonzero = 0;
        #pragma omp parallel for private(nonzero)
        for (int i = 0; i < size; i++) {
            if (arr[i] != 0)
                #pragma omp critical
                nonzero++;
        }
        return nonzero;
}
```

```
int omp4(double arr[], int size) {
        int nonzero = 0;
        #pragma omp parallel for
        for (int i = 0; i < size; i++) {
            if (arr[i] != 0)
                #pragma omp critical
                nonzero++;
        }
        return nonzero;
}
```

```
double omp5(double arr[], int size) {
        int n = 4;
        int res[4] = {0,0,0,0};
        // num_threads(4) runs the code block with 4
           threads.
        #pragma omp parallel num_threads(4) {
            int thread_id = omp_get_thread_num();
            for (int i = thread_id; i < size; i+=n) {
                if (arr[i] != 0)
                     res[thread_id]++;
            }
        }
        return res[0] + res[1] + res[2] + res[3];
}
```

```
double omp6(double arr[], int size) {
        int n = 4;
        int res[4] = {0,0,0,0};

        #pragma omp parallel num_threads(4) {
            int thread_id = omp_get_thread_num();
            #pragma omp for
            for (int i = thread_id; i < size; i+=n) {
                if (arr[i] != 0)
                    res[thread_id]++;
            }
        }
        return res[0] + res[1] + res[2] + res[3];
}
```

1. Which function(s) produce(s) the correct answer? 1, 4, 5

2. Assume cache loading/reloading is slower than thread switching. Which is the slowest correct function if:
    a. the size of the cache block is equal to `4 * sizeof(int)`? 5
    b. the size of the cache block is equal to `sizeof(int)`? 4

3. Which is the fastest correct function? 1

4. Please select the **correct and best** option to fill in the blanks to complete the SIMD implementation of counting the number of nonzero elements in a vector.

```
int sse(double arr[], int size) {
  int num_doubles = ______(a)______;

  int nonzero = 0;
  double *ans = calloc(num_doubles,
                       sizeof(double));

  //Assume sizeof(double) = 8
  __m128d ans_vec = _mm_setzero_pd();
  __m128d zeros = _mm_setzero_pd();
  __m128d ones = _mm_set1_pd(1.0);
  __m128d mask, data;
  int inc = _______________(b)_______________;

  int cutoff = _____________(c)_____________;

  for (int i = 0; i < cutoff; i += inc) {
    data = _mm_loadu_pd ((__m128d *) (arr+i));
    mask = _____________(d.i)_____________;
    mask = _____________(d.ii)_____________;
    ans_vec = _mm_add_pd(ans_vec, mask);
    _____________(e.i)_____________

  }
  _____________(e.ii)_____________

  for (int i = 0; i < num_doubles; i++)
    nonzero += ans[i];

  for (int i = cutoff; i < size; i++) {
   if (arr[i] != 0)
     nonzero++;
  }
  return nonzero;
}
```

a)
- A. 2
- B. 4
- C. 8
- D. 128

b)
- A. num_doubles;
- B. 1
- C. 4
- D. sizeof(__m128d)
- E. sizeof(double)

c)
- A. size/inc*inc
- B. size
- C. inc
- D. size*inc/inc

d.i)
- A. _mm_cmpeq_pd(zeros, data);
- B. _mm_cmpneq_pd(zeros, data);

d.ii)
- C. _mm_and_pd(mask, ones);
- D. _mm_and_pd(mask, zeros);

e.i)
- A. _mm_storeu_pd(ans,ans_vec);
- B. _mm_loadu_pd(ans, ans_vec);
- C. Leave blank/do not add any code

e.ii)
- D. _mm_storeu_pd(ans,ans_vec);
- E. _mm_loadu_pd(ans, ans_vec);
- F. Leave blank/do not add any code

Q11: Atomic Instructions

Shown below is a simplified implementation of a thread pool. The goal of a thread pool is to manage the execution of multiple threads; the goal of this specific implementation is to limit the total number of threads doing work at any given moment.

```
// There should never be more than 4 threads doing work at any given
// moment.
int counter = 4;

// This method is run by each thread.
void run_thread() {
  acquire(&counter);
  // Do work...
  release(&counter);
}
// Publicly visible method to queue up a new thread to be run.
void queue_thread() {
  // Assume that this method creates and executes a new thread.
  create_and_execute_thread(&run_thread);
}
```

Your task is to implement the acquire() and release() methods in RISC-V using `amoswap`, `lr`, and/or `sc`. **On your answer sheet, please complete the RISC-V code to implement both functions acquire() and release()**. You may not need all of the lines provided.

The expected behavior of the two methods is below. Note: **&counter is stored in register a0**.

`void acquire(int* c)`	Repeatedly checks the value of counter until it finds that counter > 0, then attempts to decrement the value of the counter. Restart if the decrement fails for any reason.
`void release(int* c)`	Repeatedly attempts to increment value of the counter until successful in incrementing

The specifications for these instructions are reproduced from the Green Sheet below:

`lr rd, (rs1)`	R[rd] = M[R[rs1]]; registers a reservation on the memory address R[rs1].
`sc rd, rs2, (rs1)`	If there is a reservation on the memory address R[rs1], then M[R[rs1]] = R[rs2] and R[rd] = 0; otherwise, R[rd] = 1.
`amoswap rd, rs1, rs2`	Atomically, R[rd] = M[R[rs1]] and M[R[rs1]] = R[rs2]

```
acquire:    lr   t0, (a0)
            beq  t0, x0, acquire
            addi t0, t0, -1
            sc   t1, t0, (a0)
            bne  t1, x0, acquire
            jr   ra
```

```
release:    lr   t0, (a0)
            addi t0, t0, 1
            sc   t1, t0, (a0)
            bne  t1, x0, release
            jr   ra
```

Q12: Virtual Memory

In this question, you will be analyzing the virtual memory system of a single-processor, single-core computer with 4 KiB pages, 1 MiB virtual address space and 1 GiB physical address space. The computer has a single TLB that can store 4 entries. You may assume that the TLB is fully-associative with an LRU replacement policy, and each TLB entry is as depicted below.

TLB Entry

Valid Bit	Permission Bits	LRU Bits	Virtual Page Number	Physical Page Number

1. Given a virtual address, how many bits are used for the Virtual Page Number and Offset?

VPN: 8, Offset: 12.

Each virtual address is log_2(1 MiB) = 20 bits in total. We know that each page is 4 KiB, so the offset is log_2(4 KiB) = 12 bits. This leaves 8 bits for the virtual page number.

2. Given a physical address, how many bits are used for the Physical Page Number and Offset?

PPN: 18, Offset: 12

The offset will remain 12 bits, as the page size is constant. As physical addresses are log_2(1 GiB) = 30 bits wide, we have 18 bits for the physical page number.

For the next 2 parts, consider that we are running the following code, in parallel, from two distinct processes whose virtual memory specifications are the same as that of above. Both arrays are located at page-aligned addresses. As a note, 65536 = 2^{16}.

Process 0	Process 1

`int a[65536];`
`for (int i = 0; i < 65536; i += 256) {`
`  a[i] = i;`
`  a[i + 64] = i + 64;`
`  a[i + 128] = i + 128;`
`  a[i + 192] = i + 192;`
`}`
	`int b[65536]`
	`for (int j = 0; j < 65536; j += 256) {`
	`  int x = j + 256;`
	`  b[x-1] = j;`
	`  b[x-2] = j+1;`
	`  b[x-3] = j+2;`
	`  b[x-4] = j+3;`
	`}`

As our computer has only a single processor, the processes must share time on the CPU. Thus, for each iteration of the processes' respective for loop, the execution on this single processor follows the diagram at the top of the next page. A blank slot for a process means that it is not currently executing on the CPU.

Time	Process 0	Process 1
0	a[i] = i;	
1	a[i + 64] = i + 64;	
2		int x = j + 256;
3		b[x-1] = j;
4		b[x-2] = j+1;
5	a[i + 128] = i + 128;	
6	a[i + 192] = i + 192;	
7		b[x-3] = j+2;
8		b[x-4] = j+3;

3. What is the TLB **hit rate** for executing the above code assuming that the TLB starts out cold (i.e. all entries are invalid)? Only consider accesses to data and ignore any effects of fetching instructions. You may assume that the variables i, j and x are stored in registers and therefore do not require memory accesses.
Remember: you must flush the TLB on a context switch from one process to another!
50%. The first access is a[0], which brings in the VPN translation for a[64], the next access. When we switch processes, the TLB will be flushed, so the first two accesses to the array b will follow the pattern miss-hit. When we switch back to process 0, we access a[128] (miss because TLB empty) and a[192] (hit because brought in by a[128]). The execution of the second two accesses to b are also miss-hit because the TLB is flushed in between. This pattern continues to give a hit rate of 50%.

As opposed to the TLB architecture described above, let us consider **a *tagged* TLB**. In a tagged TLB, each entry additionally contains the Address Space ID (ASID), which uniquely identifies the virtual address space of each process. A tagged TLB entry is shown below.

Tagged TLB Entry

Valid Bit	Permission Bits	LRU Bits	ASID	VPN	PPN

On a lookup, we consider a hit to be if the (VPN, ASID) pair is present in the tagged TLB. This redesign allows us to keep entries in the TLB even if they are not a part of the process running on the CPU, so we do not have to flush the TLB when switching between processes.

Consider that we are using a tagged TLB and running the code in the manner described above.

4. What is the **hit rate** for the tagged TLB assuming it again starts out cold? You may make the same assumptions about the variables i, j, x and ignore the effects of fetching instructions.

15/16. We first access a[0] , which brings in page 0 of the array for process 0. The first access is a miss, but the following access of a[64] is a hit because the mapping is in the TLB. When we switch to process 1, we access array b twice. These accesses, however, will be in the same page because they lie within a 1 KiB range and the start address of b is page-aligned. Thus, we will miss the first time and hit on the second. When we switch back to process 0, the entry is already there (we don't flush anymore) so we get 2 hits. Going back to process 1 will give us the same result. The next miss will occur when we get to the next page, which occurs after 4 iterations because each iteration moves 1 KiB. As there are 4 accesses per iteration, we have 15 hits for every 16 accesses (per process). Thus, in total, we have a hit rate of 15/16.

5. What is the smallest number of entries the TLB can have to still have the hit rate found in part 4?

2 Entries. We need to maintain the mapping for each processes.

Q13: KnocKnock

While working at KnockOff Corp., you are in charge of a "new and improved" Amazon's Alexa: KnockOff's Nicky. Your team will design a box to listen to user input, transmit it to KnockOff's servers, and process data. The Hamming code table is provided for your reference.

Bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7	D8	D9	D10	D11
P1	X		X		X		X		X		X		X		X
P2		X	X			X	X			X	X			X	X
P4				X	X	X	X					X	X	X	X
P8								X	X	X	X	X	X	X	X

1. Unfortunately the servers have a poor ComSat network connection and require error correcting codes to repair received packets. If compressed recordings are **34 bits**, how many bits are needed to **correct a single error** using SEC Hamming codes?

   A. 1 bit B. 2 bit C. 3 bits D. 4 bits E. 5 bits F. 6 bits G. More than 6 bits

   6 bits allow you to cover up to 57 data bits; 5 bits only give you 26 data bits)

2. To test your software, you use the following 7 bits (which include the correction code). What data bits are encoded by the following code word?

   7-bit Code Word: **0101101_2** Data Bits: 0101_2

   There was no corruption in the data bits originally

3. KnockOff's single server can't process millions of simultaneous requests. You decide to scale and find that the work comes in two parts: distributing and processing. Distribution cannot be parallelized and is 1% of the total work. Processing is the remainder. If you need an overall speedup of 50x, how many servers will you need to handle processing?

   A. 1 B. 50 C. 55 D. 93 E. 99 F. 101 G. Not possible

   1/(0.01 + 0.99/99) = 50

4. KnockOff would like your opinions on their dependable data storage on their server. Currently, the storage uses RAID 1 but you are suggesting to upgrade to RAID 5.

   a. First, name one inventor of RAID: Katz, Patterson, Gibson

   b. Assume we want to tolerate one disk failure. If a server has 3 disks (of equal capacity) filled with data, implementing RAID5 requires 1 additional disk(s), while RAID1 requires 3 additional disk(s).

   c. True/False: A single small write is faster with RAID5 than it is with RAID1.

      We still need to write to 2 disks but also need to perform 2 XOR operations according to the small-write algorithm.

   d. Small random reads have _____ throughput on RAID 5 than on RAID1

      A. higher B. lower C. about the same

   e. If a single disk fails, recovery time with RAID5 is ___ recovery time with RAID1.

      A. < B. = C. >

      RAID 5 requires XORing parity blocks with all other blocks used to compute the parity block. This also involves multiple reads and then writing the result. RAID 1 requires just 1 read and 1 write.

5. Knockoff also wants to improve their disk performance. Do each of the following improvements successfully decrease seek time:

   a. Decrease radius of the disk True / False
   b. Decrease number of platters True / False

6. KnockOff Labs experiments with replacing servers with a neural botnet of smartphones to process Nicky data. Currently, **1 million** (1,000,000) **servers** each consume **0.1 kW** and result in a **PUE of 2.0** . The smartphone setup will have a **PUE of 1.5**. with **1 million smartphones** that each consume **0.01 kW**. Electricity costs **$0.01/kWh.** Assume there are **10,000 hours/year**. What are the **cost savings** from using smartphones for a year?

   1,000,000 servers * 10,000 hours/year * 0.01 dollars/kWh *
   (2.0 * 0.1 kW/server - 1.5 * 0.01 kW/server) = $18,500,000 or 18.5 million dollars