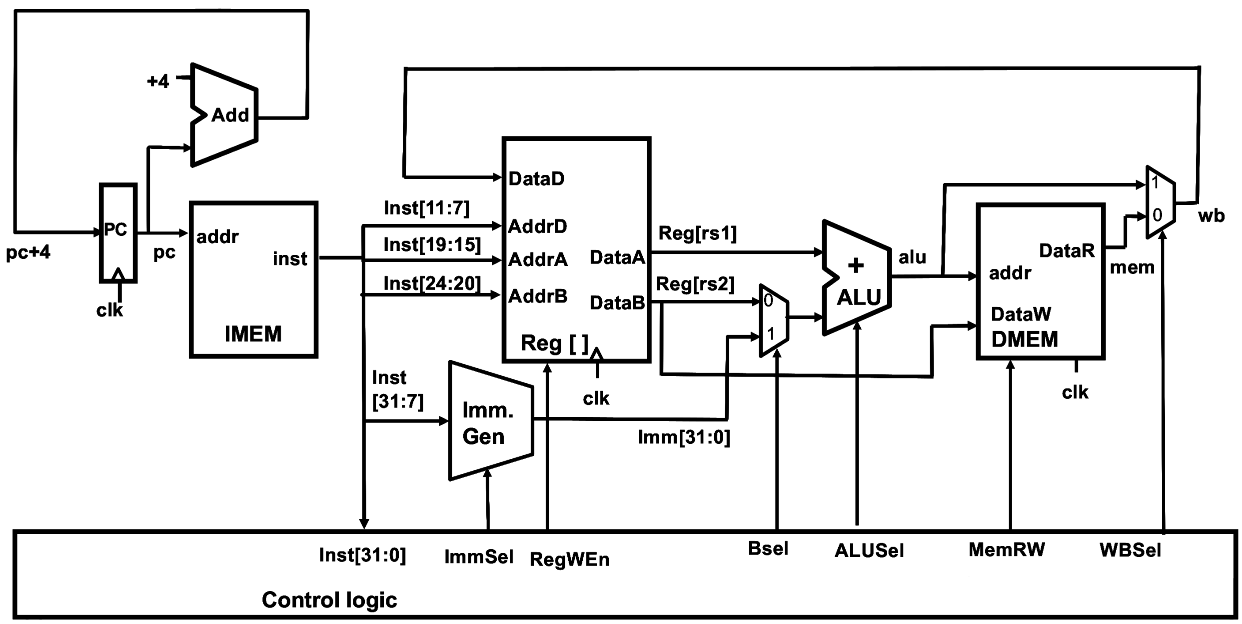
EECS 151/251A: Homework 4

Due Friday, February 25th

Problem 1: The principle, Datapath

The figure below shows a single-cycle datapath supporting a subset of the RV32I instruction set.



Part a) In order to support the following proposed instruction, new hardware are required to be added to the datapath. Draw the updated diagram to support the following instructions.

- `add rd, rs1, rs2, rs3` (operation: `rd = rs1 + rs2 + rs3`)
- `swadd rs1, rs2, imm` (operation: `M[rs1] = rs2 + imm`)
- `lwadd rd, rs2, rs1, imm` (operation: `rd = rs2 + M[rs1 + imm]`)

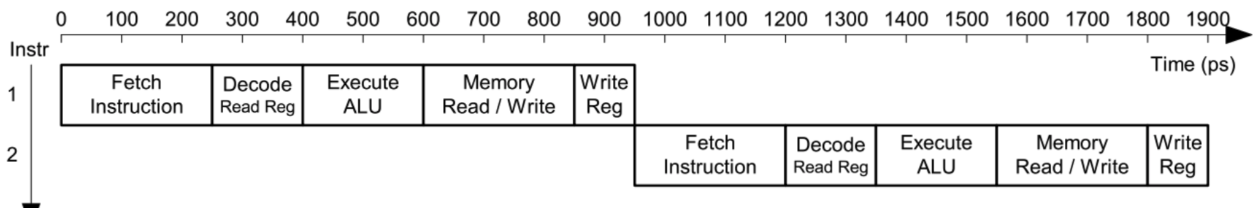
Part b) Suppose you want the design to also support B-type instructions. Modify the datapath above to support branches, including additional logic and control signals.

Part c) Consider in the provided datapath, supporting I-type and S-type instructions. In the table below, fill in the control signals for the given instructions. `ImmSel` takes the values `R`, `I`, and `S` for the instruction type, and `ALUSel` takes the ALU operation.

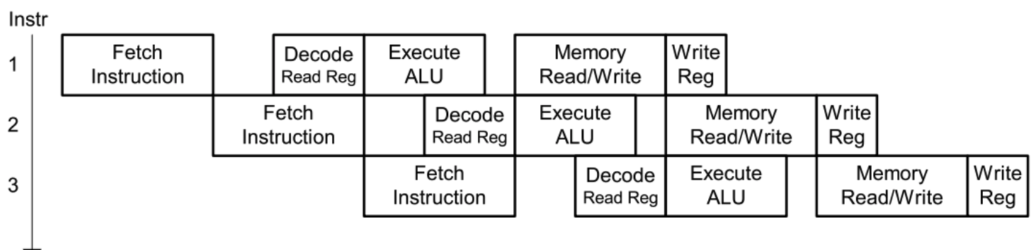
Instruction	**ImmSel**	**RegWEn**	**Bsel**	**ALUSel**	**MemRW**	**WBSel**
`add x3, x2, x1`						
`xori x5, x1, 5`						
`lw x2, 4(x2)`						
`sw x3, 8(x10)`						
`srli x1, x2, 3`						

Problem 2: Make it efficient, Pipelining

The diagram below shows 5 stage processor operating in single-cycle without pipelining, and multiple cycles with pipelining. Assume *Fetch Instruction* takes 250ps, *Decode/Read Reg* takes 150ps, *Execute/ALU* takes 200ps, *Memory Read/Write* takes 250ps and *WriteReg* takes 100ps. Plus, assume no performance overhead is introduced due to pipelining registers.



(a) Single-cycle



(b) Pipelined

Part a) What are the minimum required clock period for single-cycle and pipelined implementation?

Part b) We have a program of 2000 instructions running on each processor. What is the total time required if you run the program on the single-cycle processor? How about the pipelined processor?

Part c) A student redesigned the ALU, so now *Execute* only takes 120ps. Repeat Part a and b.

Problem 3: Ooops, watch out! Hazard

Let's say we have a simple 3 stage in-order pipelined processor with the following stages:

- Instruction Fetch: Read from IMEM, Decode instruction, Read Regfile
- Execute: Branch comparison, ALU operation
- Writeback: DMEM access, Writeback Regfile

Registers are read in the first stage and are written to in the third stage. Writes to registers occur at the end of a cycle while reads occur at the start.

Part a) Assume there is no data forwarding. How many cycles will the following assembly take to execute? Show how you derived the result.

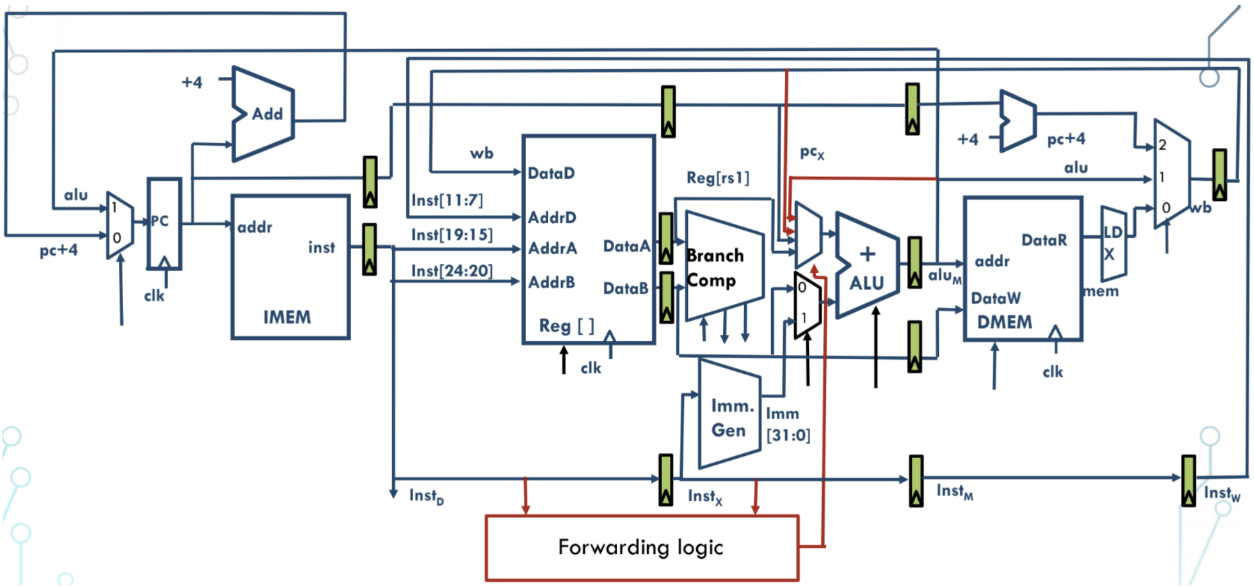
```
sub x0, x1, x2
add x2, x3, x4
slt x2, x3, x4
or x3, x2, x0
and x4, x1, x0
xor x2, x1, x4
sub x1, x2, x0
```

Part b) What is the CPI of this processor for this block of code?

Part c) Assume that we have ALU to ALU forwarding. Repeat part a and part b.

Problem 4: Have more fun with Verilog

You are given the simple assembly code below, and it runs with 5 stage pipeline.



Part a) Following shows Verilog code for an incomplete ALU.

```
wire signed [31:0] in1s, in2s;
assign in1s = in1;
assign in2s = in2;
always @(*) begin
    case (ALUSel)
        ADD: alu = in1 + in2;
        SUB: alu = in1 - in2;
        SHIFT_LEFT: alu = in1 << in2[4:0];
        LESS_THAN_S: alu = (in1s < in2s) ? 32'b1 : 32'b0;
        SHIFT_RIGHT: alu = in1 >> in2[4:0];
        OR: alu = in1 | in2;
        AND: alu = in1 & in2;
        PASS: alu = in2;
    endcase
end
```

Select all instructions below that are supported by the given datapath diagram and the given ALU module. Also, edit the Verilog code to support the selected instructions.

- `LUI rd, imm`
- `AUIPC rd, imm`
- `BLT rs1, rs2, imm`

- `SRA rd, rs1, rs2`
- `SRL rd, rs1, rs2`
- `SLT rd, rs1, rs2`
- `SLTU rd, rs1, rs2`
- `LW rd, rs1, imm`

Part b) To improve the performance, forwarding is implemented, such that the input to ALU can be from the Memory stage. In Verilog, design the control circuit coordinating the forwarding.

- RegWriteMEM: Control signal whether a register is written in the MEM stage
- WriteRegMEM: Destination register for the instruction in the MEM stage
- Rs1, Rs2: Value of *rs1, rs2* from the instructions in the EX stage
- FwdA, FwdB: Forwarding control signals for each operand

```verilog
module Forward(
    input       RegWriteMEM,
    input [4:0] WriteRegMEM,
    input [4:0] Rs1,
    input [4:0] Rs2,
    output      FwdA,
    output      FwdB
    );

    // ToDo


endmodule
```