University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 1997 D.A. Patterson
Handout #3

Assignment 2
Exercises Due Monday 9/15/97 by 4PM in box in 283 Soda and Lab (programming assignments) Due in Discussion Section on 9/18/17
CS152 Computer Architecture and Engineering

PLEASE put the TIME OR TA NAME of the DISCUSSION section that you attend as well as your name, as homeworks will be handed out in discussion. No section, no homework.

Homework Policy: Homework assignments are due at 4PM on Mondays. No late homeworks will be accepted, as we want to be able to discuss the assignments in section. Study groups are encouraged, but what you turn in must be your own work. As we agreed on the first day of class, the penalty for cheating is no credit for the full assignment. (If things get tough, its better to skip pieces of one assignment rather than take a chance on getting no credit.)

Every student should turn their own solution to the following problems by 9/15/97 from P&H: 3.14, 3.19, A.3

The first two lab problems are to be done individually, since it is very important that everybody review the MIPS instruction set. The third problem can be done in teams of two, with the teams to be randomly chosen in discussion section. If you miss your section, you'll probably have to do it on your own: hence the maximum size of a team is two, and some courageous few may do it by themselves. **This is a demanding assignment, which is why you are given time before the lab is due 9/18/97. Don't postpone until the night before the deadline!**

The software for this assignment is located in `~cs152/fa97/hw2`. You should be sure that `~cs152/fa97/bin` is on your path so that you will get correct executables for the DECstations or HPs.

Problem 1: Introduction to SPIM

This problem will give you a chance to get familiar with SPIM, a MIPS simulator, the MIPS instruction set, and the calling convention. You will learn SPIM's basic functionalities by simulating a MIPS assembly program that calculate the factorial of 5. You should read chapter 3 and appendix A to get prepared for this assignment. An on-line manual for SPIM can be found at `http://www-inst.eecs.berkeley.edu/~cs152/handouts/spim_documentation.ps`.

Copy `~cs152/fa97/hw2/fact.s` to your working directory for this assignment, say `~/cs152/hw2`. This program uses recursion to evaluate the factorial of 5 ($f(N) = N * f(N-1)$, $f(0) = 1$). Execute SPIM by typing `xspim.dec` or `xspim.hp`, depending on the type of machine you use. (This X version of SPIM is a much easier program to learn and use than the text version.) You will see that the default "start up code" appears in text segments window. Load the assembly program `fact.s` by pressing the load button, typing the file name in the dialogue box, and clicking on assembly program. Observe that the instructions for your Main program now appear in the lower portion of the Text Segments pane. You are using the default mode of SPIM which supports the extended instruction set and has no delayed jumps or delayed loads.

1.a: The program does not look exactly the same as in the `fact.s` file, because the some of the instructions are macros and SPIM translates them to bare machine instructions. However, the original codes are displayed as comments. What do the instructions "li" and "la" mean? How are they translated to bare machine instructions? Turn-in an annotated version of `fact.s` that shows how the "virtual machine" instructions are expanded or translated into native instructions. Also, annotate this with the C code that would produce the instructions.

1.b What is the starting address of your main routine? of Fact? Single-step the code from start until it reaches the first instruction of your main routine. Note that R31 has been modified. Which is the instruction that modified R31? What is the purpose of doing this?

1.c Set a breakpoint at the first instruction of your fact routine. Continue tracing through the program execution by using single stepping and breakpoints. What are the registers `sp` and `fp` used for? Draw the values on the stack when the first instruction of the `fact` procedure is about to be executed for the fourth time.

1.d: Without changing the logic of the program, improve the assembly language version by eliminating unnecessary data movement and changing register accordingly. Turn in your improved program. Use spim (spim.dec or spim.hp), the simple version of xspim, to produce a transcript showing the key stages in the program with some comments on which values matter. (Please try to keep it short.)

1.e: Convert your `fact` program to use a simple loop, rather than recursion, to compute the factorial. Show that it works.

1.f: Now it is time to switch from the virtual machine to the bare hardware. To run the bare machine execute `xpsim.dec -bare` or `xpsim.hp -bare`. This turns off the assembly language translations and imposes delayed jumps (and branches) as well as delayed loads. Convert `fact.s` to use only the bare machine by translating the non-native instructions and by inserting NOPs after the jumps, branches, and loads. (You will need to convert the call in main to fact back into a simple JAL.) Turn-in this version along with verification that it works.

1.g: Eliminate as many NOPS as possible by moving instructions around the branch or load, while retaining the logic of the program. Turn-in the improved (working) version with an explanation of the changes and verification.

Problem 2: MIPS assembly programming

2.a: Write a MIPS assembly program to compare two integer arrays. It should leave the value 0 in

R2 if the two arrays are equal, 1 otherwise. The length of the arrays should be a parameter. Your program should follow the register usage and procedure calling conventions described in Appendix A of P&H. Test your program with equal and unequal arrays of length five. Turn in the code as well as a transcript showing your program works.

2.b: Several assembly language instruction are expanded into a short sequence of machines instructions by the assembler. Show the expansions that would be used for the following instructions described in appendix A: abs, mul, neg, rem, rol, seq, bgequ, bgt, ld.

Problem 3: Writing diagnostic programs

Now we get to the hard part! You are given a buggy version of SPIM, called broken-spim. It has 3 bugs in it. Your task is to write a set of diagnostic programs to isolate the bugs. (You will verify lots of other things work. Turn this in too.) You will need to work with the bare machine for this. Here are a list of things you should check for:

1. Register zero always has the value zero.
2. All the branch and jump instructions should jump to the right address under the right condition.
3. All the branch and jump instructions with link should put the correct return address into the right register.
4. SLT(I) and STLU(I) must work properly with bit patterns representing negative two's complement integers or large natural numbers.
5. All the arithmetic and logical instructions with immediate field should extend it properly.
6. Make sure a load that follows a store to the same address reads the appropriate data.
7. The correct endianess should be used on data transfer instructions.
8. Jump and branch instruction should have a delay slot.
9. Overflow should be detected correctly in arithmetic and logical instructions.

You should make sure that the fundamental things work before you test some other things that depend on them. You can build a test program with a sequence of tests, once you have verified that simple compares and branches work, by branching to a "failure label" with an identifier for the failed test in a register. As long as there are no failures, it sequences through tests using instructions that have been previously verified to execute correctly.

Writing diagnostic programs can deepen your understanding of the instruction sets. But the purpose of this assignment is not only to learn MIPS and understand the diagnostic process, but to create programs that you can use to validate your design later in the semester. Since the project only requires you to implement only a subset of the MIPS instruction set, the diagnostic programs in this assignment will only be useful later if they are written using this limited set of instructions. **Hence you must write the diagnostics using only the MIPS instructions below:**

type	instruction
arithmetic	addu, subu, addiu
logical	and, or, xor, andi, ori, xori, lui
shift	sll, sla, srl
compare	slt, slti, sltu, sltui
control	beq, bne, j, jr, jal
data transfer	lw, sw

Turn in a description of the specific errors that you found, the test code that excited the errors, and a summary of your testing strategy.

You should keep these programs and use them to test your final project.

Generic Problem 4: Accounting

To give us feedback on the workload, approximately how many hours did you spend on the exercises? On the lab? How long before the deadline did you finish each assignment?