Final Exam

CSEE W3827 - Fundamentals of Computer Systems — April 22, 2021
Spring 2021 — Prof. Rubenstein

This final contains 3 questions (not counting question 0), totaling 90 points. Question 0 gives an additional 10 points. To get full credit you must answer all questions. **BOOKS AND NOTES ARE PERMITTED, ELECTRONIC DEVICES CAN BE USED FOR NON-COMPUTATIONAL PURPOSES AND NON-SEARCH PURPOSES.** The time allowed is 180 minutes, plus an additional 15 minute grace period to deal with upload/download issues.

Please format your exam as follows:

- Start each question on a separate page
- The file you upload should start with your UNI (preferably all lower-case).

Some advice:

- Be sure to leave some time to work on each problem. The right answer to each problem does not require a very long answer.
- Be sure to start every problem. And take some time to think about how to set the problem up before you start writing.

0. **10 points** The filename of your upload must start with your UNI (all lower case letters preferred) to receive these 10 points.

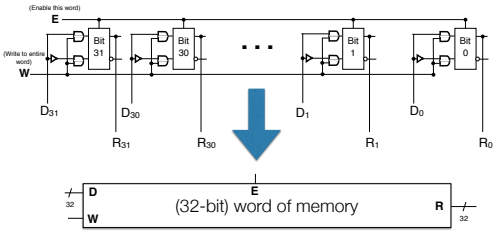
1. Consider the following 3-instruction code snippet that checks for equality of values in two locations of memory:

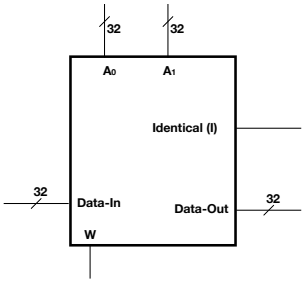
```
lw $s0, 0($s2)
lw $s1, 0($s3)
beq $s0, $s1 THEY_ARE_THE_SAME
```

The code uses three clock cycles, without even accounting for potential pipeline stalls. Suppose the comparison could be performed within memory itself using a single instruction we'll call **bme** (for branch if memory equal) of the form:

```
bme $s2, $s3, THEY_ARE_THE_SAME
```

where, as in the original snippet, $s2 and $s3 store the addresses where the comparison is being performed.





The figure on the left shows how a standard memory word slice is constructed. Using these standard memory word slices, design this new memory, depicted in the figure on the right, with the following properties:

- In addition to a first 32-bit address input (which we will call A_0), it has a second 32-bit address input, A_1.
- It has an additional 1-bit output I that equals 1 when the **contents** at addresses A_0 and A_1 are identical, and equals 0 otherwise.

In addition to the word slices, you may use any additional circuitry you think you need to complete the design (e.g., MUXes, Enablers, Decoders, AND gates, OR gates, NOT gates, XOR gates, etc.)

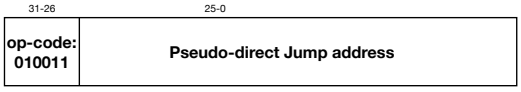
You need to only draw circuitry that shows how this extension is implemented. Do not worry about replicating the circuitry needed for normal operation (i.e., you don't have to show the circuitry that performs "normal" reads and writes from memory).

2. (30 points) A procedure often places its own return address value (in the $ra register) on the stack when making its own call to another procedure. To return, the procedure must pull the return value off the stack and then return. Sample MIPS code is shown below:

```
## code for jump-to-procedure          ## code for return
addi $sp, $sp, 4                       lw $ra, 0($sp)
sw $ra, 0($sp)                         addi $sp, $sp, -4
jal LABEL                              jr $ra
```

Suppose additional instructions jalm and jrm are added to the architecture that write and read directly from the stack instead of using the $ra register: jalm pushes the return address onto the stack, and jrm pulls the return address from the stack, $ra is not used.

The rest of this probem involves extending the traditional single cycle architecture depicted on the next page.

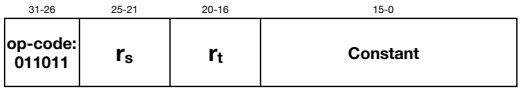


(a) (15 pts) Given the format of the jalm instruction is pictured above (containing only op-code and 26 bits of an address), enhance the architecture to implement jalm. In particular:

i. (5 pts) How should the various fields exiting Control be set?

ii. (10 pts) Add any circuitry you need to properly implement the instruction. You only need to show circuitry to make jalm work: In particular, assume Control outpus an additional field IsJALM that equals 1 only when the current instruction is a jalm. You do not need to include MUX's that would select on this signal between the existing architecture circuitry and your new circuitry: we (the graders) would know where those are needed.

Hint: It might be useful to know that the stack pointer ($SP) is register 29 in the register file.



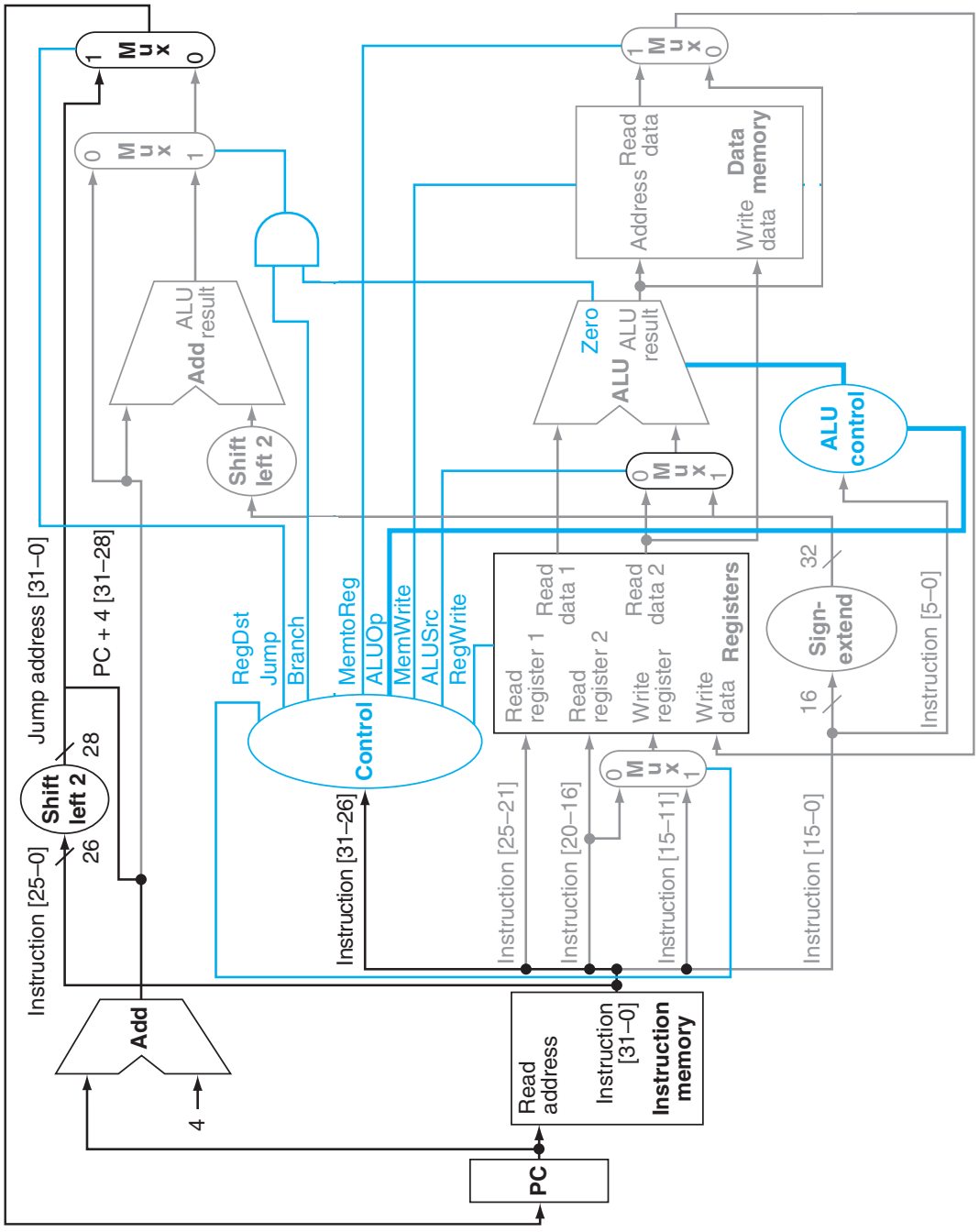
(b) (15 pts) The format for the jrm instruction is depicted above. Note that jalm instruction itself has no parameters, so you can fill in these fields in a way that might help solve the problem. Enhance the architecture to implement jrm. In particular:

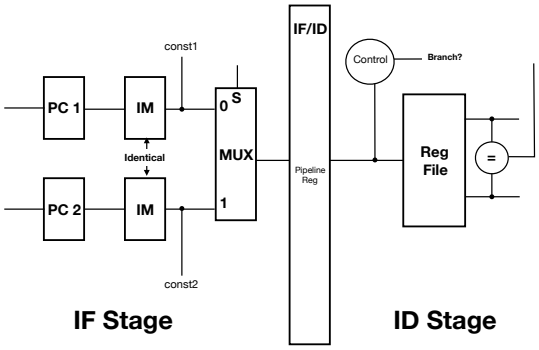
i. (5 pts) How should the various fields exiting Control be set?

ii. (10 pts) Add any circuitry you need to properly implement the instruction. You only need to show circuitry to make jrm work: In particular, assume Control outpus an additional field IsJRM that equals 1 only when the current instruction is a jalm. You do not need to include MUX's that would select on this signal between the existing architecture circuitry and your new circuitry: we (the graders) would know where those are needed.

Hint: It's probably still useful to know that the stack pointer ($SP) is register 29 in the register file.

(This figure is the single cycle architecture for problem 2.)





3. (30 points) Consider the enhancement to a pipelined architecture depicted above that prevents stalls after a branch instruction. In the IF stage, there are two program counters, each pointing to an identical instruction memory, each fetching an instruction. This enables the fetching of instructions at two different locations in memory. Constants that might be embedded within these instructions can be pulled out at this time as well, indicated as **const1** and **const2**. A MUX selects one of these instructions to be saved within the IF/ID pipeline register for the next stage

   The ID stage is the same as the final pipelined architecture covered in class: the instruction is parsed by Control, which, among other fields, indicates whether the instruction is a branch instruction (via the Branch? output). This instruction also pulls values from the register file and compares their equality (i.e., in case the instruction is beq or bne).

   The idea here is to simultaneously perform branch prediction where one PC (e.g., PC1) assumes the conditional will be false, and where the other (e.g., PC2) assumes the conditional will be true.

   Note for full credit, your solution should work even for code such as what follows:

```
   beq $s0, $s1, LABEL1
   beq $s2, $s3, LABEL2
LABEL1:
   beq $s4, $s5, LABEL3 #somewhere else not shown
LABEL2:
   beq $s6, $s7, LABEL4 #somewhere else not shown
```

   (a) (15 points) Write equations that indicate the next values of the PCs. For instance, in the regular 1-PC architecture, the equations would look like:

   - PC = PC + 4 + 4*const when the instruction in the ID stage is a branch and the conditional is true.
   - Otherwise, PC = PC + 4

   (b) (15 points) Complete the diagram above by drawing the circuitry that feeds into each PC (setting its value for the next clock cycle) as well as the circuitry that feeds into the selector of the MUX that decides which instruction remains in the pipeline.

You have reached the end of the exam. Have a great Summer!