

1 Number Representation

1.1 What is the range of integers represented by a n -bit binary number? Your answers should include expressions that use 2^n .

(a) Unsigned:

(b) Two's Complement:

(c) One's Complement:

(d) Bias (with bias b):

1.2 How many unique integers can be represented in each case?

(a) Unsigned:

(b) Two's Complement:

(c) One's Complement:

(d) Bias (with bias b):

2 C Memory Locations

2.1 Consider the following C program:

```
int a = 5;
int main()
{
    int b = 0;
    char* s1 = `cs61c";
    char s2[] = `cs61c";
    char* c = malloc(sizeof(char) * 100);
    return 0;
};
```

For each of the following values, state the location in the memory layout where they are stored. Answer with code, static, heap, or stack.

(a) s1

(b) s2

(c) s1[0]

(d) s2[0]

(e) c[0]

(f) a

- 2.2 Consider the C code here, and assume the malloc call succeeds. Rank the following values from 1 to 5, with 1 being the least, right before bar returns. Use the memory layout from class; Treat all addresses as unsigned numbers.

```
#include <stdlib.h>
```

```
int FIVE = 5;
```

```
int bar(int x) {
    return x * x;
}
```

```
int main(int argc, char *argv[]) {
    int *foo = malloc(sizeof(int));
    if (foo) free(foo);
    bar(10); // snapshot just before it returns
    return 0;
}
```

```
foo:  _____
```

```
&foo: _____
```

```
FIVE:  _____
```

```
&FIVE: _____
```

```
&x:  _____
```

3 Linked Lists

- 3.1 Fill out the declaration of a singly linked linked-list node below.

```
typedef struct node {
    int value;
    _____ next; // pointer to the next element
} sll_node;
```

- 3.2 Let's convert the linked list to an array. Fill in the missing code.

```
int* to_array(sll_node *sll, int size) {
    int i = 0;
    int *arr = _____;
    while (sll) {
        arr[i] = _____;
        sll = _____;
        _____;
    }
    return arr;
}
```

- 3.3 Finally, complete the function `delete_even()` that will delete every second element of the list. For example, given the lists below:

Before: Node 1 Node 2 Node 3 Node 4

After: Node 1 Node 3

Calling `delete_even()` on the list labeled "Before" will change it into the list labeled "After". All list nodes were created via dynamic memory allocation.

```
void delete_even(sll_node *s11) {
    sll_node *temp;
    if (!s11 || !s11->next) {
        return;
    }
    temp = _____;
    s11->next = _____;
    free(_____);
    delete_even(_____);
}
```

4 RISC-V to C

4.1 Assume we have two arrays input and result. They are initialized as follows:

```
int *input = malloc(8*sizeof(int));
int *result = calloc(8, sizeof(int));
for (int i = 0; i < 8; i++) {
    input[i] = i;
}
```

You are given the following RISC-V code. Assume register x10 holds the address of input and register x12 holds the address of result.

```
add x8, x0, x0
addi x5, x0, 8
addi x11, x0, 0
addi x12, x0, 0
```

Loop:

```
beq x5, x11, Done
lw x6, 0(x10)
add x8, x8, x6
slli x7, x5, 2
add x7, x7, x12
sw x8, 0(x7)
addi x5, x5, -1
addi x10, x10, 4
j Loop
```

Done:

```
// exit
.
```

```
// sizeof(int) == 4
int sum = 0;
```

4.2 What is the end array stored starting at register x12?

5 Advanced RISC-V

5.1 You are given the following RISC-V code:

```
Loop:   andi t2 t1 1
        srli t3 t1 1
        bltu t1 a0 Loop
        jalr s0 s1 MAX_POS_IMM
```

- (a) What is the value of the byte offset that would be stored in the immediate field of the bltu instruction?
- (b) What is the binary encoding of the bltu instruction? Please use hexadecimal to represent your answer.

5.2 As a curious 61C student, you question why there are so many possible opcodes, but only 47 instructions. Thus, you propose a revision to the standard 32-bit RISC-V instruction formats where each instruction has a unique opcode (which still is 7 bits). You believe this justifies taking out the funct3 field from the R, I, S, and SB instructions, allowing you to allocate bits to other instruction fields except the opcode field.

- (a) What is the largest number of registers that can now be supported in hardware?
- (b) With the new register size, how far can a jal instruction jump to (in halfwords)?
- (c) Assume register `s0 = 0x1000 0000`, `s1 = 0x4000 0000`, `PC = 0xA000 0000`. Lets analyze the instruction `jalr s0, s1, MAX_POS_IMM` where `MAX_POS_IMM` is the maximum possible positive immediate for `jalr`. Using the register sizes defined above, what are the values in registers `s0`, `s1`, and `pc` after the instruction executes?