You may work in pairs for this assignment. If you choose to work with a partner, make sure only one of you submits a solution, and that you paste a copy of the Partners Template that contains the names and PIDs of both students at the beginning of the file.

Prepare your answers to the following questions in a plain text file. Submit your file to the Curator system by the posted deadline for this assignment. No other file formats, or late submissions will be accepted.

You will submit your answers to the Curator System (www.cs.vt.edu/curator) under the heading `MIPS01`.

All questions refer to the completed single-cycle datapath (SCD) design, reproduced below, which supports execution of the following MIPS instructions: `add`, `sub`, `and`, `or`, `slt`, `lw`, `sw`, `beq` and `j`.

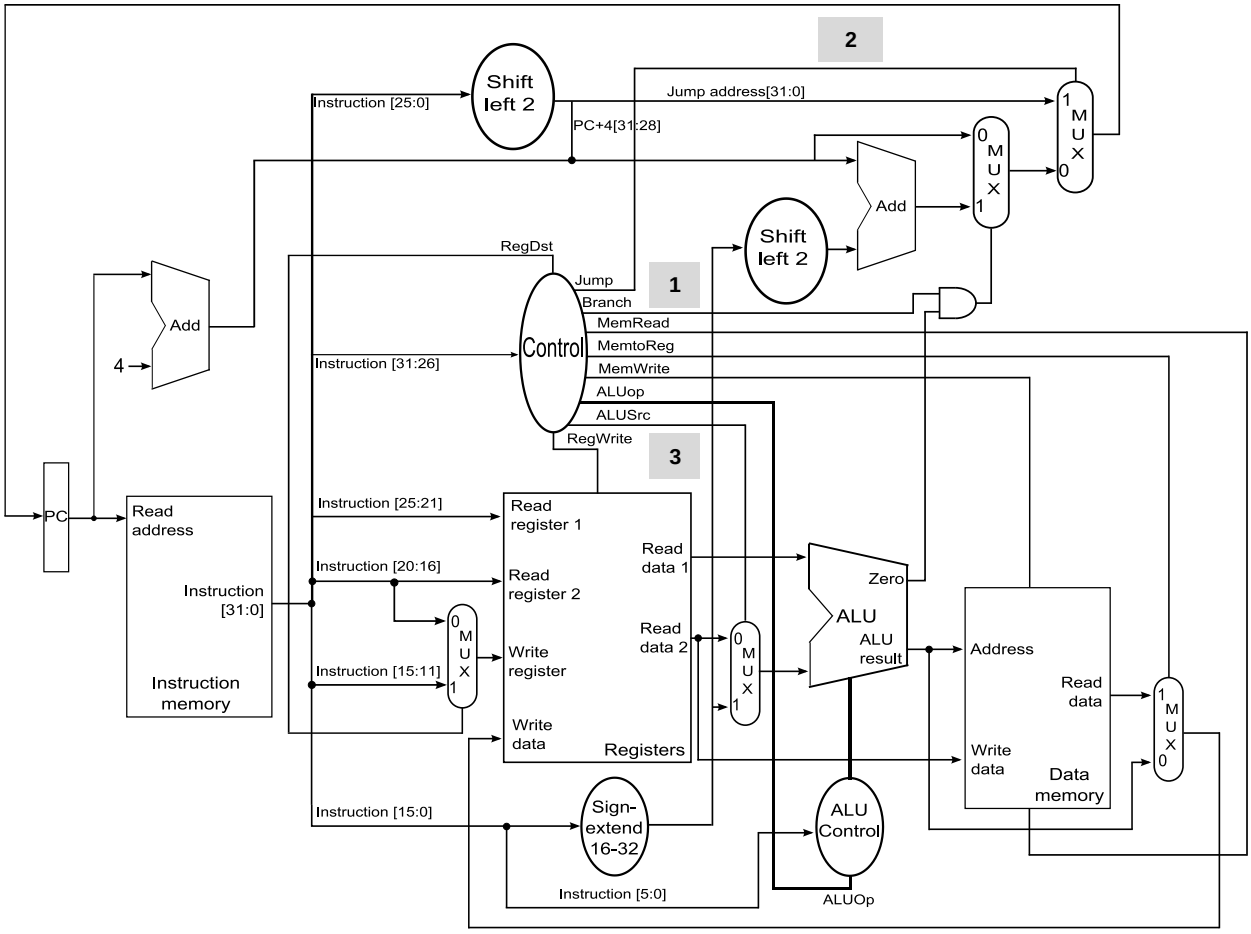


Figure 1

1. **[15 points]** Are the Branch and Zero signals both logically necessary? Or could one be omitted? Explain clearly.

2. **[15 points]** For which of the supported instructions does it not matter how the control signal Jump is set? Justify your answer precisely and completely.

3. **[20 points]** Suppose that, due to a manufacturing error, the control line ALUSrc is always set to 0. Describe the circumstances, if any, under which the following instruction would still produce the intended result:

```
lw  $t3, 8($s7)    # GPR[$t3] <-- Mem[GPR[$s7] + 8]
```

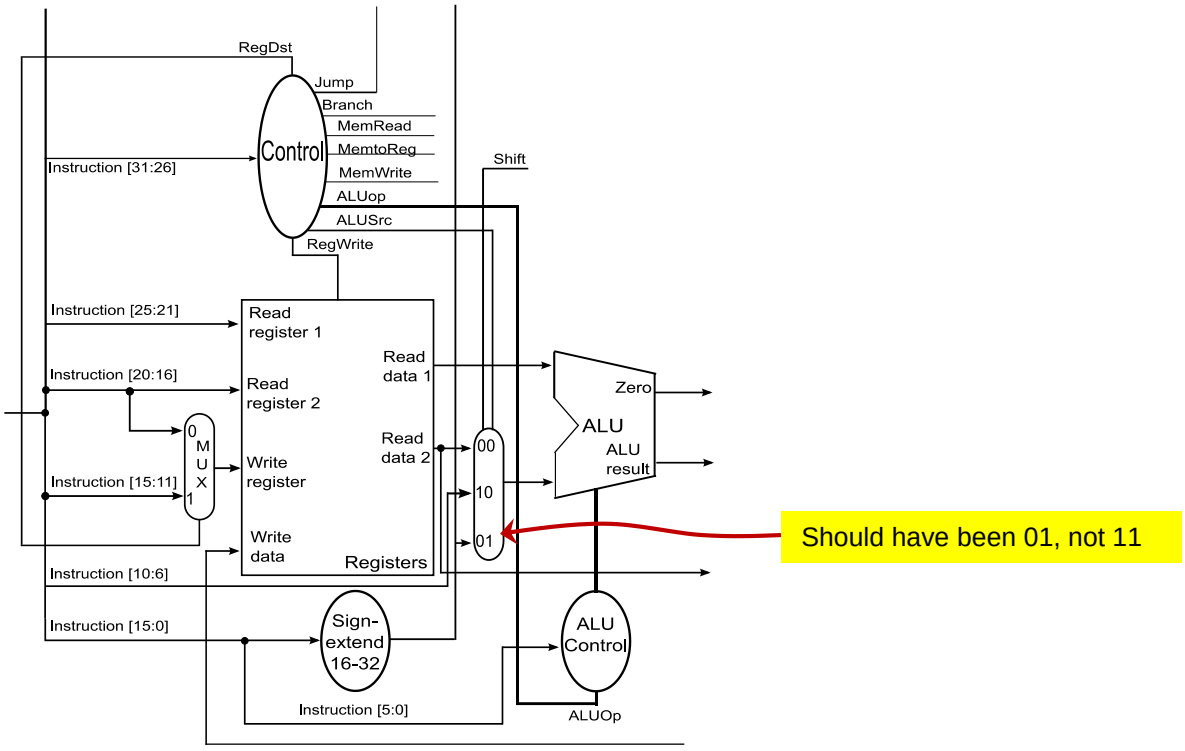
4. Consider enhancing the processor design in **Figure 1** to implement the `SLL` instruction:

```
sll rd, rs, sa      # GPR[rd] <-- (GPR[rs] << sa)
```

The SLL instruction is encoded as an R-format instruction:

000000	rs	rt	rd	sa	000000

The datapath will be modified as shown below (anything not shown here remains unchanged from **Figure 1**):



The ALU would, of course, also already include a shift unit, so what's necessary is to specify that the ALU should shift its left (upper) input, and provide the ALU with the number of places to shift.

The internals of the ALU control unit can be easily modified to take the `funct` field from an SLL instruction, and provide the necessary signal to tell the ALU to shift. We will not concern ourselves with that internal modification.

The MUX preceding the ALU is modified to take a third input, the 5-bit `shamt` field from the current instruction, which we will assume to have been properly shifted down and padded with zeros to 32-bit width. The three inputs will be numbered as shown in the diagram above.

The MUX now requires a 2-bit control signal. We can use the ALUSrc signal that was already present as the **low bit** for this MUX control. We will introduce a new control signal, Shift, which will be used as the **high bit** for this MUX control.

a) **[10 points]** Explain the circumstances under which Shift will be set to 0, and when Shift will be set to 1.

b) **[10 points]** Explain why the Control unit cannot be used to set Shift.

c) **[10 points]** What hardware unit can be used to set Shift? An existing one, or do we need a new one? If an existing unit will do, explain how it would be used. If no existing unit will do, explain why not.

5. This question relates to the tradeoffs that were inherent in the design of the MIPS32 machine language, due to the decision to strictly adopt a 32-bit format for all machine instructions.

The MIPS32 designers decided to use a 6-bit field for the opcode. That limited the number of different opcode values; as a direct result of this, the MIPS32 designers were forced to make some opcodes "special". For example, the opcode `000000` was used to specify R-format instructions, which then used a different set of bits (the `funct` field) to specify the actual instruction that was to be performed.

And, a direct result of that was that the MIPS32 datapath needed a second control unit to handle decoding R-format instructions.

Suppose the MIPS32 designers had decided to use 8 bits to specify the opcode for each instruction.

a) **[4 points]** Easy question… how many different machine instructions could have been specified using this wider opcode field? Explain why.

The MIPS32 designers would still have chosen to use a 16-bit immediate field since that corresponds to a common integer type, and still have chosen to represent every machine instruction with 32 bits.

b) **[8 points]** Describe how his would change the way I-format instructions would be formatted.

c) **[8 points]** Describe any resulting changes that would have to be made to the version of the MIPS32 datapath hardware shown in **Figure 1** to accommodate the new I-format design.