CSCI 2321 (Computer Design), Spring 2017

Homework X1

Credit: Up to 30 extra-credit points. (But be advised that you can't get more than 50 extra-credit points on this assignment and Homework X2 combined.)

1 Overview

This set of extra-credit problems covers material from the first part of the course. You can do as many as you like, but you can only receive a total of 50 extra points on this assignment and Homework X2 combined.

NOTE that the usual rules for collaboration do not apply to this assignment. More in the following section.

2 Honor Code Statement

Please include with each part of the assignment the Honor Code pledge or just the word "pledged", and the statement "This assignment is entirely my own work" (where "my own work" means "except for anything I got from the assignment itself, such as starter code, or from the course Web site or sample solutions to other assignments). For this assignment you should *not* work with or seek help from other students or from tutors, but you can consult other sources (other books, Web sites, etc.), as long as you identify them.

3 Problems

Answer the following questions. You may write out your answers by hand or using a word processor or other program, but please submit hard copy, either in class or in one of my mailboxes (outside my office or in the ASO).

1. (Optional: Up to 10 extra-credit points.) For this problem your mission is to reproduce by hand a little of what an assembler and linker would do, as you did in the last problem of Homework 3. So you are to do two phases:

   - Assembly, in which you produce for each input file the following:
     - Sizes of text (code) and data segments, in hexadecimal. (*Correction/clarification:* Remember that `la` is a pseudoinstruction that expands to a combination of `lui` and `ori`. `li` is also a pseudoinstruction that potentially expands to the same pair, but it looks like SPIM's built-in assembler produces both instructions only if the immediate value being loaded is more than 16 bits; for small values it expands it just to an `ori`. You should do likewise. (So, as long as the value being loaded is "small enough", the assembler expands `li` to just `ori`.)
     - "Relocation information", as in Homework 3 — a list/table with one entry for each instruction that needs to be patched based on where in memory the program needs to be loaded.

  - A symbol table with entries for all symbols, showing for each its name, which segment it's in (text or data), and its offset into that segment.

- Linking, in which you produce:
  - Sizes of combined text (code) and data segments, in hexadecimal.
  - A symbol table showing locations of all symbols and their addresses.
  - Patched versions of all the instructions from all the "relocation information" segments from the assembly phase, in the form described in Homework 3.

The input files are these:

- `main.s`:

```
        .text
        .globl  main
main:
# opening linkage
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)
# prompt and get two integers from "console"
        la      $a0, prompt
        li      $v0, 4          # "print string" syscall
        syscall
        li      $v0, 5          # "read int" syscall
        syscall
        la      $t0, dataX
        sw      $v0, 0($t0)     # save result in dataX
        li      $v0, 5          # "read int" syscall
        syscall
        la      $t0, dataY
        sw      $v0, 0($t0)     # save result in dataY
# call procedure to add and print
        la      $a0, dataX
        la      $a1, dataY
        jal     foobar
# closing linkage
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
        jr      $ra
        .end    main
# variables and constants
        .data
prompt: .asciiz "Enter two integers, one per line:\n"
# note that .word forces alignment -- i.e., causes assembler to insert
#   space if not on a word boundary (address a multiple of 4)
dataX:  .word 0
dataY:  .word 0
```

- `foobar.s`:

```
        .text
```

```
        .globl  foobar
foobar:
# add two integers and print result
# $a0, $a1 have addresses of two integrs
# opening linkage
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)
# compute result into $s0
        lw      $t0, 0($a0)
        lw      $t1, 0($a1)
        add     $s0, $t0, $t1
# print
        addi    $a0, $s0, 0
        li      $v0, 1          # "print int" syscall
        syscall
        la      $a0, foobar_nl
        li      $v0, 4          # "print string" syscall
        syscall
# closing linkage
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
        jr      $ra
# variables and constants
        .data
foobar_nl: .asciiz "\n"
```

and for the link step you should assume:

- Object code for `main.s` is loaded first, then object code for `foobar.s`.
- The combined text segment starts at `0x04000000`.
- The combined data segment starts at `0x10000000`.

4 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu` with each file as an attachment. Please use a subject line that mentions the course and the assignment (e.g., "csci 2321 hw X1" or "computer design hw X1"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (Optional: Up to 10 extra-credit points.) For this problem, you are to write a MIPS procedure that, given a (null-terminated) string, tries to convert it to a signed integer and reports success/failure. More explicitly, this procedure should get the address of the string as the first argument (in `$a0`) and produce two results:

   - An "error code" in `$v1`), where 0 means success, -1 means the string doesn't represent a signed integer, and -2 means the conversion wasn't possible because it would cause overflow.

- If there was no error, `$v0` should contain the result of the conversion.

So "10", "-20", and "2147483647" ($2^{31} - 1$) are all valid, but "10-", "abcd", "10ab", and "2147483648" (2^{31}) are not. To get maximum points you need to detect both kinds of errors, but you can get up to 8 points if you do everything except the check for overflow. Other "corner cases" include the empty string and "-", both of which should produce an error result (-1), but here too if you don't make that work you won't lose many points.

Starter program test-convert-int.s[1] contains code to prompt the user for a text string, read it, call the convert procedure, and print the results. Your mission is to fill in the body of the convert procedure so it works as described.

Sample executions:

```
% spim  -f test-convert-int.s
Loaded: /usr/share/spim/exceptions.s
Enter a line of text:
10
Input 10
Result 10

% spim  -f test-convert-int.s
Loaded: /usr/share/spim/exceptions.s
Enter a line of text:
-20
Input -20
Result -20

% spim  -f test-convert-int.s
Loaded: /usr/share/spim/exceptions.s
Enter a line of text:
abcd
Input abcd
Error -1

% spim  -f test-convert-int.s
Loaded: /usr/share/spim/exceptions.s
Enter a line of text:
1000000000000
Input 1000000000000
Error -2
```

HINTS:

- Note that SPIM seems happy to accept character literals in the same format as C, so for example you can put the ASCII characters for 0 in a register by writing

  ```
  li $t0, '0'
  ```

  and the same thing works for other characters, such as the null character:

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS2321_2017spring/Homeworks/HW0X1/Problems/test-convert-int.s`

```
li $t0, '\0'
```

I strongly advise that you do this rather than looking up ASCII values and putting them in your code: MIPS assembly code is hard enough to read already, and using the ASCII values directly just makes it worse.

- Think about the algorithm first, but if nothing occurs to you, see this footnote[2].

2. (Optional: Up to 10 extra-credit points.) For this problem, you are to write a MIPS procedure that, given a memory address `p` and a number of bytes `n`, prints hexadecimal representations of `n` bytes starting at `p`. So for example if the `p` points to a "ab" and `n` is 2, the procedure should print "61 62" (hexadecimal representations of ASCII values for 'a' and 'b'), while if `p` points to an integer (in memory) with value 5 and `n` is 4, the procedure should print "05 00 00 00" (why is the 5 first? SPIM is little-endian, so bytes in integer types are stored in reverse order). More explicitly, this procedure should get `p` as the first argument (in `$a0`) and `n` as the second argument (in `$a1`) and print (to the "console", using SPIM system calls) as described. It doesn't need to return anything in `$v0` and `$v1`.

   Starter program test-print-hexbytes.s[3] contains code to prompt the user for a text string, read it, call the procedure to print the whole buffer, and then prompt for an integer, read it, and call the procedure to print the 4-byte result. Your mission is to fill in the body of the print procedure so it works as described. Sample execution:

```
% spim -f test-print-hexbytes.s
Loaded: /usr/share/spim/exceptions.s
Enter a line of text:
abcd
Input abcd
Result 61 62 63 64 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Enter an integer:
20
Input 20
Result 14 00 00 00
```

   HINTS:

   - You will probably want to use the `lb` instruction ("load byte") to work with individual bytes.
   - One way to do the conversion is to split the resulting byte into two half-bytes (each representing one hex digit) and then use those as indices into a string containing all the hex digits (`"0123456789abcdef"`).

[2] You could do it in C thus, assuming `p` starts out pointing to the beginning of the string (note that this doesn't do any error checking, but you can figure that out?

```
/* put result of conversion in "work", ignoring errors */
int work = 0;
while (*p != '\0') {
    work = work*10 + (*p - '0');
    ++p;
}
```

[3] `http://www.cs.trinity.edu/~bmassing/Classes/CS2321_2017spring/Homeworks/HW0X1/Problems/test-print-hexbytes.s`

- SPIM has a "print character" system call that you wil probably find useful.