


# Programming Lab 8B

## Making Change

 [Click to download Lab8B-Main.c](#)

Topics: Replacing division by reciprocal multiplication, replacing multiplication by a sequence of addition, subtraction and shift instructions, referencing structure members from assembly.

Prerequisite Reading: Chapters 1-8  
Revised: January 21, 2021

**Background:** This lab uses the two structures defined below to specify the number of bills of each of four different denominations and the number coins of each of four types:

```
typedef struct
{
    uint32_t    twenties ;
    uint32_t    tens ;
    uint32_t    fives ;
    uint32_t    ones ;
} BILLS ;

typedef struct
{
    uint32_t    quarters ;
    uint32_t    dimes ;
    uint32_t    nickels ;
    uint32_t    pennies ;
} COINS ;
```

**Assignment:** The main program will compile and run without writing any assembly. However, your task is to create equivalent replacements in assembly language for the following two functions found in the C main program. The original C versions have been defined as “weak” so that the linker will automatically replace them in the executable image by those you create in assembly; you do not need to remove the C versions. This allows you to create and test your assembly language functions one at a time.

Create two assembly language functions that fill-in the values of the structure members by finding the smallest number of bills that add up to `dollars` and the smallest number of coins that add up to `cents`. For example, the number of twenty-dollar bills may be computed as the integer quotient of  $dollars \div 20$ ; the remainder is then used to find the number of ten-dollar bills, and so on. You will soon discover most of the code in the two functions is identical; use this fact to reduce the amount of code you write.

```
void Bills(uint32_t dollars, BILLS *paper) ;
void Coins(uint32_t cents, COINS *coins) ;
```

**Important:** The objectives of this assignment are to (1) implement multiplication by a constant without a multiply instruction, and (2) implement division by a constant without a divide instruction.

Begin by implementing your functions using `MUL` and `UDIV` instructions to be sure that your algorithms work. Then replace each multiply and divide instruction one at a time (testing each as you do) with equivalent instruction sequences. No loops allowed!

Your final solution must **not** contain any divide instructions; instead, use [this](#) webpage to find instruction sequences to divide by a constant. The only multiply instructions allowed in your functions are those found in code produced by that program.

Test your code with the main program. If your code is correct, the display should look like the image shown; the initial amount displayed is selected randomly. Use the touchscreen to edit the amount to see how it is divided into bills and coins. Pressing the blue pushbutton sets the amount to zero. If your solution is incorrect, the total at the bottom will be displayed as **white text on a red background**.

