CSCI 2321 (Principles of Computer Design), Spring 2013

Homework 3

Credit: 30 points.

1 Reading

Be sure you have read all assigned sections of chapter 2.

2 Problems

Do the following problems. 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 my mailbox in the department office.

1. (10 points) Do problems 2.19.1 and 2.19.3 from the textbook. (For problem 2.19.1, just give MIPS code; no need to count instructions.) Use the following C code.

```
int fib(int n){
        if (n==0)
                return 0;
        else if (n==1)
                return 1;
        else
                return fib(n-1) + fib(n-2);
}
```

   Clarification: The conventions for calling a recursive function are somewhat unclear — the example in chapter 2 differs a bit from a similar example in appendix B. Use whichever makes more sense to you.

   Clarification: For problem 2.19.3, show what is on the stack just after the first call to function `fib` finishes its "opening linkage" (saving registers on the stack).

2. (5 points) Do problem 2.27.4 from the textbook. Do the problem twice, once for each of the following sequences of MIPS instructions (where the base-16 number on the left represents the address of the instruction).

```
        0x00400000              beq     $s0, $0, FAR
        ...
        0x00403100 FAR:         addi    $s0, $s0, 1
```

   and

```
        0x00000100              j       AWAY
        ...
        0x04000000 AWAY:        addi    $s0, $s0, 1
```

3. (10 points) Do problem 2.31.1 from the textbook (producing something like the answer to the example on pp. 143ff — i.e., show the result of everything the linker must do). Use the text and data sizes from the problem and the object-file information in the tables below.

   Clarification: If you are puzzled by the `lui` and `ori` instructions: These two instructions are often used together to load a 32-bit constant (example on p. 128 of the textbook). Very likely the intent here is to load the address corresponding to X into register `$a0`, since The MIPS assembler transforms the pseudoinstruction `la` ("load address") into just such instructions.

   Clarification: In the referenced example there is a discussion of how to "patch" the `lw` and `sw` instructions. I'm quite skeptical about the calculation of the offset/displacement for the `sw` — I believe that the result is correct, since if you sign-extend 0x8020 and add it to the value in `$gp` you do get the address corresponding to Y. The intermediate value (negative 0x7980) seems wrong, though. The simplest way to approach this calculation may be just to guess (and check) a value that when added to the value in `$gp` gives the desired address.

   Table for procedure A:

Text Segment	Address	Instruction	
	0	`lui $at, 0`	
	4	`ori $a0, $at, 0`	
	...	...	
	0x84	`jr $ra`	
	...	...	
Data Segment	Address	Label	
	0	(X)	
	...	...	
Relocation Info	Address	Instruction Type	Dependency
	0	`lui`	X
	4	`ori`	X
Symbol	Address	Symbol	
	...	X	

   Table for procedure B:

Text Segment	Address	Instruction	
	0	`sw $a0, 0($gp)`	
	4	`jmp 0`	
	...	...	
	0x180	`jal 0`	
	...	...	
Data Segment	Address	Label	
	0	(Y)	
	...	...	
Relocation Info	Address	Instruction Type	Dependency
	0	`sw`	Y
	4	`jmp`	FOO
	0x180	`jal`	A
Symbol	Address	Symbol	
	...	Y	
	0x180	FOO	
	...	A	

4. (Optional: Up to 5 extra-credit points) Do problems 2.28.2, 2.28.3, and 2.28.4 from the textbook. Use the following MIPS instructions for problems 2.28.2 and 2.28.3, and the table below for problem 2.28.4. (Assume here that `R2`, `R3` are registers. No, I don't know why the textbook authors didn't follow the usual convention ...)

```
try:    MOV     R3, R4
        LL      R2, 0(R2)
        ADDI    R2, R2, 1
        SC      R3, 0(R1)
        BEQZ    R3, try
        MOV     R4, R2
```

Table for 2.28.4:

Processor 1	Processor 2	Cycle	Processor 1		Mem	Processor 2	
			`$t0`	`$t1`	(`$s1`)	`$t0`	`$t1`
		0	1	2	99	30	40
`ll $t1, 0($s1)`		1					
	`ll $t1, 0($s1)`	2					
	`addi $t1, $t1, 1`	3					
	`sc $t1, 0($s1)`	4					
`sc $t1, 0($s1)`		5					

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 `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 2321 homework 3"). You can develop your programs on any system that provides the needed functionality, but I will test them using the SPIM simulator on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (5 points) Add code to your solution to problem 2.18.2 (from Homework 2) to make it a complete program that prompts for values for `a` and `b` and prints the ending value of `a`. Programs `echo.s` and `echoint.s` on the sample programs page show how to input and output text and integer values.

2. (Optional: Up to 10 extra-credit points.) First do problem 2.23 from the textbook, using the condition that the input ASCII string is meant to represent a positive hexadecimal integer (so for example "1234" and "5A" are valid inputs, but "hello" is not, nor is "-1"). Then add code to the resulting function so that the result is a complete MIPS program that prompts for an input string, converts it to an integer using your function, and prints the result (in base 10). Programs `echo.s` and `echoint.s` on the sample programs page show how to input and output text and integer values.