Discussion 5

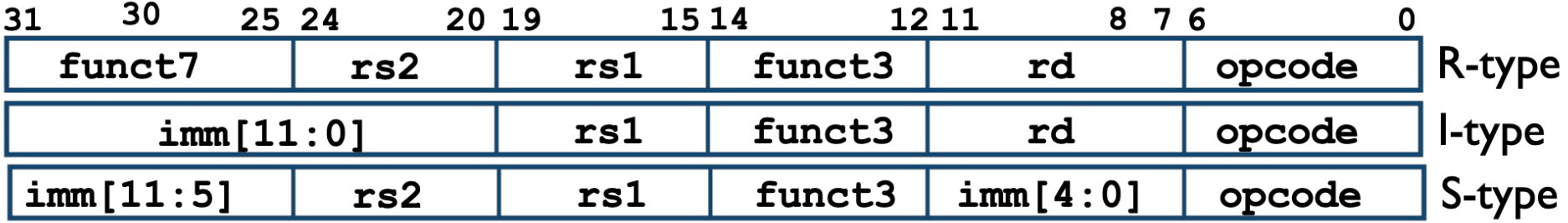
RISC-V ISA, Decode and Control Logic, Midterm Problem

RISC-V ISA

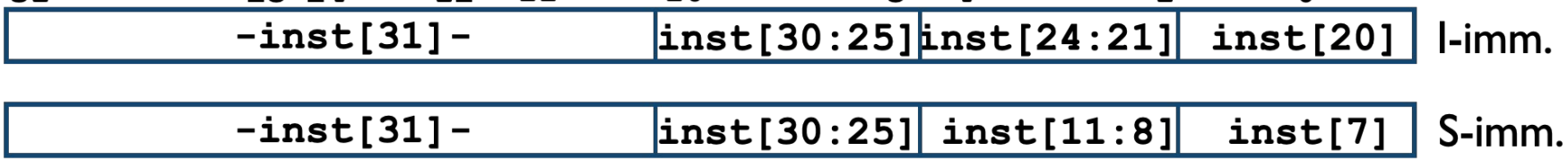
- RISC-V is a load-store RISC ISA with a small base instruction set
  - It’s the 5th major RISC ISA from UC Berkeley
- Fixed-length 32-bit instructions
- 32 registers in the base rv32i (integer) ISA, each with a width of 32 bits
  - x0-x31
  - x0 is always hard tied to 0
- Base instructions: register loading, arithmetic, logic, memory load/store, branches, and jumps
- Load-store ISA = ALU ops only allowed between registers (not memory locations directly)

RISC-V ISA

Instruction encodings, inst[31:0]



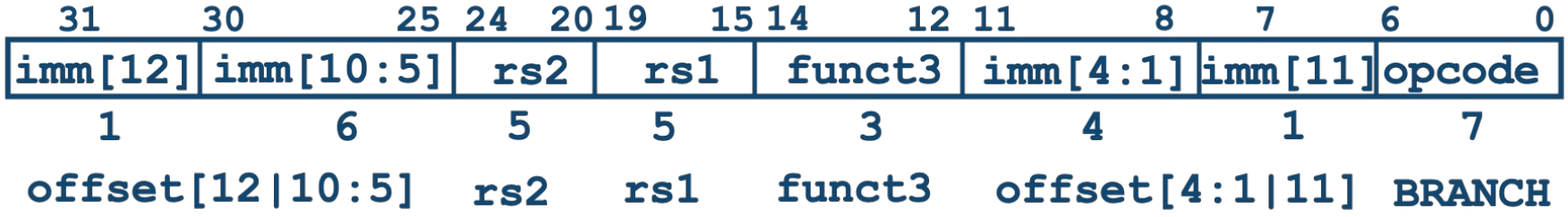
32-bit immediates produced, imm[31:0]



RISC-V ISA

- R-type instructions (register-register arithmetic)
  - add, sub sll, slt, sltu, xor, srl, sra, or, and
- I-type instructions (register-immediate arithmetic)
  - addi, slti, sltiu, xori, ori, andi, slli, srli, srai
- Memory load/store instructions (unsigned vs signed loads)
  - sw, sh, sb
  - lw, lh, lhu, lb, lbu
  - Memory is stored by byte-addressing, little endian
  - What's the difference between lb and lbu?
  - What's the largest memory address?
  - How is address overflow handled?

RISC-V ISA

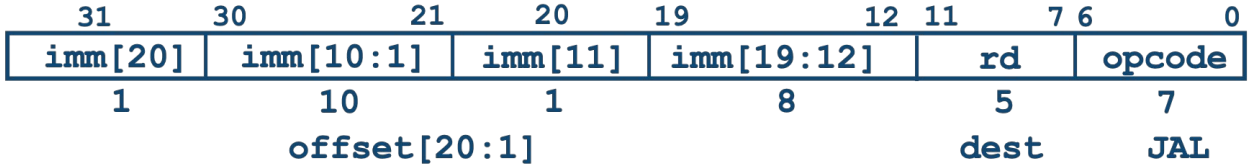


- B-format is mostly same as S-format, with two register sources (rs1/rs2) and a 12-bit immediate
- But now immediate represents values -4096 to +4094 in 2-byte increments
- The 12 immediate bits encode *even* 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it)

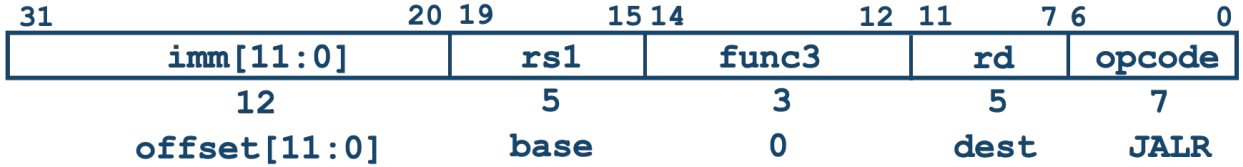
RISC-V ISA

- Branch instructions
  - beq, bne, blt, bge, bltu, bgeu
- How many comparator units are required in the branch unit?
- What's the maximum branch jumping range?
  - +- 4 kiB
  - Branch immediate implicitly sets LSB to 0 (must be a multiple of 2 bytes). Why? Compressed ISA extension
- How are branches resolved in a pipelined datapath?
  - Branch prediction and recovery/flushing on a mispredict
  - Bubble insertion / stalling

RISC-V ISA



- JAL saves PC+4 in register rd (the return address)
- Set PC = PC + offset (PC-relative jump)
- Target somewhere within $\pm 2^{19}$ locations, 2 bytes apart



- JALR Writes PC+4 to rd (return address)
- Sets PC = rs1 + immediate (and sets the LSB to 0)
- Uses same immediates as arithmetic and loads

RISC-V ISA

- Jump instructions
- JAL
  - Jump and link
  - Used commonly to jump within a +- 1 MiB range (PC relative)
  - Link = writeback of PC + 4, used to return from a function call
  - Plain unconditional jumps writeback to x0
- JALR
  - Jump and link register
  - Used less frequently to jump to any address (non-PC relative)
  - LUI + JALR is a common combination to jump across large ranges
  - Can also be paired with AUIPC
- What hazards are introduced by jumps in pipelined datapaths?

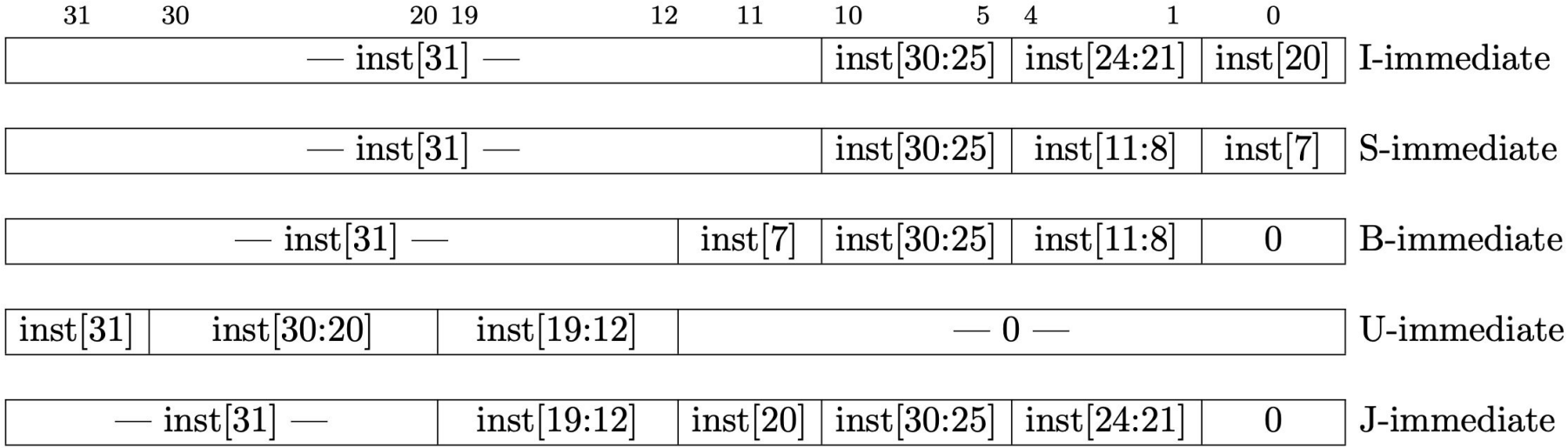
RISC-V ISA

- Register load instructions, 20-bit U immediate
- AUIPC
  - rd = PC + imm
  - Can be used to fetch the current PC
- LUI
  - rd = imm << 12

RISC-V Assembly

- Most R-type instructions are written as
  - add rd, rs1, rs2
  - xor rd, rs1, rs2
- I-type instructions
  - addi rd, rs1, 1024
- Stores
  - sw x1, imm(x2) (x1 = data, x2 = base address)
- Loads
  - lw x1, imm(x2) (x1 = dest register, x2 = base address)
- Jumps
  - jal x0, label
  - labels can be added to any line of assembly
  - label: add x1, x2, x3

Immediate Decoding



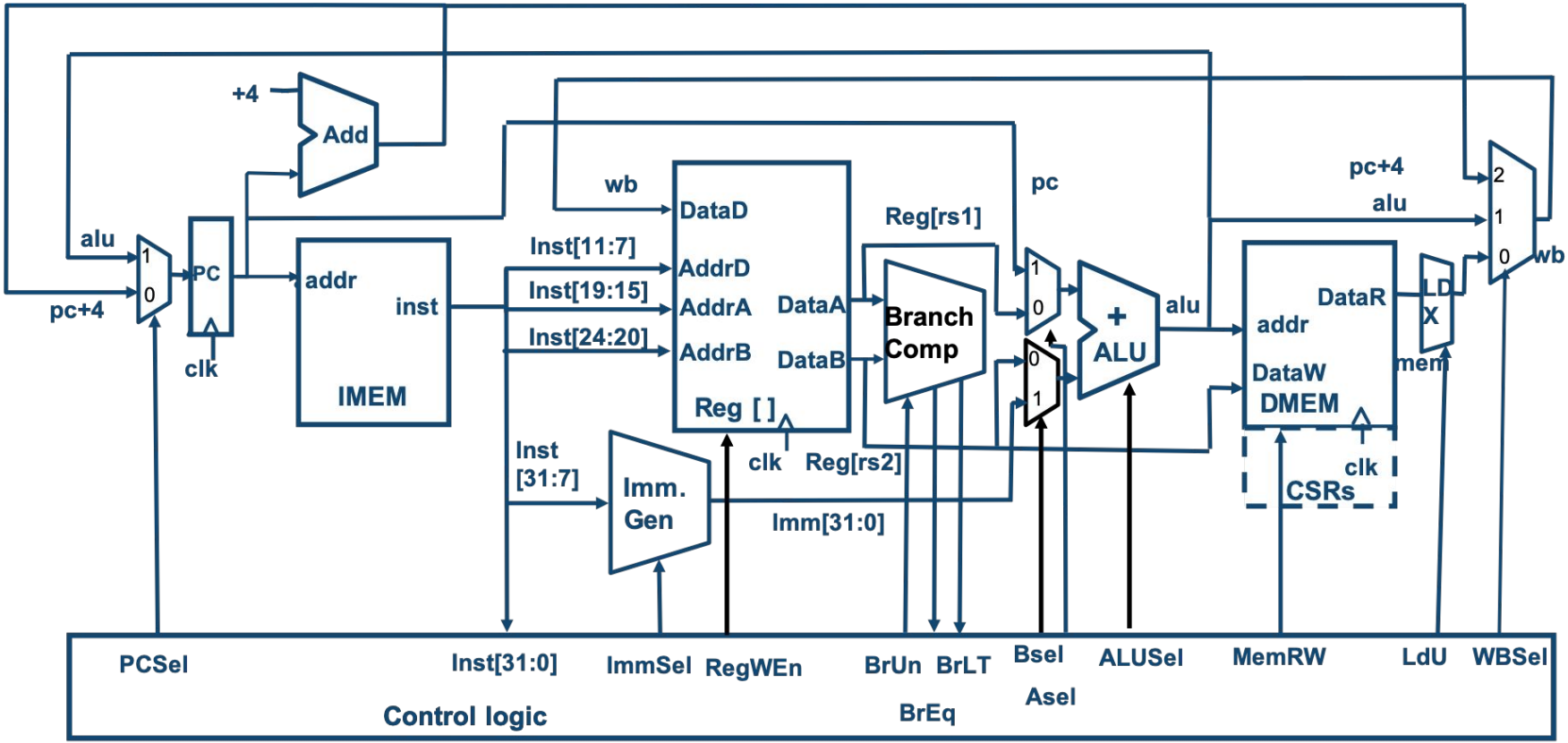
Immediate Decoding

- Note that all immediates are sign extended
  - The interpretation of the immediate depends on the instruction (slti, sltiu)
- The immediate decoder is just constructing all 5 immediate types and muxing them out based on the opcode

```
wire [31:0] i_imm = {21{inst[31]}, inst[30:25], inst[24:21], inst[20]};
wire [31:0] u_imm = {inst[31], inst[30:20], inst[19:12], 12{1'b0}};
```

- Then use a case statement to choose the actual immediate

Basic RISC-V Single-Cycle Datapath



Control Unit

- Design a control unit in Verilog to generate the control signals for the standard datapath
- Take PCSel: for what instructions should we mux in the ALU output into the PC?
  - ALU used to compute jump and branch target addresses
- Pipelining leads to hazards
  - Data hazards (read after write)
    - ALU -> ALU dependency (add then add)
    - Memory -> ALU dependency (lw then add)
    - ALU -> Memory dependency (add then sw)
    - Memory -> Memory dependency (lw then sw)
  - Control hazards (jal, jalr, branch)

Sp17 Midterm 1 #1a

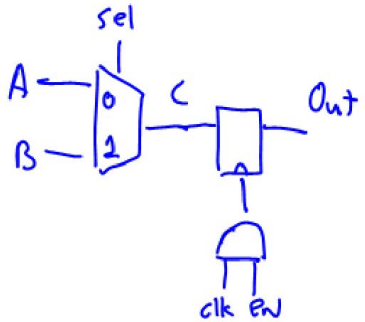
a) (**6 pts**) Draw a gate-level circuit diagram that would implement the behavior described by the Verilog code below.

```
module alpha(A, B, sel, clk, en, out)
    input A, B, sel, clk, en;
    output out;
    reg C, out;
    wire gclk;

    always @(A or B or sel) begin
            C = A;
            if (sel == 1'b1) C = B;
    end

    assign gclk = clk & en;
    always @(posedge gclk) begin
            out <= C;
    end
endmodule
```

Sp17 Midterm 1 #1a

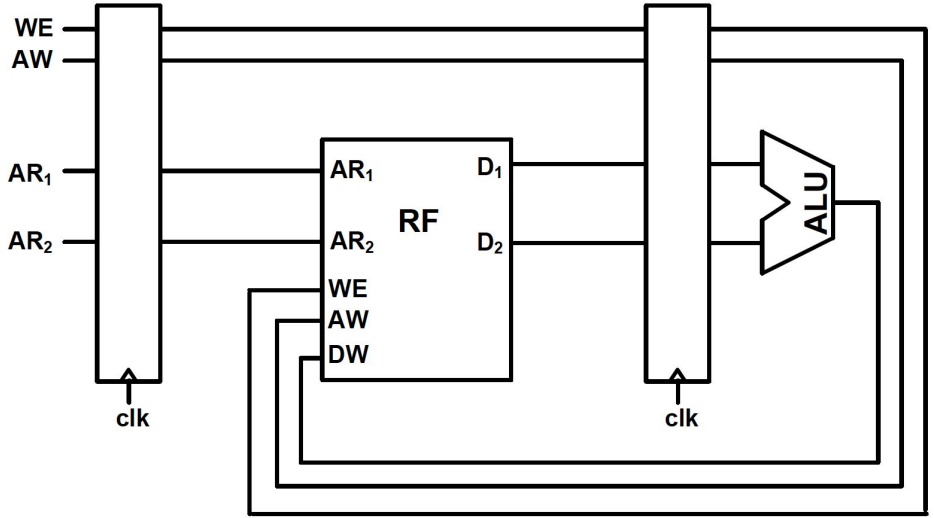
a) (**6 pts**) Draw a gate-level circuit diagram that would implement the behavior described by the Verilog code below.

```
module alpha(A, B, sel, clk, en, out)
    input A, B, sel, clk, en;
    output out;
    reg C, out;
    wire gclk;

    always @(A or B or sel) begin
            C = A;
            if (sel == 1'b1) C = B;
    end

    assign gclk = clk & en;
    always @(posedge gclk) begin
            out <= C;
    end
endmodule
```



Sp17 Midterm 1 #1b

b) **(6 pts)** For this problem we will be analyzing the timing constraints of the portion of a CPU implementation shown below. Note that the details of how this portion of the CPU actually operates are not relevant for this specific problem, but for the sake of clarity, the "RF" represents a Register File, and the "ALU" is an Arithmetic Logic Unit. For all of the flip-flops, assume that t_{clk_q} = 20ps, t_{setup} = 5ps, and t_{hold} = 25ps. Furthermore, you can assume that the delay through either the AR1 or AR2 inputs to the D1 or D2 outputs of the RF is 200ps, that the delay from the WE, AW, and DW inputs to the D1 or D2 outputs of the RF is 75ps, and that the delay from any of the ALU's inputs to the its output is 80ps.



What is the minimum clock period that the design above can support? Does the design suffer from any hold-time constraint violations? (Note that to receive credit for the answer to the second question, you must include any inequalities you used to check to hold-time constraint violations.)

Sp17 Midterm 1 #1b

Minimum cycle:

$$T_{min} = \max(t_{clk\text{-}q} + t_{RF,AR_1 \to D_1} + t_{setup},\ t_{clk\text{-}q} + t_{ALU} + t_{RF,DW} + t_{setup})$$

$$= \max(20ps + 200ps + 5ps,\ 20ps + 80ps + 75ps + 5ps)$$

$$= \boxed{225ps}$$

Hold-time: $t_{clk\text{-}q} \geq t_{hold}$? (on AW or WE paths between the registers)

$20ps \geq 25ps \rightarrow$ No!

Hold time is violated