CSCI 2321 (Computer Design), Spring 2020

Homework 4

Credit: 30 points.

1 Reading

Be sure you have read, or at least skimmed, all assigned sections of Chapter 2 and Appendix A.

2 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. (15 points) (*Note:* This problem may initially look intimidating, but if you take it step by step I think you will find it manageable.)

   For this problem your mission is to reproduce by hand a little of what an assembler and linker would do with two fairly meaningless[1] pieces of MIPS assembly code. The textbook has an example starting on p. 127 illustrating more or less what I have in mind here, and we reviewed the example in class, but on reflection it doesn't seem that clear to me, so for this assignment I want you to approach the problem a little differently.

   First, the two files, one containing a main procedure:

```
            .text
            .globl main
    main:
            addi    $sp, $sp, -4
            sw      $ra, 0($sp)
            jal     subpgm
            lw      $ra, 0($sp)
            addi    $sp, $sp, 4
            jr      $ra
            .end    main

            .data
            .globl dataX
    local:  .word   0
    dataX:  .word   1, 2
```

   and another a procedure it calls:

```
            .text
            .globl subpgm
```

[1] They don't do anything very interesting, but together they do represent a complete program.

```
subpgm:
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)
# copy data (two "words") from dataX to dataY
        la      $s0, dataX
        la      $s1, dataY
        lw      $t0, 0($s0)
        sw      $t0, 0($s1)
        lw      $t0, 4($s0)
        sw      $t0, 4($s1)
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
        jr      $ra
        .end    subpgm

        .data
        .globl dataY
dataY:  .space  8
```

For the "assembly" phase, I don't want you to actually translate the instructions into machine language, but I do want you to construct for each file a table with information as listed below. *Note* that you will need to expand the two `la` pseudoinstructions. The example in the textbook doesn't really show how to do this; they instead show how to deal with `lw` and `sw` referencing a symbol and assembled into something using the `$gp` register.[2] Instead I want you to expand these instructions in the way SPIM does: each as a `lui` followed by a `ori`. (You can see examples of this by loading any of the sample programs that use `la` into SPIM and looking at what it shows for code.)

(*Hint:* Before going further, you'll probably find it useful to write down, for each of the two files, what's in its text segment (a list of instructions and their offsets, remembering to expand any pseudoinstructions), and what's in its data segment (a list of variables/labels and their offsets and sizes).)

Then produce, for each of the two source files, a table with the following. (Use hexadecimal to represent addresses and offsets.)

- Text (code) and data sizes, in hexadecimal.
- "Relocation information": For each instruction that involves an absolute address (jumps and the instructions corresponding to a `la` pseudoinstruction):
  - Its offset in the text segment.
  - The instruction type (as in the textbook example).
  - The symbol referenced ("dependency" in the textbook example).
- A symbol table listing all symbols, showing for each:
  - Its name.
  - Which segment it's in (text or data) and its offset into that segment.

  For example. the first symbol in the first file is `main`, at offset 0 into the text segment.

[2] I'm not quite sure how they get this from MIPS assembly source; SPIM will accept load/store instructions referencing a label, but it turns them into `lui`/`ori` pairs in the same way it does for `la`.

(A real assembler would probably try to resolve references to local symbols at this point, but for simplicity I want you to just resolve them all in the next step.)

Next, "link" these two files to produce information for an executable *for the SPIM simulator*. Since programs in this simulator always have their text segments at `0x00400024` and their data segments at `0x10010000`, absolute addresses into either segment can be based on these values. (Normally an executable file might include "relocation information" for any instructions containing absolute addresses that would need to be changed when the program is loaded into memory, but we'll skip that.)

(*Hint:* Note that the text segment of the executable is just the text segment for the first file followed by the one for the second file, and similarly for the data segment. So you'll probably find it useful to come up with a list of what's in each segment, similar to what you did in the first step, but with addresses rather than offsets.)

The information I want is this:

- Text (code) and data size, in hexadecimal.
- A symbol table showing locations of *all* symbols and their addresses (e.g., `main` is at `0x00400024`). (Really I think this should just be the global symbols, but to patch the unresolved references you'll need some non-global labels, and this is the simplest way to achieve that.)
- Patched versions of the instructions from the object files' "relocation information" sections, in the form of another table, one entry per instruction, with:
  - The instruction's address.
  - The patched instruction, in a form that looks like source code but doesn't reference labels — so for example a `j main` would become `j 0x00400024`. (Use hexadecimal for the constant/immediate values here.)

For an example of what I have in mind, see this directory.

3 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 my TMail address (or you can use `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 4" or "computer design hw 4"). 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. (15 points) Problem 2.31 from the textbook asks you to write a MIPS implementation of a recursive function `fib` to compute elements of the Fibonacci sequence. For this problem, your mission is to write this MIPS function and incorporate it into a complete program that, run from SPIM, prompts for an integer value N, calls `fib` to compute the N-th element of the sequence, and prints the result. Programs `factorial-recursive.s` on the sample programs page may be helpful, since it shows how to do the needed input/output and also contains an example of a recursive procedure written in MIPS. *To get full credit, your program must use recursion, and any functions/procedures you define must follow the conventions described in the textbook and in class for passing arguments and saving/restoring registers.*

4 Honor Code Statement

Include the Honor Code pledge or just the word "pledged", plus *at least one of the following* about collaboration and help (as many as apply).[3] Text *in italics* is explanatory or something for you to fill in. For programming assignments, this should go in the body of the e-mail or in a plain-text file `honor-code.txt` (no word-processor files please).

- This assignment is entirely my own work. *(Here, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site. In particular, for programming assignments you can copy freely from anything on the "sample programs page".)*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

5 Essay

Include a brief essay (a sentence or two is fine, though you can write as much as you like) telling me what about the assignment you found interesting, difficult, or otherwise noteworthy. For programming assignments, it should go in the body of the e-mail or in a plain-text file `essay.txt` (no word-processor files please).

[3] Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.