Questions

1. In the circuit of the ALU back in lecture 4, we included an adder, an AND gate, and an OR gate. A multiplexor was used to select one of these three values. This circuit was repeated for \( n \) bits.

Consider the MIPS instruction `slt` (set if less than). This instruction takes two register arguments and writes a value `0x00000001` into the destination register if the first argument is less than the second, and it writes value `0x00000000` to the destination register otherwise.

What would you need to add to the above circuit so it also could perform the `slt` instruction? Note that the answer depends on whether we are considering bit 0 or bits 1-31 since the output value is always 0 in the latter case when the instruction is `slt`.

Hint: the `slt` case condition to be added to the MUX.

2. Give four different ways in which an address is computed or obtained as part of a MIPS instruction.

3. [This question was from the final exam in 2012.]

For each of the three MIPS instructions below:

- Specify as much of the machine code of the instruction as you can, showing how many bits are in each field and leaving unknown fields blank.
- Draw the datapath and controls for a single cycle implementation of the instruction; only include parts of the datapath that are used in the instruction; specify the bit width of any lines you draw in the datapath, and specify any known values.

(a) `addi $15, $0, -8` # add immediate
(b) `sltu $16, $15, $0` # set less than (unsigned)
(c) `jalr $16, $15` # special version of jump and link whereby you
    # jump to the address specified in a register and
    # also store return value
4. Identify the hazards in the following if-then-else sequence. Describe how to resolve these hazards using as few \texttt{nop} instructions as possible.

```assembly
if:       slt $t0, $s0, $s1
          bne $t0, $0, else
          lw $s3, 0($sp)
          add $s3, $s3, $s4
          j   endif
else:    sub $s3, $s3, $s4
endif:  and $t0, $0, $0
         or $t1, $0, $0
```
Solutions

1. [Modified June 2016]

The ALU consists of 32 circuits such as shown below. For the \texttt{slt} instruction and for bit 0, we want the circuit to output a 1 if the ”less than” condition is true and 0 otherwise. The remaining bits should output 0 for the \texttt{slt} instruction. How can both of these be achieved?

The \texttt{slt} instruction compares two argument registers. For simplicity let’s call these registers \(s_1\) and \(s_2\). The circuit should subtract \(s_2\) from \(s_1\), and then check if the result is negative. To compute the subtraction, the \texttt{slt} instruction must set the \texttt{Binvert} control to 1, just like the \texttt{sub} instruction does. To determine if \(s_1 - s_2 < 0\), check the adder output for bit 31. So, bit 31 of the adder output is used as ”\texttt{sltValue}”. See green label below.

For bit 0, the fourth input is ”\texttt{sltValue}”, i.e. it takes value 1 if \(s_1 < s_2\), and it takes value 0 otherwise. For bits 1-31, the fourth input to the MUX is always a 0 value.
2. (a) read from a register
(b) sum a base and an offset, as in a load or store word instruction
(c) sum PC+4 and an offset, as in a conditional branch
(d) concatenate the upper four bits of PC with a positive offset (this is similar to 3.) as in the j instruction.

3. (a)

```
addi   $15, 0, 8
```

Here are a few details that should be explicit in your solution.
- I-format partition with the appropriately labelled fields of each.
- The values 15 and 0 are coded in 5 bits each in the two register slots.
- fetch stage: PC register to Memory (instructions) and the 32 bit instruction is read out and decomposed into the I-format fields.
- $PC \leftarrow PC + 4$
- lines into and out of registers are labelled with bit numbers
- sign extend the immediate argument from 16 to 32 bits
- the value read from the register is fed to the ALU, along with the sign extended value; result is written into a register
The datapath here is similar to (a) so I’ll only mention the key difference: \texttt{sltu} is R format so the partition of the instruction into fields is different. You need to know how many bits are in each field. The rs, rt, rd fields are 15 (01111), 0 (00000), 16 (10000). Note that the \texttt{slt} details from Question 1 are not explicit here since those details are within the ALU.
(c) The datapath for the jalr instruction is a combination of jr and jal datapaths which I sketched out in class.

\[ \text{jalr} \quad \text{pc} \quad \text{rs} \quad \text{rd} \quad \text{func} \quad \text{R format} \]

- You might use an I format or an R format for this. (The correct answer is R format but there is no way to know that.)
  If you assumed I format, then the rs register (source) register 15 (01111) is read out and fed into the PC, and the destination register would be rt and would have the value 16 (10000). This is where PC + 4 is written.
  If you assumed R format, then the rs register (source) register 15 (01111) is read out and fed into the PC, and the destination register would be rd and would have the value 16 (10000). This is where PC + 4 is written.
- PC is updated with the 32 bit value from the source register. To keep the figure simple, I have not bothered with the MUX for selecting the PC write.
- The PC+4 value is written into a register so there should be a line from the PC to the register. (I have not bothered with a MUX there, but in general there would be one.)
4. There are four (pipeline) hazards to be identified here.

(a) (data hazard) The result of $\text{slt}$ is not known until the WB stage, but it is needed by the $\text{bne}$ instruction. The value is known after the ALU stage of $\text{slt}$ and must forwarded to the ALU stage of the $\text{bne}$ instruction.

(b) (control hazard) The $\text{lw}$ and $\text{add}$ instructions will be fetched while the $\text{bne}$ is decoded and goes through the ALU, respectively. The $\text{lw}$ and $\text{add}$ instructions should only be executed if $\text{bne}$ condition is not met. If the branch condition is met, then the WriteReg control of the $\text{lw}$ and $\text{add}$ instructions should be set to 0. They will still go through the pipeline but they won’t write back.

(c) (data hazard) The $\text{lw}$ instruction gets a word from Memory but this only happens at the end of the MEM stage. This value is needed at the ALU stage of the $\text{add}$ instruction. Thus, a $\text{nop}$ needs to be inserted between $\text{lw}$ and $\text{add}$.

You may have been tempted to move the $\text{or}$ instruction in there, instead of $\text{nop}$. But this is incorrect since $\text{add}$ is conditioned on the branch, but $\text{or}$ is not conditioned on the branch.

(d) (control hazard) The $\text{sub}$ enters the pipeline after $\text{j}$ but it shouldn’t be executed. So we can (re)set its RegWrite control to 0 so it doesn’t write in the WB stage.