Discussion 07

Single Cycle Datapath

Aditya Balasubramanian

`aditbala [at] berkeley [dot] edu`

Announcements

Agenda

- SDS Review
- Single Cycle Datapath

SDS Review

Definitions (Pt. 1)

- Clk
  - Central timing unit of the entire SDS; usually only one clock per system
- State element
  - Any clocked element: stores values
  - Only does computation things at the rising edge of the clock
  - E.g. registers
- Logic element
  - Any unclocked elements: does not store value
  - Computes ALL THE TIME!
  - E.g. combinatorial logic elements (AND gates, OR gates, etc.)

Definitions (Pt. 2)

- Flip-flop
  - state element that stores 1 bit's value (0/1)
- Register
  - `n`-bit state element; created with `n` chained FFs
  - `D` = input; `Q` = output
  - Reads value from `D` at clock tick, puts value into `Q`

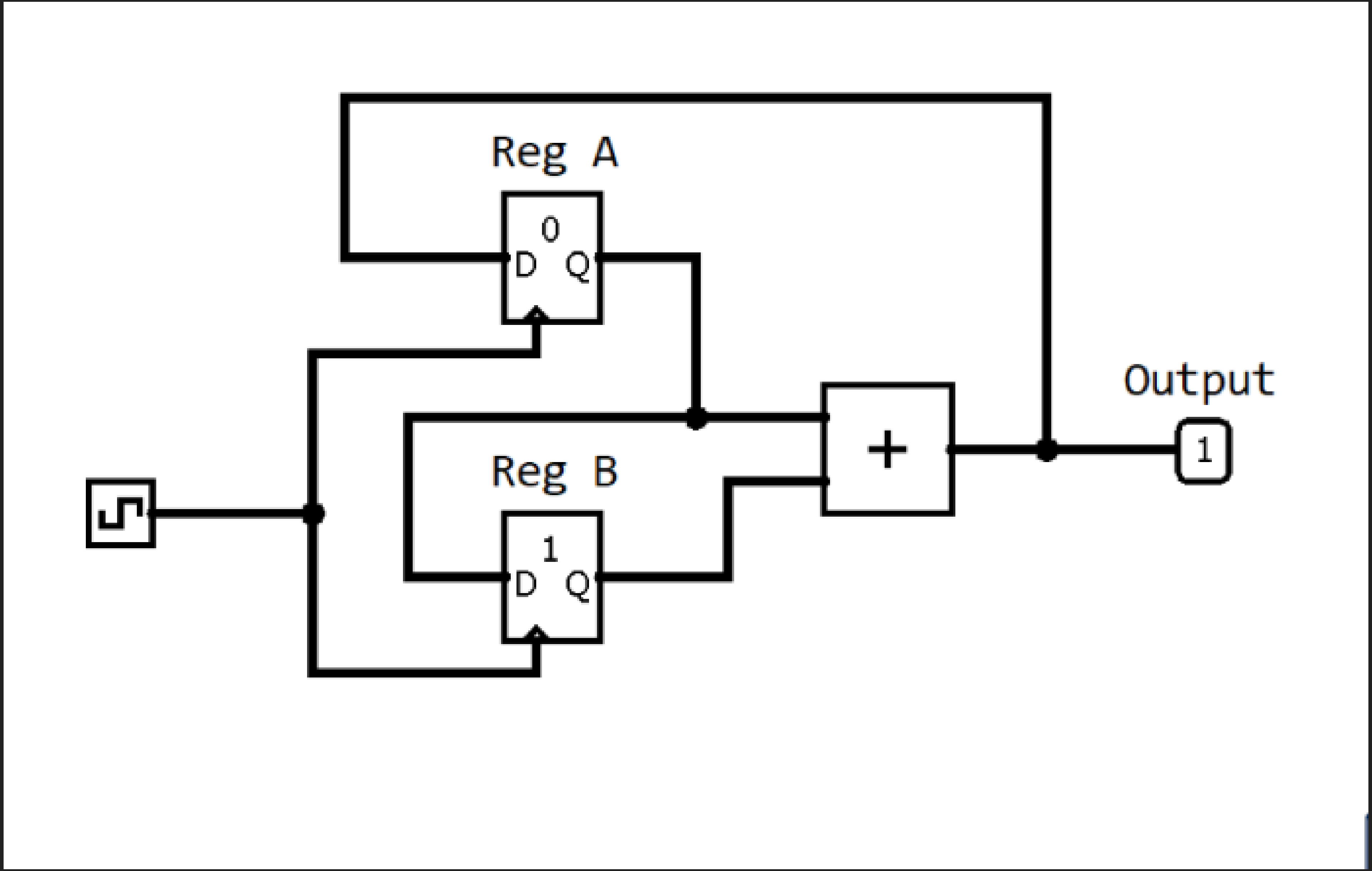
Definitions (Pt. 3)

- Rising clock edge (RCE)
  - clk goes from 0 => 1; Usually instantaneous
  - Triggers all the state elements dependent directly on the clock
- Falling clock edge

  clk goes from 1 => 0
- Setup time
  - Time BEFORE RCE where input must be stable
- Hold time
  - Time AFTER RCE where input must be stable
- Clock-to-q time (c2q)
  - Time after RCE needed for value in Q to change

Definitions (Pt. 4)

- Combinational logic delay
  - Combinatorial delay between 2 state elements
  - Usually Sum of total delays within the path from the Q of one register to the D of another (or the same) register
- Critical path
  - Total delay between 2 state elements
  - Clk-to-q (reg1) + longest CL + setup time (reg2)
- Maximum clock frequency
  - 1 / minimum clock period


Reg A
0
D
Q
Reg B
1
D
Q
+
Output
1

Equations

$$\text{max hold time} \leq t_{\text{clock to q}} + \text{shortest CL}$$

- Any longer of a hold time means that value has potential to change

$$\text{cycle time} \geq t_{\text{clock to q}} + \text{longest CL} + t_{\text{setup}}$$

- Cycle time = clock period
- Any shorter cycle time means the values may not finish computing correctly in time

Single Cycle Datapath

What Even is a datapath?

- Given a program counter (the address of the instruction we need to execute)
- Update the registers to reflect the given instruction
- Every single update we need to make will be in the RISC-V Reference Card: VERY USEFUL
- Standard datapath accounts for all real RISC-V 32bit instructions
- No pseudoinstructions

Datapath Basics

- Wiring accounts for flow of data
- Control logic block does the hard work of "thinking"
- CL takes data/signal in from datapath ⇒ arrows point towards control logic
    - Instr[31:0]; BrEq; BrLT
- CL feeds signal back to datapath ⇒ arrows point away from control logic
    - Calculates signals to tell datapath what components of datapath to use
    - All other signals in standard datapath

Stages of a Single Cycle Datapath

Instruction Fetch (IF)

1. Update the value in PC
    - (**PCSel** = 0) PC = PC + 4
    - (**PCSel** = 1) PC = ALU output
2. Fetch instruction from IMEM at PC
    - IMEM is “Instruction memory”, generally in the code section!

Instruction Decode (ID)

1. Parse instruction from 32 bits binary into useful signals
2. Access RegFile with `rs1`, `rs2`, and `rd`
   a. We can obtain the register numbers from the reference card
   RegFile output the data that is read
   b. Access ImmGen using **ImmSel**
3. Generates immediate based on instruction type, and pad them to made full 32 bits

Execute (EX)

- ALU Operation using ALUSel
  - Inputs to the ALU:
    - (ASel = 0): Use `R[rs1]`
    - (ASel = 1): Use `PC`
    - (BSel = 0): Use `R[rs2]`
    - (BSel = 1): Use `Immediate`
  - Does regular ALU operations (for `I` and `R` type instructions)
  - Also does all the adding! (except for `PC + 4`)
- OR Branch Comparator
  - (**BrUn** = 0) Compare `R[rs1]` and `R[rs2]` signed
  - (**BrUn** = 1) Compare `R[rs1]` and `R[rs2]` unsigned
  - Outputs **BrEq** and **BrLt**

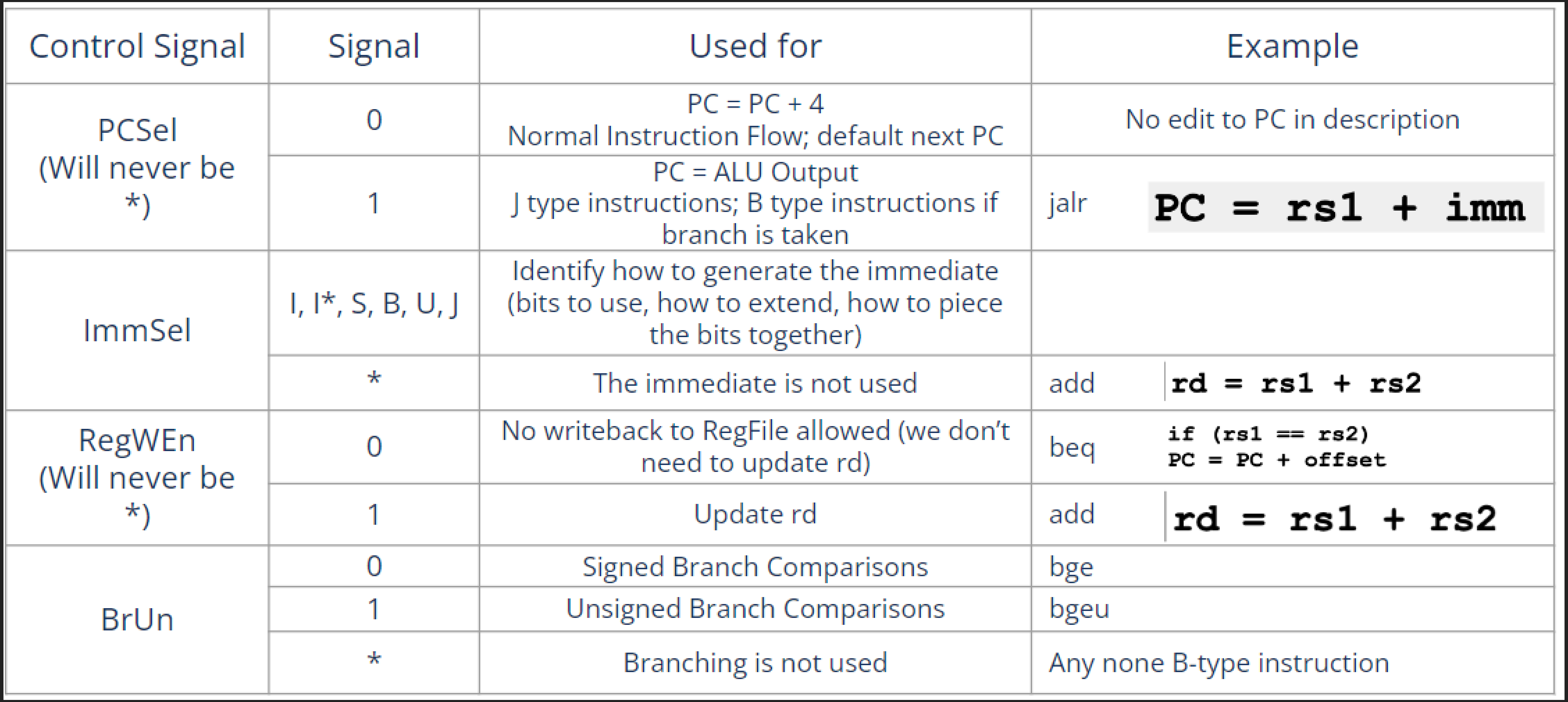
Memory Access (Mem)

1. Access DMEM (Data Memory)
    a. Inputs:
    i. Addr: address we’re accessing, calculated by ALU
    ii. DataW: data we want to put into memory; only used for `S` type instructions
    iii. (**MemRW** = 0) do not write into memory
    iv. (**MemRW** = 1) write into memory

Write Back (WB)

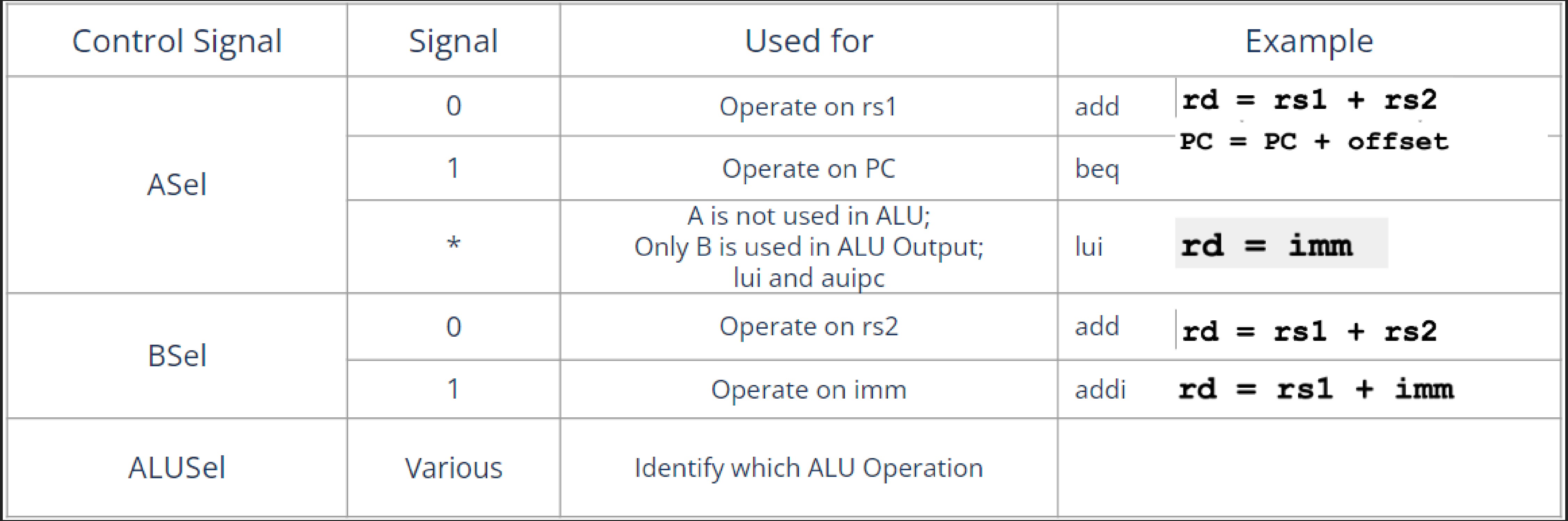
- Selects which value to write to `rd`
  - Inputs:
    - (**WBSel** = 0) Data Memory that is read (with appropriate editing depending on whether we have `lb`, `lh`, `lw`)
    - (**WBSel** = 1) ALU Output
    - (**WBSel** = 2) PC + 4
- Output a value that goes into DataD of RegFile
  - **RegWEn** determines whether we actually update `rd`

Control Signals



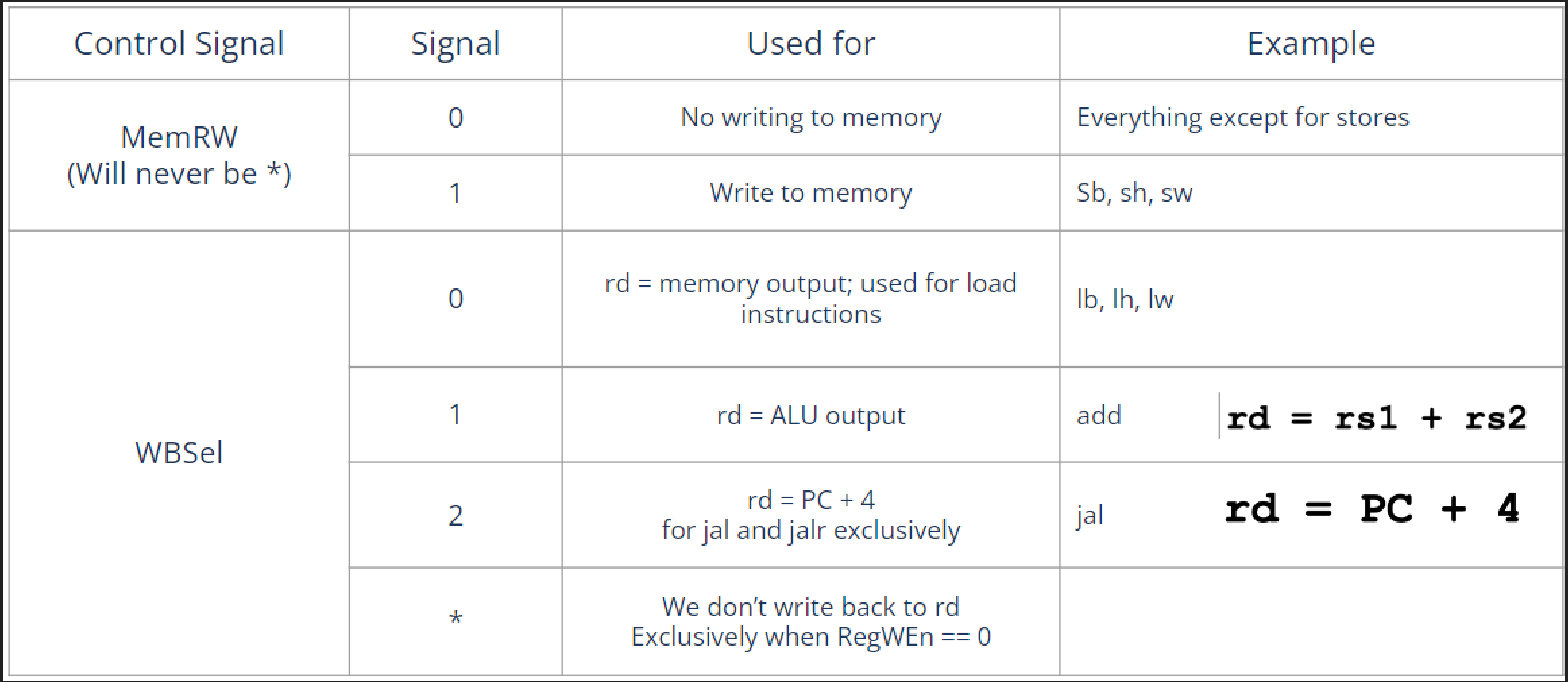
Control Signal	Signal	Used for	Example
PCSel (Will never be *)	0	PC = PC + 4 Normal Instruction Flow; default next PC	No edit to PC in description
	1	PC = ALU Output J type instructions; B type instructions if branch is taken	jalr `PC = rs1 + imm`
ImmSel	I, I*, S, B, U, J	Identify how to generate the immediate (bits to use, how to extend, how to piece the bits together)	
	*	The immediate is not used	add `rd = rs1 + rs2`
RegWEn (Will never be *)	0	No writeback to RegFile allowed (we don't need to update rd)	beq `if (rs1 == rs2)` `PC = PC + offset`
	1	Update rd	add `rd = rs1 + rs2`
BrUn	0	Signed Branch Comparisons	bge
	1	Unsigned Branch Comparisons	bgeu
	*	Branching is not used	Any none B-type instruction

Control Signals Pt. 2



Control Signal	Signal	Used for	Example	
ASel	0	Operate on rs1	add	`rd = rs1 + rs2`
	1	Operate on PC	beq	`PC = PC + offset`
	*	A is not used in ALU; Only B is used in ALU Output; lui and auipc	lui	`rd = imm`
BSel	0	Operate on rs2	add	`rd = rs1 + rs2`
	1	Operate on imm	addi	`rd = rs1 + imm`
ALUSel	Various	Identify which ALU Operation		

Control Signals Pt. 3



Control Signal	Signal	Used for	Example
MemRW (Will never be *)	0	No writing to memory	Everything except for stores
	1	Write to memory	Sb, sh, sw
WBSel	0	rd = memory output; used for load instructions	lb, lh, lw
	1	rd = ALU output	add `rd = rs1 + rs2`
	2	rd = PC + 4 for jal and jalr exclusively	jal `rd = PC + 4`
	*	We don't write back to rd Exclusively when RegWEn == 0	

Thank you!