CS 220 Exam 2 Study Questions

1. Work on the homework questions due before Spring break.
2. I may ask some questions from Exam 1.
3. What is the cycle time of a processor with a 1.33GHz clock rate (frequency). Express your answer in nanoseconds (ns).

   Cycle time and frequency are inversely related. So f = 1/t   or t = 1/f

   1/(1.33e9 GHZ) = 7.519e-10 sec = .7519e-9 sec = .75 ns.

4. A processor that has a 0.67ns clock rate has a frequency of ______________________. Express answer in MHz.

   .67 ns = .67e-9 sec   f = 1/.67e-9 = 598.8 MHz

3. Express **-120** as an 8-bit two's complement binary number.

   -120 = -128 + 8 = 10001000

4. Express **-120** as a 32-bit two's complement integer. Write your answer in hex.

   Need to sign extend with all ones, so 0xFFFFFF88

5. The formula for converting Celsius temperatures to Fahrenheit is ***F = 32 + C 9/5***

   a. Create a directory named **`c2f`** in your **CS220** repo.

   ```
   mkdir c2f
   ```

   b. In the **`c2f`** directory write a C file named **`c2f.c`** that implements a function named **`c2f`**. The function **`c2f`** takes an integer and returns an integer (not a double). Don't worry about these being integers and not doubles.

   ```
   int c2f(int c) {
       return 32 + c*9/5;
   }
   ```

   c. In the **`c2f`** directory write a C header file named **`c2f.h`** that contains a declaration for the **`c2f`** function.

   ```
   extern int c2f(int c);
   ```

d. Create a file named **main.c** that contains a **main** function that uses your **c2f** function. Your program should take the temperature being converted as a command line argument, call the **c2f** function, and print the result.

```
#include <stdio.h>
#include "c2f.h"
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("%d\n", c2f(atoi(argv[1])));
}
```

e. Write a file named **c2f.s** that implements the **c2f** function as an ARM assembly language function.

```
.cpu cortex-a53
.global c2f
.text

c2f:
    mov r1, #9          // need 9 and 5 in registers
    mov r2, #5          // for multiplication and division
    mul r0, r0, r1      // r0 = c*9
    sdiv r0, r0, r2     // r0 = r0/5
    add r0, r0, #32     // r0 = r0 + 32
    bx lr
```

f. Compile and test your program. Here are some sample runs from my implementation.

```
pi@raspberrypi:~/CS220Spring20/quiz2 $ ./c2f -40
-40 Celsius is -40 Fahrenheit

pi@raspberrypi:~/CS220Spring20/quiz2 $ ./c2f 100
100 Celsius is 212 Fahrenheit

pi@raspberrypi:~/CS220Spring20/quiz2 $ ./c2f 0
0 Celsius is 32 Fahrenheit
```

```
gcc -o c2f main.c c2f.s
```

6. The following C function computes x^y.

```
int xtoy(int x, int y) {
    int currsqr = x, rv = 1;

    while (y > 0) {
        if (y & 1)
            rv *= currsqr;
        currsqr *= currsqr;
        y >>= 1;
    }
    return rv;
}
```

   a. Make a table that traces the values of **currsqr**, **rv**, and **y** for each iteration of the loop when computing **xtoy(3,9)**

currsqr	rv	y
3	1	9
9	3	4
81	3	2
6561	3	1
43046721	19683	0

And the function returns 19683, the correct answer.

   b. Convert **xtoy** to an ARM assembly function.

```
.text
xtoy:
    mov r2, r0    // currsqr
    mov r3, #1    // return val
    push { r4}

while:
    cmp r1, #0       // while (y > 0)
    ble endwhile
    and r4, r1, #1   // if (y & 1)
    cmp r4, #0
    beq endif
    mul r3, r3, r2
endif:
    mul r2, r2, r2
    lsr r1, r1, #1   // logical shift right  y >>= 1;
    b while
endwhile:
    mov r0, r3
    pop { r4 }
    bx lr
```

7. What do each of the Linux commands do?

```
a. ls         list the files in the current directory
b. pwd        prints the present working directory
c. mkdir      makes a new folder (or directory)
d. ls -r      list files in reverse alphabetical order
e. cp         copy files
f. mv         move or rename files
g. ls ..      list the files in the parent directory
h. ls ../..   list the files in the grandparent directory
i. ls ./././  list the files in the current directory
j. ls dir/..  list the file in the current directory
```

8. Assume the following three function definitions of **f**, **g**, and **h** are in a file named **funcs.c**. Write an ARM assembly language version of the file that implements **f**, **g**, and **h**. Calling **f(1,2,3)** should return 48.

```
int h(int z) { return z * 2; }

int g(int x, int y) { return h(x + y); }

int f(int a, int b, int c) { return h(2) * g(a, b+c); }
```

```
h:
    lsl r0, r0, #1    // z * 2
    bx lr

g:
    push { lr }       // save link register because calling h
    add r0, r0, r1    // set up call to h with x + y
    bl h
    pop { lr }
    bx lr

f:
   push { r4-r7, lr }
   mov r4, r0         // save a
   mov r5, r1         // save b
   mov r6, r2         // save c
   mov r0, #2         // call h(2)
   bl h
   mov r7, r0         // save return result of h because calling g
   mov r0, r4         // set up call to g
   add r1, r5, r6
   bl g
   mul r0, r7, r0
   pop { r4 - r7, lr }
   bx lr
```

9. Consider the following variable declarations

```
int x = 77;
int *p = &x;
```

a. Which of the following are aliases for **x**

```
*x    illegal, x is not a pointer so it doesn't make sense to say *x

&x    x is an int and &x is a pointer, so they are not the same

p     p is a pointer and x is an int, so they are not the same

*p    *p and x refer to the same memory, so they are aliases

**p   illegal, p is a pointer, *p is an int, and you can't say **p

&p    p is a pointer and &p would then be a pointer to a pointer, so
      not the same as x.

*&p   &p is a pointer to a pointer, so *&p would be just a pointer, so
      not the same as x.

*&x   This is the same as x. So they are aliases
```

b. Which of the following are valid assignment statements. Here valid means that the compiler will not issue an error or warning. The compiler would give a warning if there was a type issue.

```
p = x;      // type error
x = p;      // type error
&x = 88;    // error, can't change the address of a variable
*p = 33;    // OK
x = 23;     // OK
p = NULL;   // OK
```