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

Fall 1997 D.A. Patterson
Handout #9

Assignment 5
CS152 Computer Architecture and Engineering

Due Dates:

- **10/17:** Team evaluation due for single cycle processor lab
- **10/17:** Proposed division of work for this lab due
- **10/23:** Design review during discussion section
- **10/24:** Exercises due at noon at 283 Soda
- **10/28:** Lab due at 283 Soda
- **10/31:** Team evaluation due for pipeline processor lab

PLEASE put the DISCUSSION SECTION TIME (or number) that you attend as well as your name, as graded homework will be handed out in discussion. No number, no homework.

Every student should turn their own solution to the following problems by 10/24/95 from P&H: 5.6 (see following note), 5.16, 5.24, 6.5, 6.7. These should be done individually to ensure that everybody gets a chance to do the reading. They will help you in the project portion of the assignment as well.

Add the following to **5.6**: There are multiple solutions; choose the solution that minimizes the number of clock cycles for this added instruction. Show the steps in executing the jal instruction in the multi-clock datapath, using the same breakdown of steps as used in pages 382 to 389. Show the additions to the finite state machine of Figure 5.42 on page 396 to implement the jal instruction.

1 Goal

In this assignment, you will pipeline your processor. You will still use the same memory structure as in the last assignment and will get a chance to enhance that feature of your processor in the next lab. **This assignment is non-trivial** so don't wait to get started.

Instructions on what you will need to turn in for this lab will be posted to the newsgroup early next week.

This lab assignment is to be completed with your project partners. To help you build a successful machine, we will give you intermediate milestones for this assignment. As soon as possible you and your partners should form a strategy for dividing up the work and completing this lab. E-mail this strategy to your teaching assistant(s) no later than Friday, October 17. During discussion section on either Thursday, October 23 your group will have a design review with your TA(s). Please email your TA(s) a coherent version of each member's on-line logbook the day before your meeting so that they can review your progress off-line. Finally, on October 28th, you will turn-in your completed project in Soda 243 (noon).

NOTE: *Each* member of the group is required to keep and maintain his or her *own* online logbook for this and future lab assignments!

2 Team Evaluation

To help us understand how your team is functioning we are instituting a bi-weekly survey.

Based on your own observations, evaluate the performance of the other members of your team during the last 2 week period. You do not evaluate yourself. Assume an average team member would receive a score of 20 points. Top performers would receive more, sliders less. There are two constraints your point distribution must satisfy:

1. The total points distributed to your team members equals 20 times the team size minus one (i.e., yourself).
2. The maximum score you can assign to any one person is 30 points.

For example, suppose you were on a 5 person team with the following other members: Sue Superstar, Ole Outstanding, Annie Average, and Ned Neverthere. You would have 80 points (4 times 20) to distribute. Your distribution might look like the following:

Name	Score
Sue Superstar	23
Ole Outstanding	23
Annie Average	20
Ned Neverthere	14
total	80

You should reevaluate your team members every 2 weeks and base your evaluation only on performance over the last 2 weeks. These scores will be used for grading. Be honest and fair as you would hope others will be.

Turning this in: you must report this information to your GSI(s) by email on Friday, October 17, and Friday, October 31. We will not accept evaluations received after 8am on the following Friday. Your email message should have the following form:

<team name>	
<discussion section>	
<your name>	
Time in meetings:	<real num>
Time working:	<real num>
Time reviewing:	<real num>
<member name>	<score>
<member name>	<score>
...	

3 Pipelining your design

You have to implement the following five-stage pipeline:

IF Instruction Fetch

Access the instruction cache for the instruction to be executed.

ID Instruction Decode

Decode the instruction and read the operands from the register file. For branch instructions, calculate the branch target instruction address and compare the registers.

EX Execute

Bypass operands from other pipeline stages. One of the following activities can occur:

- For computational instructions, the ALU or the shifter performs the arithmetic or logical instructions.
- For load or store instructions, the data address is calculated.

MEM Data Memory Access

Access the data cache for load or store instructions.

WB Write Back

Write result to register file.

In a way similar to what is done in book, add the appropriate pipeline registers to your single cycle design. Assume that memory access takes only one cycle in your implementation so far.

Here are the instructions you must implement:

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

You should implement the components you need in VHDL. All VHDL models should correspond to realistic components (e.g. register, comparator, etc). You should include reasonable delays in your VHDL models; use these numbers:

Name	Delay (ns)
nand	0.5 (for reference)
bts4x32	2.0
sign_ext	1.6
m16x21	6.6
m32x21	7.0
m32x31	8.0
m32x51	10.0
m5x21	6.0
reg32	3.0
regfile	20.0
shifter	6.0
ALU	20.0

If you chose to implement pieces in logic you may get faster times.

4 Handling Hazards

Here is a list of the hazards you must handle:

- Data Hazards

  Data hazards occur when the data produced by an instruction is used as an operand by subsequent instructions.

  The ALU is one source of data hazards. Add the necessary forwarding logic and buses so that the result of an ALU operation can be immediately used by the following three instructions without waiting for the data to be written in the register file.

  Load instructions also present a data hazard because the data is not available until the end of the MEM phase. The MIPS instruction set specifies that load instructions have a one cycle delay. That means that the compiler cannot generate code sequence where the data of a load will be used during the following cycle. Implement your datapath so that it has one load delay slot.

- Control Hazards

Branch instructions present a common case of control hazards. Comply with the MIPS instruction set definition and implement your processor so that it has one branch delay slot that is always executed regardless of the result of the branch.

5 Extra Credit: Interlocking Loads

Your pipelined processor design could stall the pipeline if the instruction immediately after a load tries to use the first instruction's destination register. This feature was added to later versions of the MIPS instructions set.

6 Extra Credit: Handling Exceptions

Your pipelined processor design could handle external interrupts. To implement interrupts in your processor you will need to add a maskable CAUSE and an EPC (Exception Program Counter) register. Add a 3 bit bus into the processor to determine the cause of the interrupt. Call these Int0, Int1 and Int2.

Exceptions can be of two types, *internal and external.* For now the only internal interrupts are illegal instructions and arithmetic overflow. For the external interrupts you watch the 3 bits coming in and determine the type of interrupt.

To support interrupts add the following instructions to your processor:

Op	Description
mtc0	move to system control coprocessor
mfc0	move from system control coprocessor
rfe	restore from exception

In order to have interrupts occur easily, also add the following operations:

Op	Description
add	signed add; overflow exception possible
sub	signed subtract; overflow ("underflow") exception possible
addi	signed add immediate; overflow exception possible

(Note that the destination register is not updated when an overflow exception occurs.)

In addition to the above new instructions, you must write and test appropriate software routines for handling interrupts.