#cec32xNotes, cec32x_ws_22_WS04_printing_in_binary_format

WS 4. Printing in Binary Format in C

Introduction

Here is a real programming problem we want to solve. The standard `printf` function in C can be used to print out an integer in the octal, decimal, and hexadecimal forms, but not the binary form. In this workshop, we solve this problem. Specifically, we write C functions that can be used to print an uint16_t integer in two different binary forms, the **compact** one and the **verbose** one, as detailed later.

Note that we don't need to use the Discovery Board for this workshop. The simulator in Keil is used for simplicity.

The basic idea is to convert an unsigned integer into a **string** of zeros and ones that represent the binary form of this number and then print out this string. For the conversion, please use the method we discussed in Section 1.1.2 of Module 4 of the class notes. Remember you have to end the sprint with a true 0 (not char "0").

Use the working code of Workshops 1 or 2 to get started. (This means that you need to use —with changed names—the project management files and the source files of Workshop 1 to save time. In addition, you need to use the following `main.c`, `print_binary.c`, and `print_binary.h` files in your project. You have to figure out how to add the new files to your project, including how to add a .c file to the project and put the .h file in the include path. (Ask if you have questions.)

The `main.c` file:

```c
#include <stdio.h>
#include "print_binary.h"

char str_c[30], str_v[30];

int main(void) {
    uint16_t test_num[] = {0x1234, 0x5678, 0x9ABC, 0xDEF0};
    int N = sizeof(test_num) / sizeof(test_num[0]);

    printf("Testing the printout of binary numbers:\n");
    for (int i = 0; i < N; i++) {
        printf("Compact display for 0x%04X is ", test_num[i]);
        print_str_compact(test_num[i], str_c);
        printf("\n");
        printf("Verbose display for 0x%04X is ", test_num[i]);
        print_str_verbose(test_num[i], str_c, str_v);
        printf("\n");
    }

    while (1);
}
```

Note that we have defined two global strings of 30 chars each:

- `str_c` is used to hold the compact version of the binary expression of a number using ASCII '0's and '1's. For example, if the integer is 0x0039, `str_c` should be `0000000000111001`. Note that you need to end the string with number 0 ('\0').
- `str_v` is used to hold the verbose version of the binary expression of the number. For example, the verbose version of the above compact expression is `0b0000_0000_0011_1001`.

The `print_binary.h` file:

```
#ifndef __PRINT_BINARY_H
#define __PRINT_BINARY_H

#ifdef __cplusplus
extern "C" {
#endif

////////////////////////////////////////////////
#include <stdint.h>

void convert_uint16_number_to_binary_str(uint16_t num, char *cPtr);
void print_str_compact(uint16_t num, char *cPtr);
void print_str_verbose(uint16_t num, char *cPtr_c, char *cPtr_v);

////////////////////////////////////////////////

#ifdef __cplusplus
}
#endif

#endif /* __PRINT_BINARY_H */
```

The `print_binary.c` file:

```
#include "print_binary.h"

void convert_uint16_number_to_binary_str(uint16_t num, char *cPtr) {
}

void print_str_compact(uint16_t num, char *cPtr) {
}

void print_str_verbose(uint16_t num, char *cPtr_c, char *cPtr_v) {
}
```

Workshop tasks

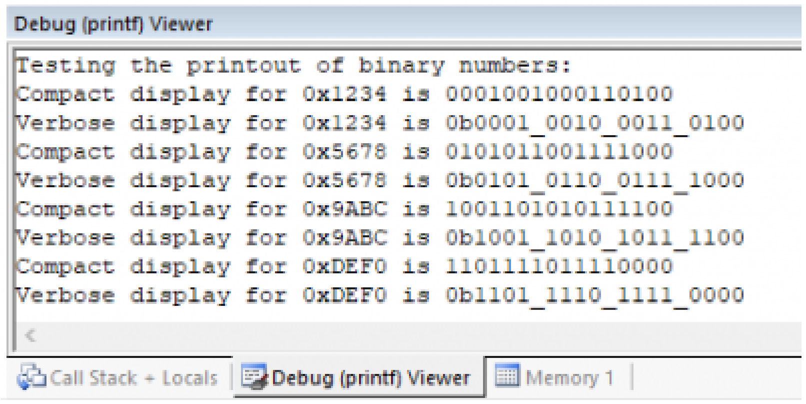
Below are the programming tasks which are all in the `print_binary.c`.

- (40 points) Write a function named `convert_uint16_number_to_binary_str` which converts a uint16 `num` to a string of binary bits and save the results to `cPtr`.
- (10 points) Write a function named `print_str_compact` which can be used to print the compact form of the binary string. Note that you need to call `convert_uint16_number_to_binary_str` in this function to perform the conversion and then print. Note that you cannot print the newline "\n" since we may join strings in the same line. This is the reason why we have used `printf("\n");` after calling this function.
- (40 points) Write a function named `print_str_verbose` which can be used to print the results in the verbose form. In this function, you need to call `convert_uint16_number_to_binary_str` as well. You then need to build `cPtr_v` from `cPtr_c`. Finally, you print out `cPtr_v` in a manner similar to `print_str_compact`.

You may want to perform this project using the following steps:

- First, get started by running an existing example project, such as the one from Workshop 1 or Lab 1. Move to the next step when the example project works.
- Then, you need to copy the code snippet given above and try to compile it. It should compile without any error. You can then run the project and see the printout results. Of course, you don't have the correct results yet.
- Next, you need to program the project step by step. Always program a couple of lines and then compile and debug. For this project, we mainly use `printf` to see the results. In the labs, we will play with true debug skills.

The running result should be similar to



```
Testing the printout of binary numbers:
Compact display for 0x1234 is 0001001000110100
Verbose display for 0x1234 is 0b0001_0010_0011_0100
Compact display for 0x5678 is 0101011001111000
Verbose display for 0x5678 is 0b0101_0110_0111_1000
Compact display for 0x9ABC is 1001101010111100
Verbose display for 0x9ABC is 0b1001_1010_1011_1100
Compact display for 0xDEF0 is 1101111011110000
Verbose display for 0xDEF0 is 0b1101_1110_1111_0000
```

printf_binary_format

Submission of your work

(10 points)

Each student needs to submit their own results—code snippet for the main function and the screenshots for the above printout results in the form of a pdf file. Put your name at the top of the page before the code snippet. Name your file as cec320_ws4_lastname_firstname(or initial).pdf. You also need to zip all your project files (not including the build files) into a single zip file and upload this file as well.

- Screenshots of your C functions.
- Screenshot of the running results.