Homework Assignment #2 CS U380

John Casey(after Isailovic(UCB)) - Spring, 2009

Due date

Turn in solutions to all of this programming homework by Sunday, Feb. 22.

Place your answers to all the questions below in a single file called "hw2.txt", unless the question says to do otherwise. We'll tell you how to hand in that file. A complete submission should include the files "hw2.txt", "loopscript", and "testscript". If you don't understand any of these directions, ask us.

1 Background reading

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

2 Short exercises and small functions

2.1 P&H

Problems 2.4.1 - 2.4.5

2.2 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.

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 observe this code. You can use programs we wrote to help test your code.

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

```
mkdir    myhw2
cp    /course/csu380jc/hw2/*    myhw2
```

2.2.1 multloop

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 you have written your function, you can test it by

```
1. Open ''mainloop.s''
2. Assemble
3. Run
```

2.2.2 multtest

Make sure you have a copy of the file "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 you have written your function, you can test it by

```
1. Open ''maintest.s''
2. Assemble
3. Run
```

Make a script and hand it in, just as you did for multloop.

3 Longer exercise: sprintf in MAL

This longer exercise is also due Sunday, Feb. 22

Write your solution to this question in a file named "sprintf.s". Write a MAL 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 any number of arguments, all passed in registers as described below. 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., you should just output f as a way of handling it gracefully.

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. Also, your sprintf procedure should work with the main program in file "spf-main.s".

Make sure you have a copy of the file "spf-main.s". When you have written your program, type it up near the end of "spf-main.s"

When you have written your function, you can test it by

```
1. Open ''maintest.s''
2. Assemble
3. Run
```

Make a script and hand it in, just as you did for multloop.

4 Longer exercise: sprintf in MAL

This longer exercise is also due Sunday, Feb. 22

Make sure you have a copy of the file "spf-main.s". When you have written your function, type it up near the end of "spf-main.s".

Write a MAL 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 any number of arguments, all passed in registers as described below. 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

When you have written your function, you can test it by

```
1. Open ''spf-main.s''
2. Assemble
3. Run
```

This main program is online in /course/csu380jc/hw2; you can copy it to the directory you're working in.

When everything is working, make a script, name it "sprintfscript", and hand it in.