1 Single-cycle datapath control

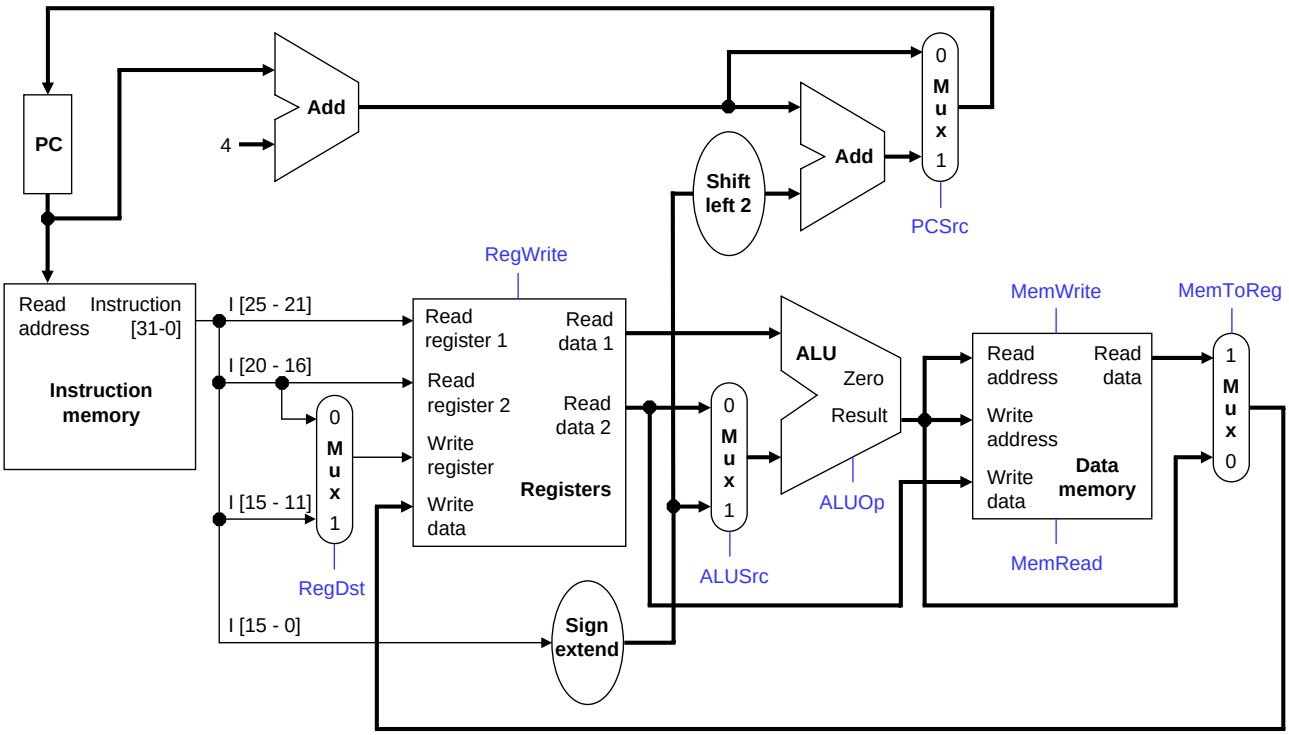
We'll first review single-cycle datapaths by computing the control signals for two instructions, `lw` (which was covered in lecture and should be review), and a hypothetical new instruction `lwd`:

```
lw $rt, offset($rs)    sets    R[$rt] = Mem[R[$rs] + offset].
lwd $rd, $rs, $rt      sets    R[$rd] = Mem[R[$rs] + R[$rt]].
```

Convince yourself that the datapath does not need to be changed and compute the control signals necessary for these two instructions. Recall that bits 31–26 of an R-type instruction form the *opcode*, bits 25–21 form the **rs** register, bits 20–16 form the **rt** register and bits 15–11 form the **rd** register.

inst	PCSrc	ALUSrc	ALUOp	MemWrite	MemRead	MemToReg	RegDst	RegWrite
`lw`								
`lwd`								

Refer to the datapath below.



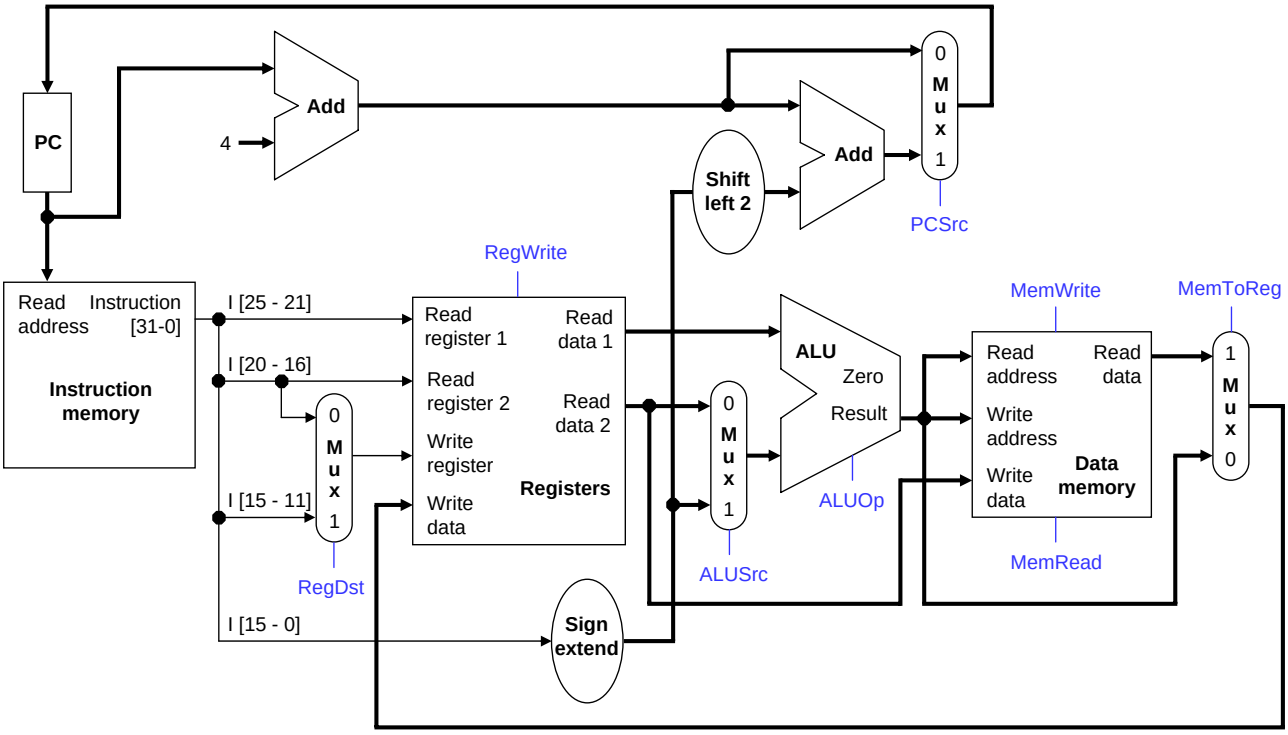
2 Single-cycle datapath modifications

Using the above datapath, we'll present an example of how to extend a datapath to support additional instructions. We'll demonstrate the extensions necessary to support the jump-and-link `jal` instruction. Recall that `jal` is a J-type instruction that performs two operations:

```
R[31] = PC + 4
PC = (PC & 0xf0000000) | (target << 2)  # the 26-bit target specifies bits [27:2] of new PC
                        # $ra is register 31
```

You are encouraged to use the datapath above to take notes on the `jal` modifications.

1. Extend the datapath shown below to support the `jr` instruction. This is an R-type instruction, and the new PC value should be the value in register **rs**.



2. Extend the datapath shown below to support load upper immediate `lui`. Recall that `lui` is an I-type instruction that performs the following operation:

   ```
   R[$rt] = imm << 16
   ```

   Assume that the ALU does not support shifts and insert a special-purpose shift unit (which is really just wires) for this operation.