HOMEWORK III
ECE 3710 — Microcomputer Hardware and Software
Utah State University
Due date: **September 27, 2013**

Problem 1. Write a program that uses bit-banding to copy data from `PF.4` and `PF.3` to `PC.6` and `PC.7`, respectively (i.e. `PF.4` → `PC.6` and `PF.3` → `PC.7`).

Problem 2. Write a function that uses bit-banding and the `IT` instruction to check if a number placed at a given memory location within the bit-band is

(a) even

(b) divisible by four.

If a number is even the function should return `01`, if divisible by four `11`, else `00`. The memory address of the number is placed on the stack prior to the function being called and the result should be placed on the stack prior to the function returning.

Problem 3. (20 points) You've recently been hired by, let's say, the *No Such Agency* (NSA) to be part of a team building a block cipher encryption/decryption machine. Your task is to implement a routine that encrypts an 8-bit input. The location of the input data (known as the plaintext) is placed in `R0` before the function is called; the location for the encrypted data (known as the ciphertext) is to be placed in `R0` before the routine returns. The 8-bit encryption key is supplied on `PB`; i.e. you have to read `PB` to know the encryption key (this allows the user to enter an arbitrary key).

The encryption algorithm you have to use only encrypts 4-bits of data at a time. Here's how we perform the encryption. Allow M_1 to be the lower four bits of the 8-bit input M. Similarly, let K_1 denote the lower four bits of the encryption key K and K_2 its upper four bits. To encrypt M_1:

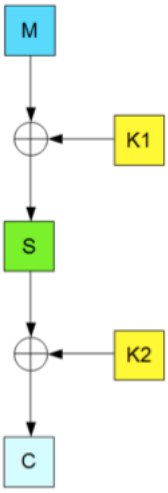
(a) XOR M_1 and K_1;

(b) use the table below to map the result of the XORing to its substitute; and

(c) XOR the substitute with K_2.

The same procedure is followed for the remaining 4 bits. The following figure illustrates this process (the 'S' refers to the table).

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	7	4	3	11	0	9	14	5	8	12	1	6	13	11	2	10



For example, to encrypt $M_1 = \texttt{0101}$ using $K = \texttt{00111100}$:

(a) `0101` XOR `1100` = `1001`;

(b) `1001` = 9 $\rightarrow$ 12 = `1100`; and

(c) `1100` XOR `0011` = `1111`.

HINT: use the `DCB` directive to define the output line of the table in memory. That way, you can add the starting address of the table to the value of the input to grab the correct output (think about copying the string in lab one). Also, use the simulator!

NOTE: there is a typo in the table. An input of 3 should map to an output of 15.

Problem 4. Making use of at least one instruction that uses the flexible second operand with a shift, write a program that reads from `PD.4--6` and writes to `PD.1--3` (i.e. `PD.4` $\rightarrow$ `PD.1`, `PD.5` $\rightarrow$ `PD.2`, and `PD.6` $\rightarrow$ `PD.3`). Your solution must not affect pins zero or seven.

Problem 5. (EXTRA CREDIT) Create a .lst by hand for the following code:

```
Start
  ldr  r3,=Start
  mov  lr ,r3
  cmp  r3 ,lr
  beq  func

func
  push {r3}
  ldr  r0 ,[SP,#12]
  bx  lr
```

NOTE: You are free to check your work using an assembler or simulator; however, you must show the steps you used to calculate the opcode and corresponding data, if any, for each instruction to receive credit. To force the assembler to generate 16-bit instructions you can use the `CODE16` directive.