CS 61C Fall 2023

Garcia, Yokota Midterm

Solutions last updated: Tuesday, October 17, 2023

PRINT your name: ______________________________

PRINT your student ID: ______________________________

You have 110 minutes. There are 7 questions of varying credit (100 points total).

Question:	1	2	3	4	5	6	7	Total
Points:	13	20	14	19	15	18	1	100

For questions with **circular bubbles**, you may select only one choice.

- ○ Unselected option (completely unfilled)
- ● Only one selected option (completely filled)
- ◎ Don't do this (it will be graded as incorrect)

For questions with **square checkboxes**, you may select one or more choices.

- ☐ You can select
- ■ multiple squares
- ■ (completely filled)

Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation. For coding questions with blanks, you may write at most one statement per blank and you may not use more blanks than provided.

If an answer requires hex input, you must only use capitalized letters (`0xDEADBEEF` instead of `0xdeadbeef`). For hex and binary, please include prefixes in your answers unless otherwise specified, and do not truncate any leading 0's. For all other bases, do not add any prefixes or suffixes.

Write the statement below in the same handwriting you will use on the rest of the exam.

> I have neither given nor received help on this exam (or quiz), and have rejected any attempt to cheat; if these answers are not my own work, I may be deducted up to 0x0123 4567 89AB CDEF points.

SIGN your name: ______________________________

This content is protected and may not be shared, uploaded, or distributed.

This page intentionally left (mostly) blank.

The exam begins on the next page.

This content is protected and may not be shared, uploaded, or distributed.

Q1 (Potpourri) **(13 points)**

For Q1.1-Q1.3, convert the 8-bit binary `0b0100 1011` to decimal, treating it as...

Q1.1 (1 point) ...an **unsigned** integer.

> **Solution:** 75
> **Grading:** All-or-nothing.

Q1.2 (1 point) ...a **two's complement** integer.

> **Solution:** 75
> **Grading:** All-or-nothing. Credit was awarded if the answer was the same as Q1.1 to avoid double jeopardy.

Q1.3 (1 point) ...a **sign-magnitude** integer.

> **Solution:** 75
> **Grading:** All-or-nothing. Credit was awarded if the answer was the same as Q1.1 or Q1.2 to avoid double jeopardy.

Q1.4 (2.5 points) What is the decimal value of `z` in the snippet of C code below?

```
int8_t x = 101;
int8_t y = 77;
int8_t z = x + y;
```

> **Solution:** -78
> **Grading:** Partial credit was awarded for one minor computational error in converting 2's complement (-77 for forgetting the +1, and 78 for forgetting to negate the number).

Q1.5 (3 points) Translate the following RISC-V instruction to hexadecimal: `sw s0 10(s2)`

> **Solution:** `0x00892523`
> **Grading:** Opcode, `funct3`, `rs1`, `rs2`, and immediate were graded separately. A -1.5 point penalty was applied if the answer was not a 32-bit hexadecimal integer.

This content is protected and may not be shared, uploaded, or distributed.

For Q1.6-Q1.8, indicate the stage of CALL that...

Q1.6 (1.5 points) ...converts pseudoinstructions into equivalent instructions.

○ Compiler ● Assembler ○ Linker ○ Loader

Grading: All-or-nothing.

Q1.7 (1.5 points) ...determines if functions are called with valid variable types.

● Compiler ○ Assembler ○ Linker ○ Loader

Grading: All-or-nothing.

Q1.8 (1.5 points) ...jumps execution to the start of the program.

○ Compiler ○ Assembler ○ Linker ● Loader

Grading: All-or-nothing.

This content is protected and may not be shared, uploaded, or distributed.

Q2 🍕🍉🍰🥧 *(C)* **(20 points)**

Recall in lab, we implemented some functions for a `vector_t` struct. In this question, we will add support for slicing a `vector_t`.

To do so, we've made some updates to the `vector_t` type from lab. You may assume that all necessary standard libraries are included.

```
typedef struct vector_t {
    // Number of elements in the vector; you may assume size > 0
    size_t size;

    // Pointer to the start of the vector
    int* data;

    // Number of child slices
    size_t num_slices;

    // Array of the vector's child slices, or NULL if num_slices == 0
    struct vector_t** slices;

    // true if the vector is a child slice of another vector, otherwise false
    bool is_slice;
} vector_t;
```

Useful C function prototypes:

```
void* malloc(size_t size);
void free(void *ptr);
void* calloc(size_t num_elements, size_t size);
void* realloc(void *ptr, size_t size);

size_t strlen(char* s);
char* strcpy(char* dest, char* src);
```


This content is protected and may not be shared, uploaded, or distributed.

Implement the function `vector_slice`, which should return a slice of a `vector_t` at the given indices, with the following signature:

- `vector_t* v`: A pointer to the parent vector to create the slice from.
- `int start_index`: The beginning index of the new slice's data (inclusive)
- `int end_index`: The ending index of the new slice's data (exclusive). You may assume that `end_index > start_index`.
- Return value: A `vector_t*` representing data as described by `start_index` and `end_index`. A parent `vector_t` shares (portions of) its `data` array with all of its descendant slices.

For example:

```
// vec_a has the elements [0, 1, 2, 3, 4]
vector_t* vec_a = /* omitted */;

// vec_b should be of size 2 and have the elements [1, 2]
vector_t* vec_b = vector_slice(vec_a, 1, 3);

vec_b->data[1] = 10;
// At this point, vec_a should be [0, 1, 10, 3, 4]
//                vec_b should be [1, 10]
```

```
1 vector_t* vector_slice(vector_t* v, int start_index, int end_index) {
2     vector_t* slice = calloc(1, sizeof(vector_t));
                                 Q2.1
3     if (slice == NULL) { allocation_failed(); }
4     slice->size = end_index - start_index;
                        Q2.2
5     slice->data = &v->data[start_index];
                        Q2.3
6     slice->is_slice = true;
                 Q2.4
7     v->slices = realloc(v->slices, (v->num_slices+1)*sizeof(vector_t*));
                                          Q2.5
8     if (v->slices == NULL) { allocation_failed(); }
9     v->slices[v->num_slices] = slice;
                   Q2.6          Q2.7
10    v->num_slices += 1;
            Q2.8
11    return slice;
12 }
```

Grading: Each subpart was graded individually, and partial credit was awarded for minor errors that was unambiguously not a result of a conceptual misunderstanding.

This content is protected and may not be shared, uploaded, or distributed.

The `vector_t` struct definition is repeated below for your convenience:

```
typedef struct vector_t {
    // Number of elements in the vector; you may assume size > 0
    size_t size;

    // Pointer to the start of the vector
    int* data;

    // Number of child slices
    size_t num_slices;

    // Array of the vector's child slices, or NULL if num_slices == 0
    struct vector_t** slices;

    // true if the vector is a child slice of another vector, otherwise false
    bool is_slice;
} vector_t;
```

To accommodate these changes to the `vector_t` type, we need to update the `vector_delete` function to properly free our new data structure. When a `vector_t` is deleted, all descendant slices of the `vector_t` should also be freed. You may assume that `vector_delete` is only called on a `vector_t` that is not a slice.

```
 1 void vector_delete(vector_t* v) {
 2     if (!v->is_slice) {
                Q2.9
 3         free(v->data);
                Q2.10
 4     }
 5     for (int i = 0; i < v->num_slices; i++) {
 6         vector_delete(v->slices[i]);
                       Q2.11
 7     }
 8     if (v->num_slices > 0) {
 9         free(v->slices);
                 Q2.12
10     }
11     free(v);
        Q2.13
12 }
```

Grading: The question was split into the following 4 subparts during grading:

- Freeing `v->data`
- Freeing `v->slices`
- Freeing `v`

This content is protected and may not be shared, uploaded, or distributed.

- Deleting each vector in `v->slices`

For the first three subparts, full credit was awarded if the line was entirely correct, and half credit was awarded when the line was present, but was not entirely correct (e.g. called `free` on the same pointer multiple times, calling `free` with an incorrect condition, etc).

For the final subpart, full credit was awarded for using `vector_delete`, while half credit was awarded for using `free`.

Errors were categorized into minor and major errors. Minor errors are syntax errors that still demonstrate understanding of the question, but would cause the compiler to fail (e.g. missing parentheses, missing `v->`). Major errors are errors that demonstrate some misunderstanding of the topic being assessed (e.g. off-by-one level of reference, multiplying the index by the size of each element in the array). Unambiguous misspellings or typos were not penalized.

This content is protected and may not be shared, uploaded, or distributed.

Q3 👉🎈🎈👈 *(Floating Point)* (14 points)

For the entirety of this question, assume that we are using an IEEE 754 standard floating-point representation with 16 bits: 1 sign bit, 7 exponent bits (with a standard bias of -63), and 8 significand bits.

For questions Q3.1 and Q3.2, convert the value to hexadecimal, using the floating-point representation above.

Q3.1 (2 points) -2.25

Solution: $-2.25 = -1.125 \times 2^1$ is $1.001_2 \times 2^1$.
After the bias, this means the exponent we need to encode is $1 + 63 = 64$ or `0b1000000`.
`0b1 1000000 00100000`
`0b1100 0000 0010 0000`
`0xC020`
Grading: The sign, exponent, and mantissa bits were graded separately as all-or-nothing. A 1 point penalty was applied if the answer had incorrect number of bits.

Q3.2 (2 points) 1.75×2^{-63}

Solution: This is a denormalized number, as the smallest possible exponent in this system is -62.
$1.75 \times 2^{-63} = 1.11_2 \times 2^{-63} = 0.111_2 \times 2^{-62}$
`0b0 0000000 11100000`
`0b0000 0000 1110 0000`
`0x00E0`
Grading: The sign, exponent, and mantissa bits were graded separately as all-or-nothing. A 1 point penalty was applied if the answer had incorrect number of bits.

For questions Q3.3 and Q3.4, what is the value of the floating-point number? Express your answer as $x \times 2^y$, where x is a decimal such that $1 \le |x| < 2$ and y is an integer. For NaN, write "NaN"; for positive infinity, write "$+\infty$"; for negative infinity, write "$-\infty$".

Q3.3 (2 points) `0x7F9A`

Solution: `0x7F9A`
`0b0111 1111 1001 1010`
`0b0 1111111 10011010`
With an exponent field of `0b1111111` and nonzero mantissa, this is a NaN.
Grading: All-or-nothing.

This content is protected and may not be shared, uploaded, or distributed.

Q3.4 (2 points) `0x7000`

Solution: `0x7000`

`0b0111 0000 0000 0000`

`0b0 1110000 00000000`

This exponent is $64 + 32 + 16$. After subtracting the bias of -63, this leaves us with a final exponent of 49.

The mantissa is all 0, so together with the implied one this gives an answer of 1×2^{49}.

Grading: The sign, exponent, and coefficient were graded separately as all-or-nothing. A 1 point penalty was applied if the answer had incorrect number of bits.

Q3.5 (3 points) List all numbers k in this floating-point representation such that the smallest floating-point number strictly greater than k is exactly $2k$. Express your answer as $x \times 2^y$, where x is a decimal such that $1 \leq |x| < 2$ and y is an integer. If there are no such numbers, write "None".

Solution: The wording of the question doesn't allow for negative answers - something greater than a negative number will always have smaller absolute value.

Additionally, k cannot be a normal number, since we essentially need k to be equal to the step size between representable numbers. Even among the denormal numbers, where the step size is $2^{-62} \times 2^{-8} = 2^{-70}$, $k = 2^{-70}$ is the only value that works.

Grading: Half credit was awarded for off-by-one errors (2^{-69} or 2^{-71}. 0.5 points were deducted for having additional answers.

Q3.6 (3 points) Express, in hexadecimal, the smallest strictly positive representable number k such that $k + 1$ is also representable in this floating-point representation.

Solution: Another way of reading this question is that $k+1$ must be the smallest representable number strictly greater than 1. We know that there are only eight mantissa bits, meaning that the smallest number greater than one is 1.00000001_2. Subtracting 1 gives $k = 0.00000001_2 = 1 \times 2^{-8}$.

Adding the bias of 63 gives 55, or `0b0110111`.

`0b0 0110111 00000000`

`0b0011 0111 0000 0000`

`0x3700`

Grading: The sign, exponent, and mantissa bits were graded separately as all-or-nothing. A 1.5 point penalty was applied if the answer had incorrect number of bits.

This content is protected and may not be shared, uploaded, or distributed.

Q4 🚀🛸🚁🛩 *(RISC-V)* **(19 points)**

The Jetidiah Propulsion Laboratory (JPL) is planning to race some vehicles around Jezero crater, including PersevRyance, CuriosEddy, and Bengenuity. They would like to write some RISC-V code to determine the best vehicle, and they need your help!

The `race` function, which has been correctly implemented for you, is defined as follows:

- Input in `a0`: `id`, the ID of a vehicle. You may assume that $0 \leq$ `id` < 200.
- Output in `a0`: The distance traveled (in nanometers) by the vehicle in a sprint as an unsigned integer. You may assume that this value is less than 2^{32}.

Implement `best_vehicle` as follows, using the `race` function:

- Input in `a0`: `num_vehicles`, the total number of vehicles being raced. The function should race vehicles with IDs between 0 and `num_vehicles - 1` (inclusive).
  You may assume that $0 <$ `num_vehicles` ≤ 200.
- Output in `a0`: The vehicle ID that traveled the farthest.
- `best_vehicle` should race the vehicles and return the ID of the vehicle that traveled the farthest. In case of a tie between multiple vehicles, return the lowest ID among those vehicles.

For example, given the following return values from the `race` function, `best_vehicle(5)` should race 5 vehicles (IDs 0, 1, 2, 3, and 4) and return 2, and `best_vehicle(2)` should race 2 vehicles (IDs 0 and 1) and return 0.

ID	race(ID)
0	600,000,000
1	100,000
2	3,000,000,000
3	1,564
4	3,000,000,000

This content is protected and may not be shared, uploaded, or distributed.

```
1  best_vehicle:
2       addi sp sp -20
           Q4.1
3       sw s0 4(sp)
           Q4.2
4       sw s1 8(sp)
           Q4.3
5       sw s2 12(sp)
           Q4.4
6       sw ra 16(sp)
           Q4.5
7       mv s0 a0
8       addi s1 x0 0 # The best distance so far
9       addi s2 x0 0 # The index of the best vehicle so far
10      addi t0 x0 0 # The index of the vehicle being tested
11 Loop:
12      mv a0 t0
         Q4.6
13      sw t0 0(sp)
        Q4.7
14      jal race
        Q4.8
15      lw t0 0(sp)
        Q4.9
16      bgeu s1 a0 Continue
           Q4.10
17      mv s1 a0
18      mv s2 t0
19 Continue:
20      addi t0 t0 1
           Q4.11
21      blt t0 s0 Loop
          Q4.12
22      mv a0 s2
          Q4.13
23      # Epilogue Omitted
24      # You may assume that the epilogue has been correctly
25      #     implemented based on your prologue.
26      ...
27      jr ra
```

Solution: Prologue: We need to decrement the stack pointer by 20 bytes for the following registers: `s0`, `s1`, `s2`, `ra`, and `t0`. The first three must be saved since we overwrite their values and they are saved registers. `ra` must be saved since we call another function within `best_vehicle`. `t0` must be saved since we rely on its value being unchanged across the call to `race`. Note that we cannot store `t0` in the prologue since its value changes during the execution of this function.

Loop: Before calling `race`, we need to move the index of the vehicle being tested to the `a0` register. To maintain the index counter in `t0` (a temporary register), we need to store `t0` before calling `race` and load in `t0` after. We need to use `bgeu` to compare the value returned from `race` since we

This content is protected and may not be shared, uploaded, or distributed.

return the lowest ID in the case of a tie, and the comparison must be unsigned since the values being compared may go up to $2^{32} - 1$, which exceeds the range of a signed 32-bit integer.

Continue: Add 1 to our current index of vehicles and then check to see that our index is less than `num_vehicles`. Before returning, we need to move the index of the best vehicle being tested (which is stored in `s2`) into `a0`.

Grading:

For each blank, a -0.5 point penalty is applied if the answer duplicates parts of the line that is already provided.

Prologue: 8 points total. For Q4.1 (2 points), 1 point was awarded for shifting `sp` in any way. For Q4.2-Q4.5 (6 points), 1 point each was awarded for storing `s0`, `s1`, `s2`, and `ra`, as well as no overlap between the store offsets, and not exceeding the amount of space set in Q4.1. There are also -1 point penalties given for minor syntax errors, and overwriting `sp...sp+3` (which is explicitly used later in the skeleton), and a -2 point penalty if stores are performed relative to the same non-`sp` register.

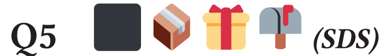
Moves: 1 pt each for Q4.6, Q4.11, and Q4.13, with a 0.5 point partial if the answer uses an add with an immediate operand or an addi with a register operand, but is otherwise correct.

Handling `t0`: 1 point each for 4.7 and 4.9.

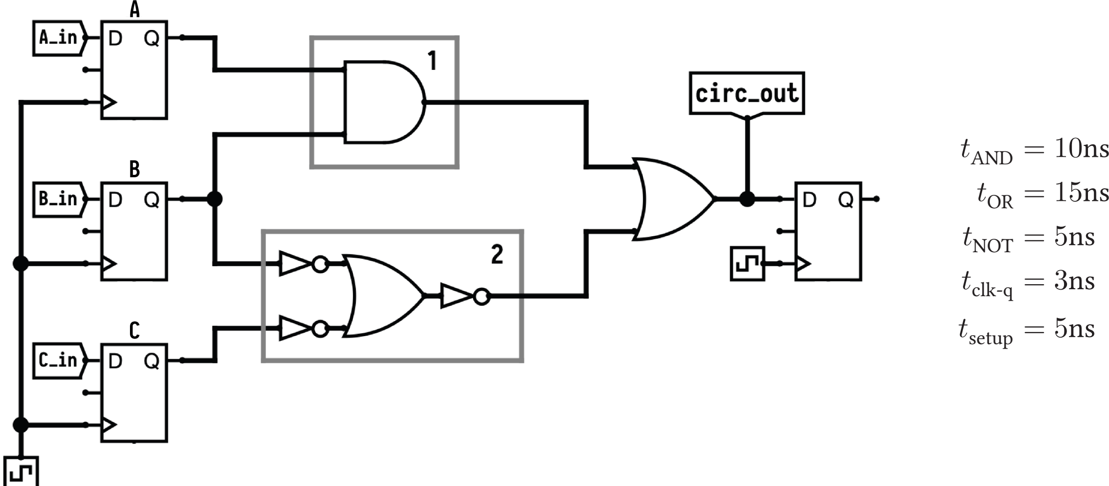
Jumps: 1.5 points for Q4.8, with a 0.5 point partial if the answer unconditionally jumps to `race` but doesn't store return address in `ra`.

Branches: 4.5 points total. 2.5 points for Q4.10 and 2 points for Q4.12, each with a -1 point penalty if the branch conditional is off by one, and a -0.5 point penalty for Q4.10 if the comparison isn't unsigned.

This content is protected and may not be shared, uploaded, or distributed.

Q5  *(SDS)* **(15 points)**

Consider the following circuit and timings:



Q5.1 (2.5 points) What is the **minimum clock period**, in nanoseconds, for the circuit above such that it will always exhibit well-defined behavior?

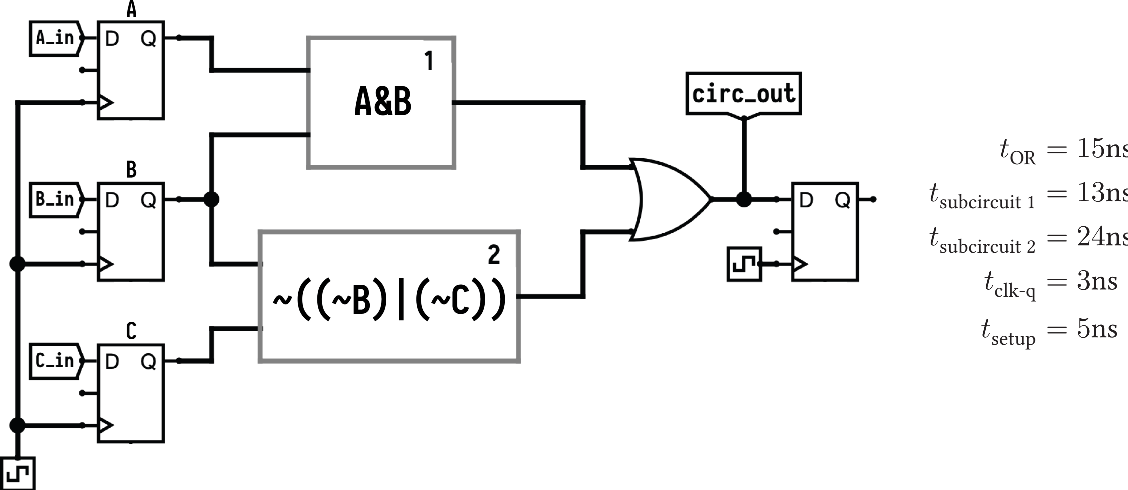
> **Solution:** 48ns. By inspection, we see that the longest combinational path starts at the Q-end of either register `B` or `C`, and passes through subcircuit `2`, an `OR` gate, and ending at the D-end of the right-most register.
>
> Adding up the delays along this path, we have $t_{\text{NOT}}+t_{\text{OR}}+t_{\text{NOT}}+t_{\text{OR}} = 5+15+5+15 = 40$ns. With the formula clock period $\geq t_{\text{clk-q}} + t_{\text{longest combinational path}} + t_{\text{setup}} = 3 + 40 + 5 = 48$, we get that the minimum clock period we can have is 48ns.
>
> **Grading:** Partial credit for identifying the longest combinational path.

Q5.2 (2.5 points) What is the **maximum hold time**, in nanoseconds, for the circuit above such that it will always exhibit well-defined behavior?

> **Solution:** 28ns. By inspection, we see that the shortest combinational path starts at the Q-end of register `A`, passing through subcircuit `1`, an `OR` gate, and ending at the D-end of the right-most register.
>
> Adding up the delays along this path, we have $t_{\text{AND}} + t_{\text{OR}} = 10 + 15 = 25$ns. With the formula hold time $\leq t_{\text{clk-q}} + t_{\text{shortest combinational path}} = 3 + 25 = 28$, we get that the maximum hold time we can have is 28ns.
>
> **Grading:** Partial credit for identifying the shortest combinational path.

This content is protected and may not be shared, uploaded, or distributed.

For the rest of this question, consider the updated circuit diagram and timings below. Subcircuits 1 and 2 are black-boxes that compute the Boolean expressions shown in the diagram. Regardless of your previous answers, assume the clock period is 80ns, and for subcircuits 1 and 2, their respective minimum and maximum combinational logic delays are equal. For example, if the input to subcircuit 1 changed at $t = 0$ (and no other changes occur), then the output to subcircuit 1 will change at $t = 13$ only.



Until this circuit settles at the correct input, it calculates two other Boolean expressions at the tunnel `circ_out`. Assume that register outputs are all 0 at $t = 0$, and that there are no setup or hold time violations.

Q5.3 (2 points) Let t_1 be the first time that `circ_out` changes. What is t_1 in nanoseconds? Clarified during exam: t_1 is the first time that `circ_out` *may change*.

> **Solution:** 31ns. This is the time it takes a change at inputs `A_in` or `B_in` to reach `circ_out`. This involves passing through `clk-to-q` of register `A` or `B`, and then subcircuit `1`, then the `OR` gate= $3 + 13 + 15 = 31$ns.
>
> **Grading:** Partial credit was awarded for identifying the correct combinational path that first reaches `circ_out`.

Q5.4 (2 points) Let t_2 be the second time that `circ_out` changes. What is t_2 in nanoseconds?

> **Solution:** 42ns. This is the time it takes a change at inputs `B_in` or `C_in` to reach `circ_out`. This involves passing through `clk-to-q` of register `B` or `C`, and then subcircuit `2`, then the `OR` gate = $3 + 24 + 15 = 42$ns.
>
> **Grading:** Partial credit was awarded for identifying the correct combinational path that reaches `circ_out` second.

Q5.5 (2 points) What simplified Boolean expression does the circuit calculate while $0\text{ns} \leq t < t_1$?

> **Solution:** 0. Since the wires are initialized to zero, the output at `circ_out` will be 0.

This content is protected and may not be shared, uploaded, or distributed.

Q5.6 (2 points) What simplified Boolean expression does the circuit calculate while $t_1 \leq t < t_2$?

> **Solution:** `A&B`. This is the output from subcircuit `1` ORed with 0.

Q5.7 (2 points) What simplified Boolean expression does the circuit calculate while $t_2 \leq t < 80$ns?

> **Solution:** `(A|C)&B`. This is equal to `(A&B)|(~((~B)|(~C))`.

> **Grading:** This entire section was graded either holistically or itemized, depending on what would attain a higher score. Holistically, credit was awarded for missing the zero case and directly putting `A&B` and `(A|C)&B` in order in any two out of the three answer blanks, as well as claiming that the first time the circuit changed was 42ns. Penalties were given for non-simplified Boolean expressions, and partial credit was awarded to answers that left out `clk-to-q` in the calculation of t_1 and t_2.

This content is protected and may not be shared, uploaded, or distributed.

Q6 😈🐶🐶🐶 *(CS61Cerberus)* **(18 points)**

Heracles has been tasked with yet another labour: taming the CS61Cerberus, the three-headed guard dog of Hades. Each head of CS61Cerberus has a favorite number (independent of the other heads), which is represented as an n-bit unsigned integer. In order to tame him, Heracles must determine the favorite numbers of all heads, by saying numbers and seeing how the heads respond.

Heracles can say any n-bit unsigned integer to any head, and the head will respond either `true` or `false` according to its rule and favorite number:

- The left head (Orion) receives its input and performs a bitwise OR on it with its favorite number. If the result is 0, Orion returns `true`. Otherwise, Orion returns `false`.
- The middle head (Andy) receives its input and performs a bitwise AND on it with its favorite number. If the result is 0, Andy returns `true`. Otherwise, Andy returns `false`.
- The right head (Luxor) receives its input and performs a bitwise XOR on it with its favorite number. If the result is 0, Luxor returns `true`. Otherwise, Luxor returns `false`.

Q6.1 Write a function **andy**, which receives an unsigned integer as input and returns a **bool** according to how Andy would react to that integer, assuming $n = 32$.

```
1 bool andy(uint32_t input) {
2     // Andy's favorite number has been omitted.
3     uint32_t andnum = /* omitted */;

4     return (andnum & input) == 0;
5 }
```

Solution: As this problem was primarily a test on syntax, syntax errors were penalized harshly here. Common errors included using logical AND instead of bitwise AND (&& instead of &) and having incorrect operator precedence (== has precedence over &, so the parentheses are necessary in order for the code to work). Alternate solutions exist, including `!(andnum & input)`. Bitwise NOT was accepted as an alternate solution to logical NOT, as it still yields a value that is "truthy", even though true is formally defined in C as 1.

Heracles writes three algorithms, **solve_orion**, **solve_andy**, and **solve_luxor** to find and return the favorite number of each head. Each of the three solver functions accepts one argument: a function that accepts a number and returns the response from the corresponding head.

Write **solve_orion**, **solve_andy**, and **solve_luxor** (for simplicity, your code only needs to work when $n = 32$). Your solution must be asymptotically optimal for full credit.

Additionally, write the worst-case asymptotic time complexity of the optimal solver function in Θ notation with respect to n, the number of bits that the head accepts. The Θ bound must hold for all values of n, not just when $n = 32$.

If no such algorithm exists, write "Impossible" in the code block, and leave the Θ bound blank.

This content is protected and may not be shared, uploaded, or distributed.

Q6.2

```
uint32_t solve_orion(bool(*orion)(uint32_t)) {
    // Your code here or Impossible
    Impossible
}
```

Solution: If Orion's number is nonzero, then orion will always return false, regardless of input. Therefore, it is impossible to solve this problem.

No credit was granted for solutions that worked only on 0, since it is trivially possible to make a solution that works for exactly one value (ex. `return 500;` works if and only if Orion's favorite number is 500)

No credit was granted for solutions that stated "Impossible" AND provided a Theta bound, as this was seen as an attempt to answer with two different answers (Impossible plus a partial credit attempt for a correct Theta bound); per standard policy, the worse of the two answers was chosen to be graded.

Q6.3

```
uint32_t solve_andy(bool(*andy)(uint32_t)) {
    // Your code here or Impossible
    uint32_t result = 0;
    for(int i = 0; i < 32; i++) {
        if(!andy(1<<i))
            result += 1 << i;
    }
    return result;
}
```

Solution: Andy returns true if and only if the input and Andy's favorite number share no 1 bit. Thus, `andy(1<<i)` is true if and only if the ith bit in Andy's favorite number is 0. We can thus construct the number with $\Theta(n)$ queries to Andy. This is asymptotically optimal; 2^n distinct numbers can be Andy's favorite number, so our decision tree must have at least 2^n leaves. Since each query branches at a factor of 2, we must have a decision tree with at least n depth, and therefore our code must run in $\Omega(n)$ time.

Multiple answers were possible; any answer that successfully solved Andy was given some credit (though inefficient answers also required a correct theta bound). Note that simply returning the first n that returns false from Andy is not correct; for example if Andy's favorite number was 3, `andy(1)` would return false before `andy(3)` is run.

This content is protected and may not be shared, uploaded, or distributed.

Q6.4
```
uint32_t solve_luxor(bool(*luxor)(uint32_t)) {
    // Your code here or Impossible
    uint32_t result;
    while(!luxor(++result));
    return result;
}
```

Solution: Luxor returns true if and only if the input is exactly the same as Luxor's favorite number. Therefore, our only solution is to check every value of n to see if it's Luxor's favorite number (as no other information is given from false values). Therefore, our optimal solution runs in $\Theta(2^n)$ time.

The most common error was to use an int as a looping variable in a for loop (ex. `for(int i = 0; i < 1<<32; i++)`), and not having the correct syntax when writing 2^{32}. Note that on 32-bit systems, ints are 32 bits long, so the loop terminates early when i overflows. Similar issues occur when using an `int32_t`, but some solutions that used `uint32_t` were accepted. For example, the loop `for(uint32_t i = 0; i <= 0xFFFF FFFF; i++)` worked, as all i are less than or equal to `0xFFFF FFFF`. Since Luxor has some favorite number, this loop is guaranteed to terminate, instead of infinite-looping.

Multiple solutions were possible; the given solution is a particularly funny one that uses some of the less-common features of C for code-golf purposes.

Explanation of the provided solution:

Note that we're allowed to start with any 32 bit number as the first input to Luxor. Thus, the solution is allowed to use the garbage value of an undeclared `result` instead of setting it to a starting value. Further, note that the return statement in the solution is NOT part of the while loop body; the while loop body is empty, and the return statement occurs immediately after the while loop terminates. Finally, a pre-increment was necessary instead of a post-increment, in order to have the correct result value.

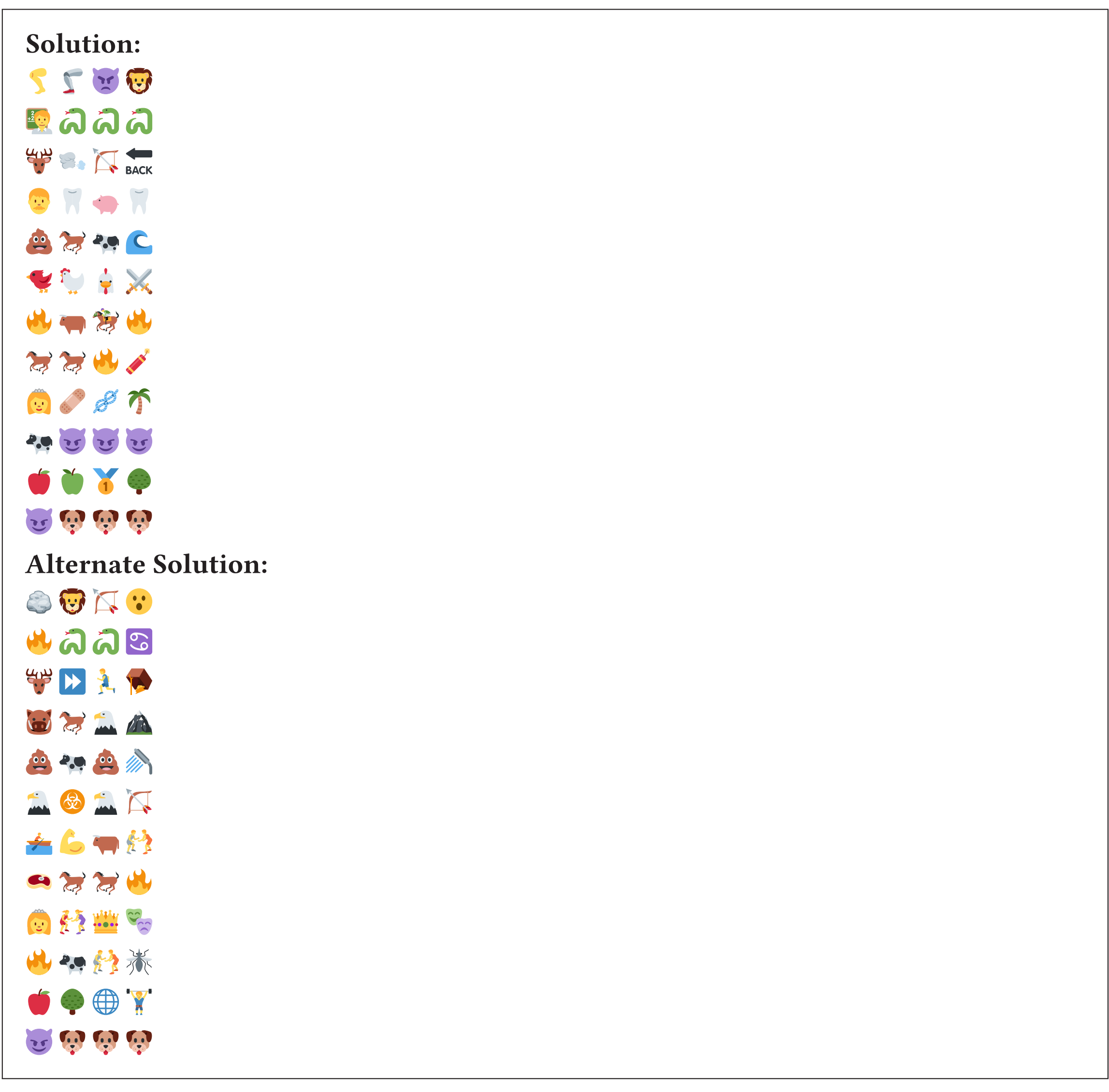
This content is protected and may not be shared, uploaded, or distributed.

Q7 🏁🔚🏁🔚 *(The Finish Line)* **(1 points)**

Everyone will receive credit for this question, even if you leave it blank.

Q7.1 (1 point) CS 61A defeated the Lambdanean Hydra in Fall 2022, CS 61B cleaned the Javaugean Stables in Spring 2023, and you've just tamed(?) CS61Cerberus. What are the 9 other labours of Heracles? Express your answer in emoji drawings.

Solution:



Alternate Solution:

Q7.2 (0 points) Is there anything you want us to know?

Solution: ¯_(ツ)_/¯

This content is protected and may not be shared, uploaded, or distributed.