Midterm Review Questions (without Answers)

The midterm exam will be broken into two components:

- A lab-based portion, on August 04, 2020
- A written (lecture-based) portion, on August 05, 2020

The lab-based portion will require you to write assembly code on your laptops or lab machines, which is to be turned in via Canvas by the end of the class. It will be similar in style to the rest of the assignments in the course.

The written portion will require you to:

- Understand number representation and numeric operations (from the first three labs)
- Read and understand assembly code
- Answer short-answer questions related to numeric operations and assembly

The lecture-based portion is heavily biased towards numeric representation, though you should expect some assembly-based questions.

You may bring the following materials into the exam:

- A calculator with exponentiation capabilities.
- The attached handout, which consists of the ARM reference card along with all the SWI codes you may need. A copy of this handout will be distributed at the beginning of the exam.

The review below, **in addition to everything you wrote for your labs**, is intended to be comprehensive. All topics which could potentially be on the exam are somehow covered by this review.

Questions

1. The leftmost bit of a 32-bit number is in what position?
2. Shifting an unsigned binary number `N` two positions to the left is equivalent to multipling `N` by what (in decimal)?
3. Shifting an unsigned binary number `N` four positions to the right is equivalent to performing truncating division (ignoring the remainder) by what (in decimal)?
4. For ANY unsigned binary number, which bit must you look at in order to determine if the number is odd or even?
5. What is `-8` in twos complement representation? Represent your solution using 8 bits.
6. What is `1 + 1` with a carry-in bit set?
7. What is `1 + 1` without a carry-in bit set?
8. What is `1 + 0` without a carry-in bit set?
9. What is:

```
  11111101
+ 01000101
```

   Specify if the result has a carry-out set and if the result sets the overflow bit.

10. What is:

```
  10010110
- 11101010
```

    Specify if the result has a carry-out set and if the result sets the overflow bit.

11. Consider an unknown binary number `N`. Using only bitwise operations and bitmasks, give an expression that will produce `N`, *except* that bit 7 is guaranteed to be one. Express any bitmasks using 2-digit hexadecimal.
12. While this isn't a review question, be familiar with the process to convert between binary and decimal floating point representations.
13. What is wrong with the following code, if anything?

```
  .equ Exit, 0x11
  .equ Open, 0x66
  .equ Close, 0x68
  .equ Read_Int, 0x6C

  .data
filename:
  .asciz "myFile.txt"

  .text
  .global _start
_start:
  ;; open the file
  ldr r0, =filename
  mov r1, #0
  swi Open

  ;; read an integer from it
  swi Read_Int

  ;; close the file
  swi Close

  ;; exit the program
  swi Exit
  .end
```

14. What is wrong with the following code, if anything?

```
  .equ Write_Int, 0x6B

  .text
  .global _start
_start:
  ;; print out 42
  mov r0, #1
  mov r1, #42
  swi Write_Int
```

15. Write ARM assembly code which will read two integers from the file `myFile.txt` and print them out.

16. Consider the following code, which sets up a `.data` section:

```
  .data
label1:
  .asciz "Hi"
label2:
  .word 1, 2
label3:
  .asciz "Bye"
```

Assuming the `.data` section starts at address `0`, how does this look in memory? Use the following table as a template.

Value																					
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

17. Convert the following Java/C-like code into ARM assembly. The names of the variables reflect which registers must be used for the ARM assembly.

```
if (r0 >= 5) {
  r1 = r6;
} else {
  r2 = r7;
}
```

18. Convert the following Java/C-like code into ARM assembly. **Use branch intructions instead of conditional execution.** The names of the variables reflect which registers must be used for the ARM assembly.

```
if (r5 < r6) {
  r2 = r3;
  print_string("Less");
} else if (r5 == r6) {
  r3 = r4;
  print_string("Equal");
} else {
  r4 = r5;
  print_string("Greater");
}
```

19. Convert the following Java/C-like code into ARM assembly. The names of the variables reflect which registers must be used for the ARM assembly.

```
for (int r2 = r1; r2 <= 150; r2 += 4) {
  int r3 = (r2 - 1) * (r2 + 1);
  print_int(r3);
  print_char('\n');
}
```

20. Convert the following Java/C-like code into ARM assembly. The names of the variables reflect which registers must be used for the ARM assembly. Non-register variable names indicate a value that should be stored in memory.

```
int[] myArray = new int[]{19, 21, -5, 4};
int r2 = 0;
```

```
int r3 = 0;
do {
  r2 += myArray[r3];
  r3++;
} while (r3 < 4);
print_int(r2);
```

21. Convert the following Java/C-like code into ARM assembly. The names of the variables reflect which registers must be used for the ARM assembly. Non-register variable names indicate a value that should be stored in memory.

```
int myArray[4] = {19, 21, -5, 4};
int* r2 = myArray;
int r3 = 4;
int r4 = 0;
do {
  r4 += *r2;
  r2++;
  r3--;
} while (r3 != 0);
print_int(r4);
```

22. Convert the following Java/C-like code into ARM assembly. The names of the variables reflect which registers must be used for the ARM assembly.

```
if (r2 < r3 && r3 < r4) {
  r5 = r6;
} else {
  r6 = r5;
}
```

23. Convert the following Java/C-like code into ARM assembly. The names of the variables reflect which registers must be used for the ARM assembly.

```
if (r2 < r3 || r3 < r4) {
  r5 = r6;
} else {
  r6 = r5;
}
```