Homework Assignment #2 CS2600

John Casey(after Isailovic(UCB)) - Spring, 2012

Due date: Feb. 19

Using Blackboard, turn in solutions to this programming homework by Feb. 19.

A complete submission should include the files "mainloop.s", "maintest.s", "spf-main.s". If you don't understand any of these directions, ask me.

1 Background reading

P&H 2.1-2.4, 2.6-2.8, 2.13-2.14, 3.1 - 3.2 up to p. 227, and, in the appendix, pages B-22 to B-26.

2 Multiplication

Write the code for this question using only the following subset of MIPS instructions: add, sub, addi, lw, sw, beq, bne, slt, slti, j, jal, jr. (If you really, really want to, you can use sll.) In particular, you are not allowed to use mult or div.

To make it easier to write the code, you can use pseudo-instructions like bge (**B**ranch on **G**reater or **E**qual). You can find the branch pseudo-instructions on pp. B-59 – B-63.

Note: Be sure to translate the code as is. Do not take shortcuts or otherwise make modifications to it in order to optimize. The whole point is to watch this code run.

You can get a copy of all the files you'll need by doing:

```
mkdir    myhw2
cp    /course/cs2600jc/.www/hw2/*    myhw2
```

2.1 Multiplying by using a loop

Make sure you have a copy of the file "mainloop.s". When you have written your program, type it up at the end of "mainloop.s".

Translate the following C code into MIPS. You must follow the register conventions as usual.

```
/*     Multiply a number by another between 0 and 255, */
/*     using a loop.                                   */

int multloop (int multiplicand, int multiplier) {

    int product;

    if (multiplier > 255) product = -1;
    else if (multiplier < 0) product = -1;
    else {
        product = 0;
        while (multiplier > 0) {
            product += multiplicand;
            multiplier -= 1;
        }
    }
    return product;
}
```

When it's working, hand in "mainloop.s", with the rest of the assignment.

2.2 Multiplying by using tests

Make sure you have a copy of the program "maintest.s" When you have written your program, type it up at the end of "maintest.s".

Translate the following C code into MIPS. You must follow the register conventions as usual.

```
/* Multiply a number by another between 0 and 255, using tests. */
int multtest (int multiplicand, int multiplier) {

    int product;
    int m1, m2, m3, m4, m5, m6, m7;

    m1 = 2*multiplicand;
```

```
m2 = 4*multiplicand;
m3 = 8*multiplicand;
m4 = 16*multiplicand;
m5 = 32*multiplicand;
m6 = 64*multiplicand;
m7 = 128*multiplicand;

if (multiplier > 255) product = -1;
else if (multiplier < 0) product = -1;
else {
    product = 0;
    if (multiplier >= 128) {
        product += m7;
        multiplier -= 128;
    }
    if (multiplier >= 64) {
        product += m6;
        multiplier -= 64;
    }
    if (multiplier >= 32) {
        product += m5;
        multiplier -= 32;
    }
    if (multiplier >= 16) {
        product += m4;
        multiplier -= 16;
    }
    if (multiplier >= 8) {
        product += m3;
        multiplier -= 8;
    }
    if (multiplier >= 4) {
        product += m2;
        multiplier -= 4;
    }
    if (multiplier >= 2) {
        product += m1;
        multiplier -= 2;
    }
```

```
        if (multiplier >= 1) {
            product += multiplicand;
            multiplier -= 1;
        }
    }
    return product;
}
```

When it's working, hand in "maintest.s", with the rest of the assignment.

3 Longer exercise: sprintf in MIPS

This longer exercise is also due Feb. 19

Write your solution to this question in a file named "sprintf.s". Write a MIPS implementation of the standard C library function sprintf:

```
int sprintf (char *outbuf, char *format, ...)
```

There is documentation for this function on page 155, and also on page 245 of K&R.

sprintf turns the characters that would be printed by a corresponding printf into a string. (C++'s string streams provide another version of this facility.)

Your function must accept up to four arguments, all passed in registers $a0 to $a3. The first argument is the address of a character array into which your procedure will put its results. The second argument is a format string in which each occurrence of a percent sign ('%') indicates where one of the subsequent arguments is to be substituted and how it is to be formatted.

The remaining arguments are values that are to be converted to printable character form according to the format instructions. sprintf returns the number of characters in its output string (not including the null at the end).

You do not have to implement all of the formatting options of the real sprintf (K&R, pp. 244–245). Here are the ones you must implement:

%d convert integer to decimal

%x convert integer to hexadecimal

%c include one character argument in result
%s include string of characters in result
%% include a percent sign in result

Don't implement width or precision modifiers (e.g., format specifier which you don't have to support, e.g.,

3.1 Background

The procedure-calling convention we've been using up to now uses four registers ($a0 - $a3) for passing arguments down to procedures. We'll stick to that convention for the sprintf project, so your code will not have to handle calls to sprintf with more than four parameters.

3.2 Miscellaneous requirements

For this project, the return value should be in $v0. How to program your sprintf:
Take a copy of spf-main.s and type your code where ther is a stub for the function sprintf().

This main program is online in /course/cs2600jc/.www/hw2; you can copy it to the directory you're working in.

When it's working, hand in "spf-main.s" with the rest of the assignment.