Final Exam Solutions

CSEE W3827 - Fundamentals of Computer Systems S2: 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.

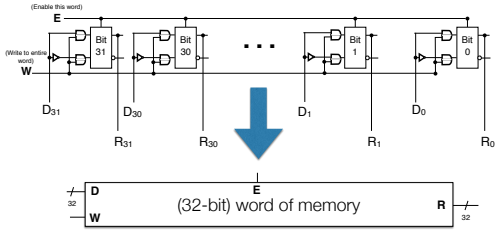
0. **10 points** The filename of your upload must start with your UNI (all lower case letters preferred) to receive these 10 points.

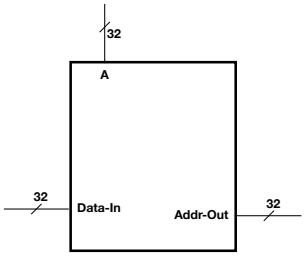
1. Given $s0 holds a 32-bit data value and $s1 holds an address, the MIPS code that sets $s1 to the address in memory (after where $s1 initially points) that holds a value matching the value in $s0 is as follows:

```
START:
    lw $s2, 0($s1)
    beq $s2, $s0, DONE
    addi $s1, $s1, 4
    j START
DONE:
```

Each loop in the above takes 4 clock cycles, so if the first match occurs 4*N addresses later, finding that address would take 4*N clock cycles.

We could, instead, build this functionality into the memory itself.

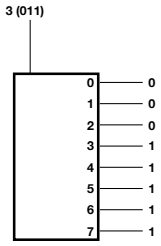


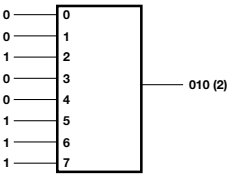


Show how to implement a memory whose circuitry can provide this result. (Do not worry about having your memory implement normal "reads" and writes"). The memory is depicted above right takes in a starting address A, a value to search for (Data-In), and returns the first address after A that matches this value, returning 0 if no such address is found.

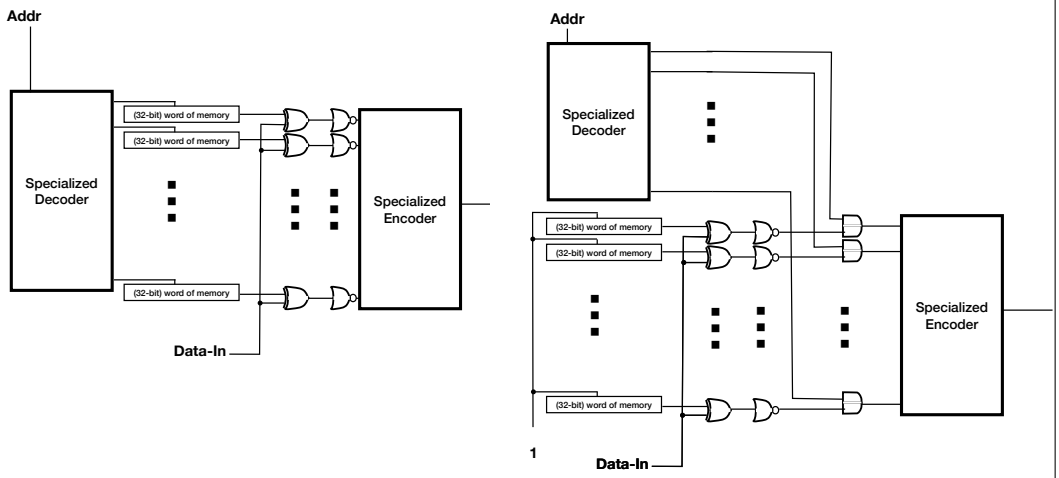
To build your memory you may use:

- Word slices of memory whose design is depicted above left.
- A specialized decoder that takes as input a k-bit string S and outputs a 0 on all outputs whose number is less than S (interpreted as an unsigned binary), and outputs 1 on all other outputs. An example with $S = 3$ is depicted below left.
- A specialized encoder circuit that takes 2^k inputs (numbered from 0 to $2^k - 1$) and returns the number (as an unsigned binary) of the lowest-numbered input that equals 1, returning 0 if no input equals 1 (and of course if the 0th input equals 1). An example is depicted below right where input 2 is the first non-0.





Answer: An acceptable answer is one that works for all cases where the value in Data-In is non-zero. For this solution, we use the specialized decoder to enable the memory cells that occur at or after the specified address. All memory cells are XOR'd with the Data-In input, whose output will equal 0 wherever the memory cell matches the Data-In value. These bits can then all be NOR'd together, such that the result equals 1 only for a match. These results are then fed into the specialized encoder, which identifies the address of the first match. This design is pictured below left.



It is possible to also identify the address, including when Data-In equals 0. For this design, Enable all memory cells and perform XOR with Data-In, and NOR the resulting bits togther, such that the result equals 1 only memory words that matched the Data-In. AND these with the outputs of the specialized decoder, and feed these results directly into the specialized encoder. This solution is pictured above right.

2. (30 points) Recall that the following code snippets respectively "push" and "pop" items on/off the stack:

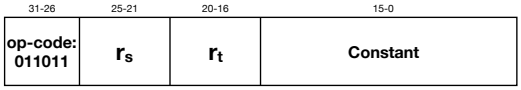
```
## code for push                    ## code for pop
addi $sp, $sp, -4                  lw $s0, 0($sp)
sw $s0, 0($sp)                     addi $sp, $sp, 4
```

Suppose the following instructions are added to the instruction set architecture that have the same behavior as above:

```
## copy $s0 onto top of stack       ## remove stack top, place in $s0
push $s0                           pop $s0
```

(a) (10 pts) The architecture depicted on the next page is enhanced from the normal architecture only in that the MUX that feeds into the Write Register of the Register File can select from three possible locations in the instruction word (instead of the normal two).

Assuming the stack pointer register $sp remains in the register file as register number 29 (11100 in 5-bit binary), using the existing architecture, explain how to implement **push $rx** within this architecture when the 32-bit instruction has the format:



Explain:

i. (5 pts) How the fields in the instruction should be filled in? (I.e., what should r_s, r_t and constant be set to, given that the only information in the instruction itself is the register $rx whose data should be pushed onto the stack?)

Answer: We can set r_s to 29, such that $sp is read from and written to during the clock cycle (so that -4 can be subtracted from it). r_t will be the register provided as a parameter to push (which will get written to memory). We set the constant to -4 so that the ALU can perform the addition (or set to 4 and request that ALU does the subtraction).

ii. (5 pts) What should the value of each of the fields exiting the Control circuit be? For ALUOp, you can just indicate the operation (i.e., '+', '-', AND, OR, shift-left, etc.) as opposed to giving the 2-bit Op code.

Answer:

- RegDst: set to 10 (2) such that bits 25-21 (r_s) are written to (these were set above to 29).
- Branch: 0
- MemRead: (we ignore this one)
- MemToReg: 0 (such that ALU computation of $SP - 4 is sent for writing the register file).
- ALUOp: since our constant was -4, we add. If the constant were 4, we would subtract.
- MemWrite: 1 (push writes to memory)
- ALUSrc: 1 (we want the constant embedded in the instruction)
- RegWrite: 1 (we want to write to $sp)

(b) (5 pts) To implement pop, we need to implement the $sp register outside of the register file (similar to how the program counter lies outside the register file). Explain why (one sentence should suffice).

Answer: Pop needs to write the to register file in order to store the value pulled from memory in the register specified as a parameter in the pop instruction. Since the register file can only write to one register at a time, the modification to $sp cannot be performed in the same clock cycle, and hence must be external to the register file.

(c) (15 pts) With $sp implemented outside of the register file, show how to implement the **pop** instruction. Assume Control has one additional 1-bit output, Pop, that equals 1 only when the instruction is a pop. (Do not worry about implementing push for this case).

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

Answer:

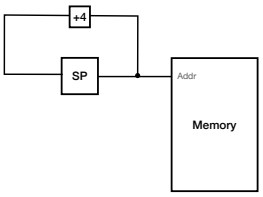
- RegDst: set to 00 such that bits 20-15 (r_t) are written to (assuming we still have the register provided in the pop instruction mapped to these bits). If they are mapped to bits 25-21, we could keep RegDst set to 10 (2).
- Branch: 0
- MemRead: (we ignore this one)
- MemToReg: 1 (such that top-of-stack value is written back to the register file).
- ALUOp: since our constant was -4, we add. If the constant were 4, we would subtract.
- MemWrite: 0 (pop does not need to write to memory)
- ALUSrc: Depends on implementation, but the way we will implement, does not matter (we will feed $sp directly into the address field of memory).
- RegWrite: 1 (we want to write to the register located in field r_t of the instruction.

ii. (5 pts) How should the fields of the instruction be filled in (you can assume a different op-code in this case of 010011).

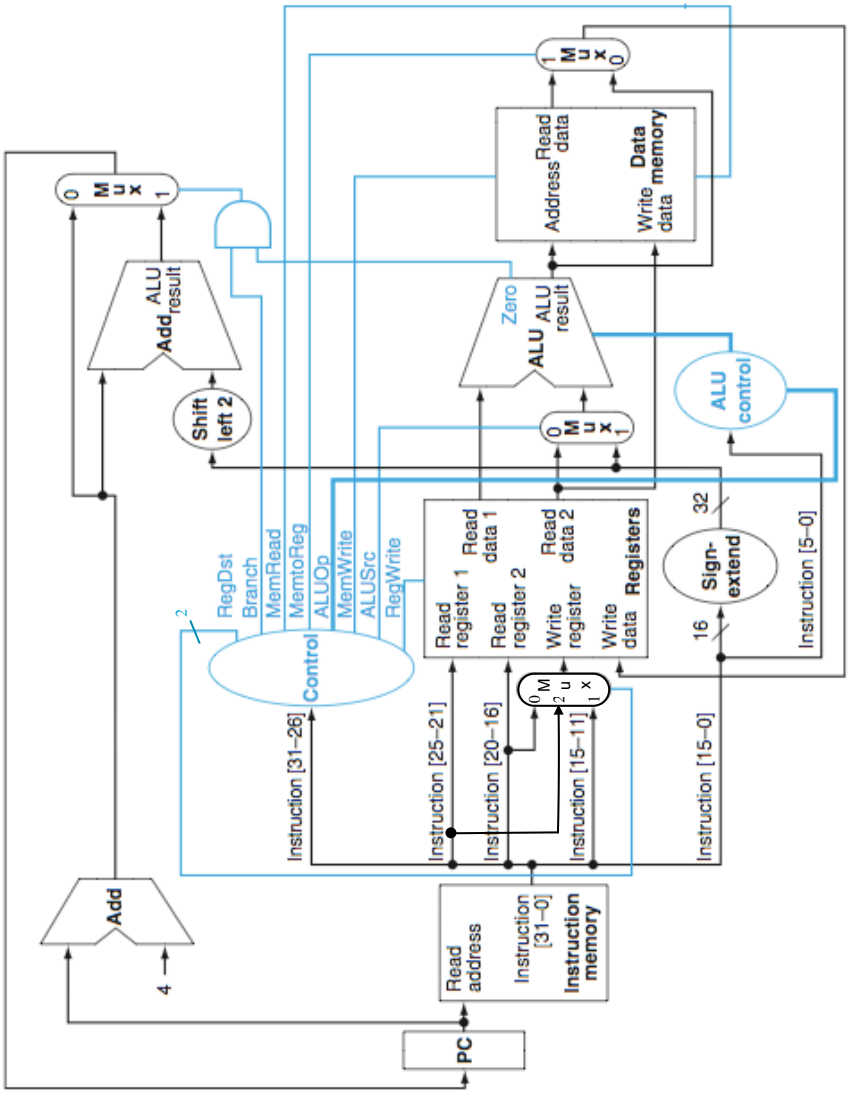
Answer: To match with what we described above, we put the register specified as the parameter of pop ($rx) into r_t so that it is the register to be written to. Since $sp is no longer in the register file, it doesn't have a register number so we don't use the r_s field. Finally, we need to feed $sp into the memory address directly, prior to adding 4, so there is really no advantage to adding 4 via the ALU, so the constant field is not used.

iii. (5 pts) How $sp, now outside the register file, interfaces with the single cycle circuitry. You need only draw the regions/enhancements to the circuitry that involve the stack pointer register.

Answer: The stack pointer must be fed directly into the address input of memory, such that the value at this location is read out to be written to the destination register. The value in the stack pointer must be increased by 4 and re-stored into the stack pointer (effectively removing the item from the stack).



(This figure is the enhanced arch for problem 2a.)



3. (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, as follows:

```
lw $ra, 0($sp)
addi $sp, $sp, -4
jr $ra
```

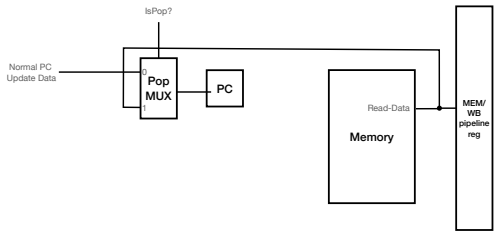
(a) (10 points) Just like jump, the instruction that initially follows jr into the pipeline will be invalid and force a stall. However, in the existing architecture, there is an additional stall that will happen prior to the aforementioned stall. In one sentence, explain what causes it.

Answer: The jr instruction must receive the value retrieved by lw. lw will be near the end of the MEM stage when it retrieves that value, which would happen when jr is near the end of the ID stage. The jr instruction needs to be stalled so that lw can enter the WB stage and write to $ra in the register file and then jr can read it.

(b) (20 points) Show what data forwarding should be added to avoid this stall. Depict only the additional circuitry needed. You can assume that the hazard detector and forwarding circuitry are adjusted to accommodate your changes.

(Hint: recall that only the registers in the register file perform write before read within a clock cycle.)

Answer: Forward the value that comes out from memory (that will get written to $ra in the next clock cycle) directly to the PC, as shown below: note the MUX and the IsPop? flag is added only for clarity - not needed for credit.



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