Lecture 08

A SINGLE CYCLE PROCESSOR
PART 1

RV64I Arithmetic/Logic Instructions

Here are the arithmetic machine words.

		funct7	funct3	opcode
	add	0000000	000	0110011
→	addw	0000000	000	0111011
	sub	0100000	000	0110011
→	subw	0100000	000	0111011
	slt	0000000	010	0110011
	sltu	0000000	011	0110011

		funct7	funct3	opcode
	addi	-	000	0010011
→	addiw	-	000	0011011
	slti	-	010	0010011
	sltiu	-	011	0011011
	lui	-	-	0110111
	auipc	-	-	0010111

	funct7	funct3	opcode
xor	0000000	100	0110011
or	0000000	110	0110011
and	0000000	111	0110011
xori	-	000	0010011
ori	-	010	0010011
andi	-	011	0010011

	funct6	funct3	opcode
sll	0000000	001	0110011
srl	0000000	101	0110011
sra	0100000	101	0110011
slli	0000000	001	0010011
srli	0000000	101	0010011
srai	0100000	101	0010011

		funct6	funct3	opcode
→	sllw	0000000	001	0111011
→	srlw	0000000	101	0111011
→	sraw	0100000	101	0111011
→	slliw	0000000	001	0011011
→	srliw	0000000	101	0011011
→	sraiw	0100000	101	0011011

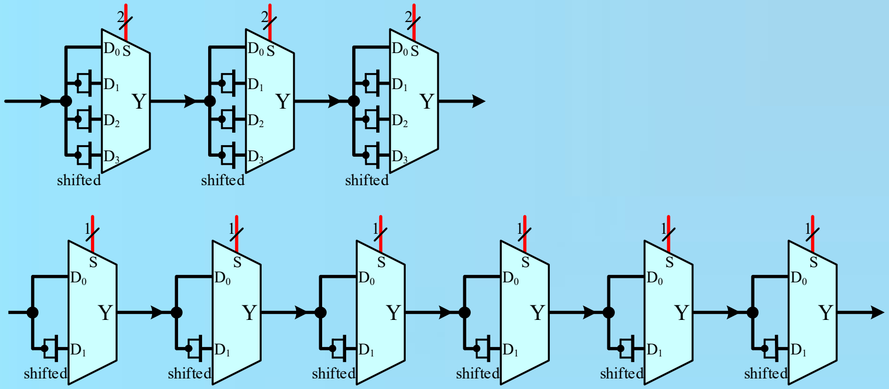
- These are the 21+9 = 30 ALU-type instructions.
- Notice that 9 of these are word instructions. (32-bit)

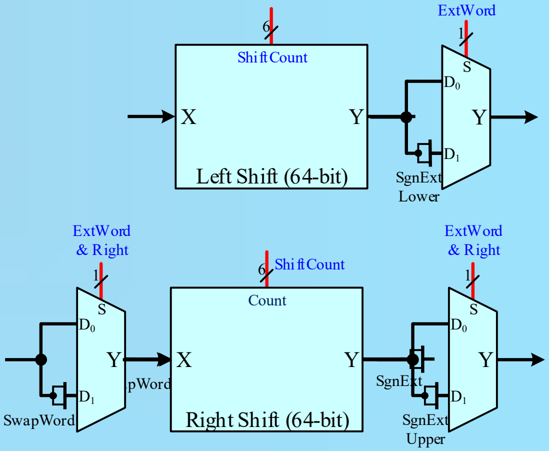
An **RV64I** Barrel Shifter

There are many design alternatives for a barrel shifter.

For **RV64I** we need { `sll`, `srl`, `sra` } { `sllw`, `srlw`, `sraw` }

1) A Chain can be implemented with a combination of 8-channel, 4-channel & 2-Channel MUXes.
2) The Left/Right logical shifts can be combined into a single chain.
3) The `sllw` (32-bit) instruction can be implemented by sign-extending the lower word of the `sll` chain.(64-bit)
4) Right-Shifted Word (32-bit) Instructions can be implemented by transferring the lower word of the input to the upper word before entering a 64-bit chain. The final output would be constructed by sign-extending the upper word at the chain output.
5) The `srlw` and `sraw` (32-bit) instructions can be implemented using method (4)





The barrel shifter timing is not as critical as the arithmetic unit.

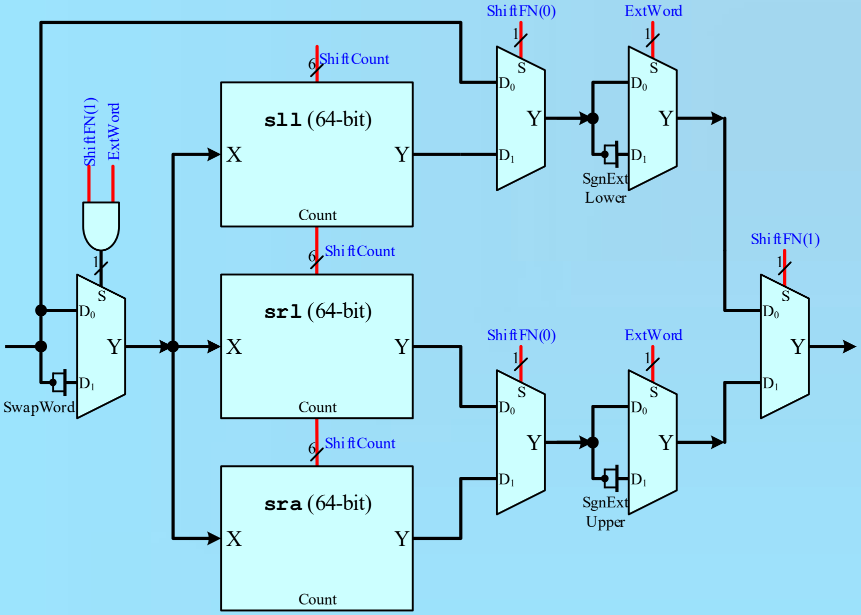
The optimal choice depends upon how the MUXes are to be physically realized.

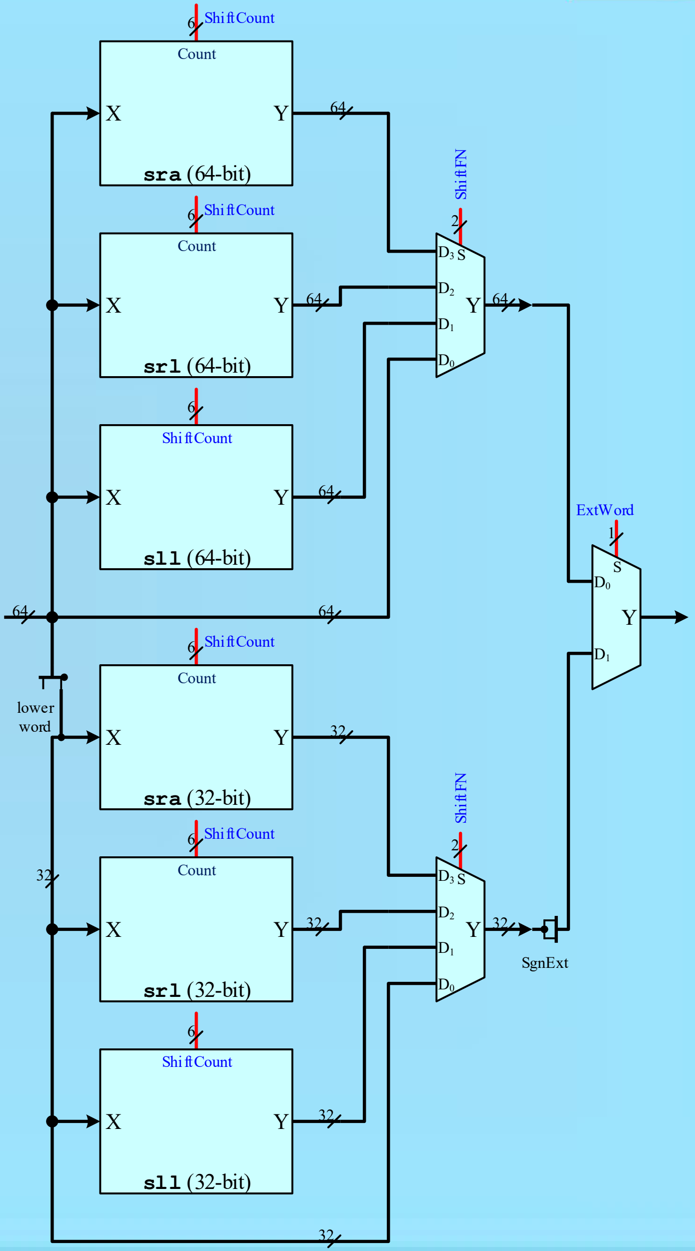
A good engineer will optimise the choice by analysing the resource and timing results after place & route for **all** possible **choices**.

A Simple **RV64I** Barrel Shifter

For your next lab we shall adopt a very primitive solution that has not been optimised.

- All 64-bit instructions,{ **`sll`**, **`srl`**, **`sra`** } are realised as independent chains.
- All 32-bit instructions, { **`sllw`**, **`srlw`**, **`sraw`** } use the 64-bit chains.
- All chains are constructed using 4-channel MUXes.

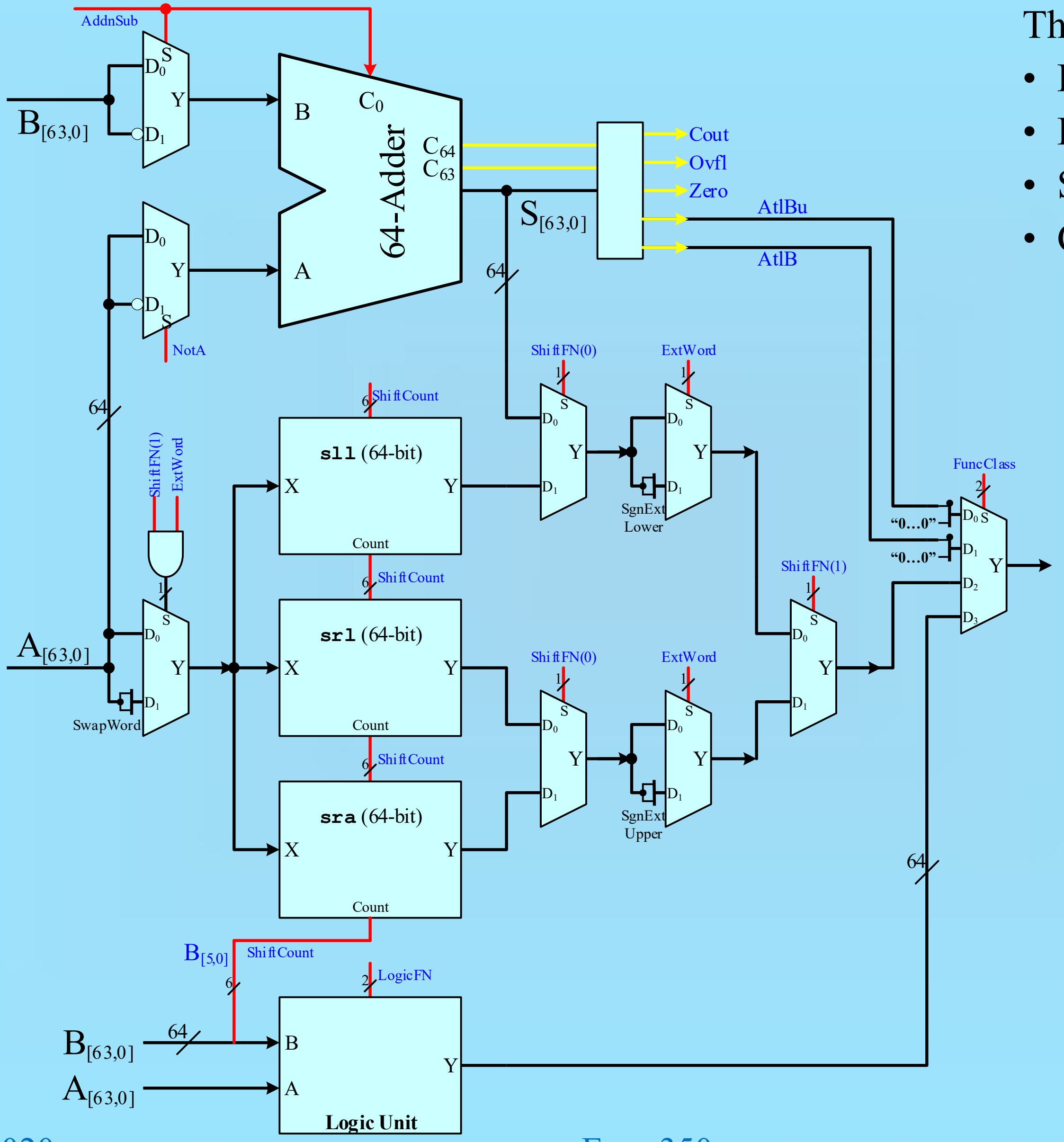




An **RV64I** Execution Unit

The Execution Unit has:

- Data Inputs (black)
- Data Outputs (black)
- Status Outputs (yellow)
- Control Inputs (red)



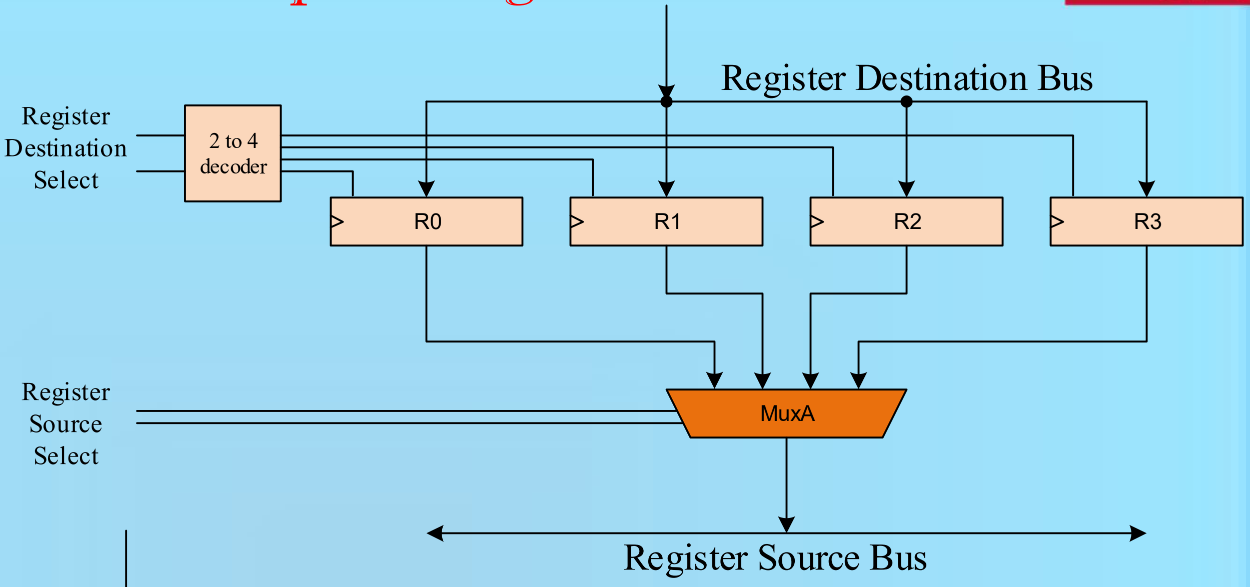
ShiftFN		operation
0	0	**arith**
0	1	**sll**
1	0	**srl**
1	1	**sra**

LogicFn		operation
0	0	Lui
0	1	A xor B
1	0	A or B
1	1	A and B

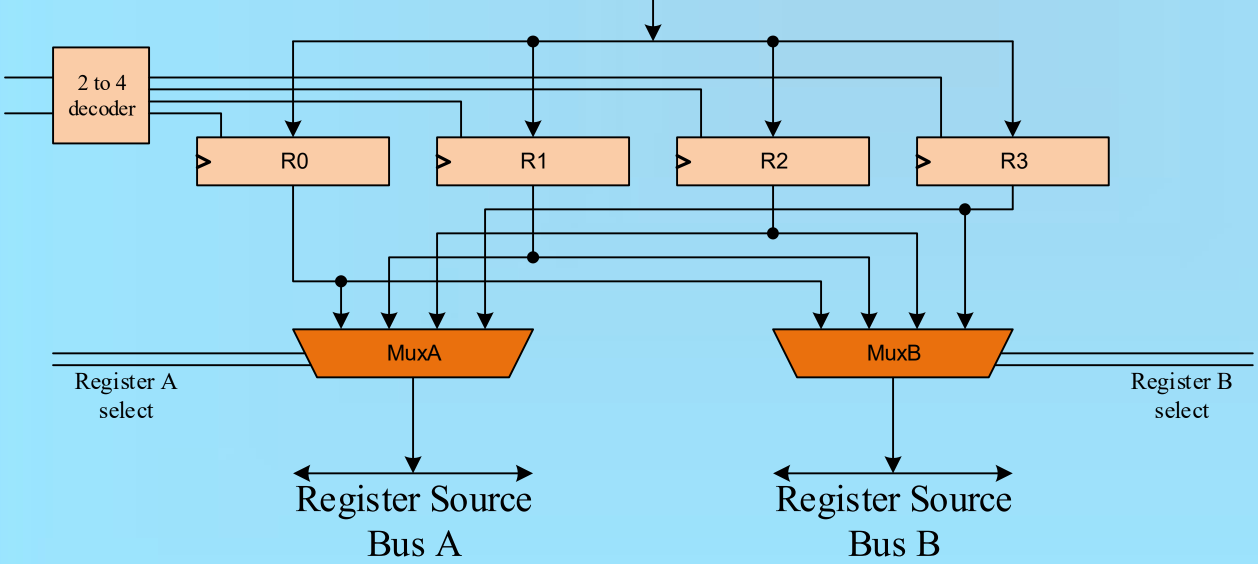
FuncClass		operation
0	0	**sltu**
0	1	**slt**
1	0	**shift/arith**
1	1	**logic**

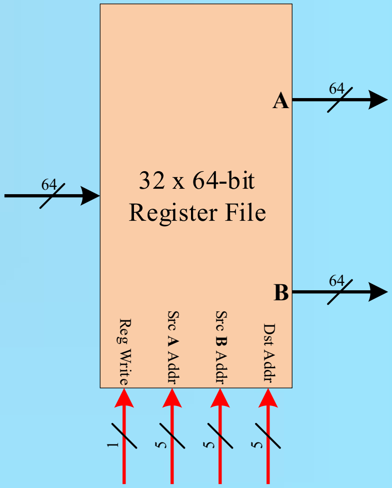
The Datapath Registers

The Registers available to the programmer are contained within a single entity called a Register File.



The device is designed so that two register "reads" and one register "write" actions may occur simultaneously with independent registers.



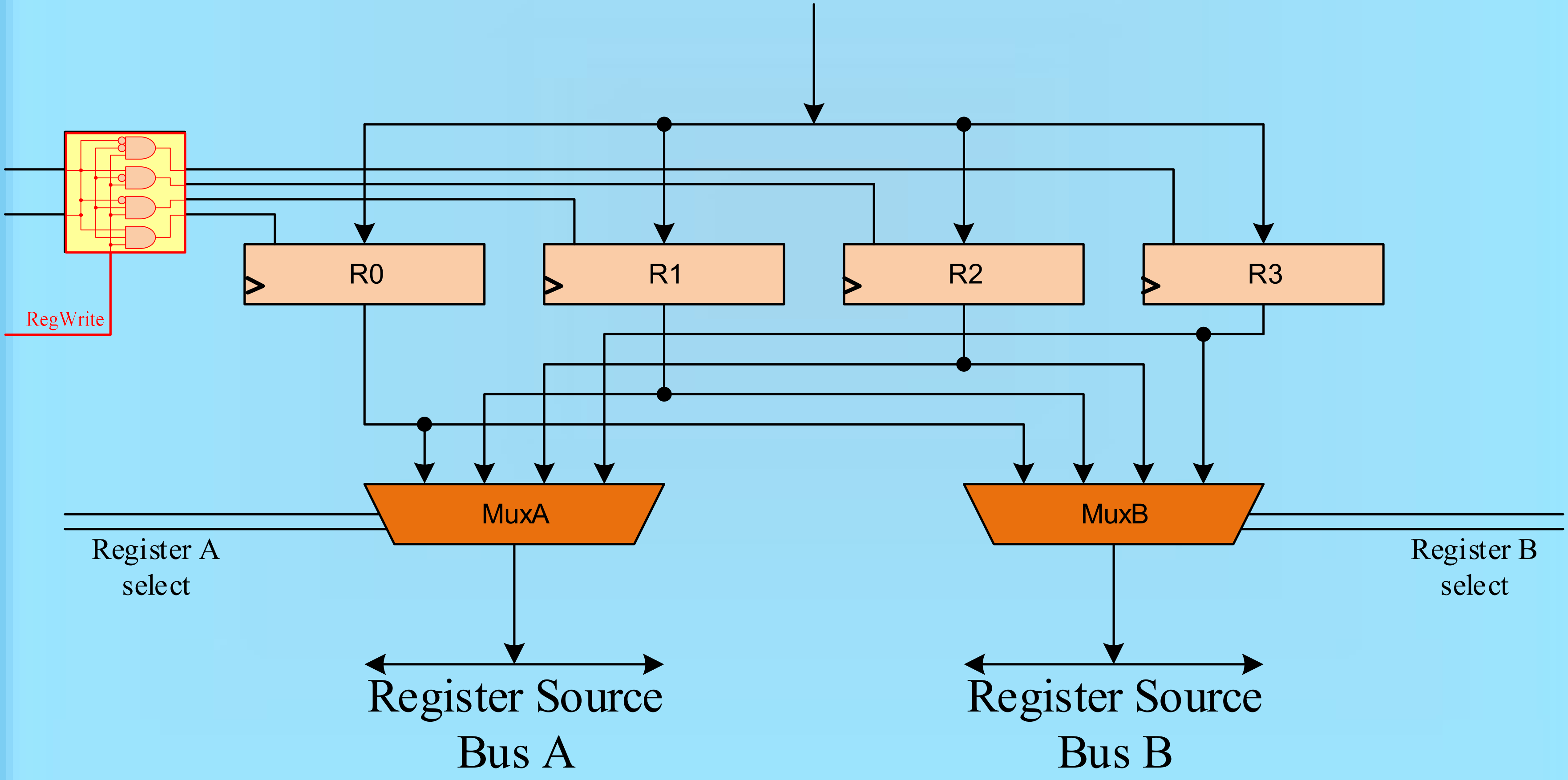


As an exercise you should try to design a register file that has Multiple Destination busses.

The Register File

The Register File Must have an input that enables or disables the action of storing bit patterns into the registers. This is called the **RegWrite** signal.
(When is the bit pattern on the destination bus transferred to the selected register?)

Alternatively, for simplicity and speed the registers may be implemented as **D-type Latches** instead of Flip-flops. (Hennessy & Patterson assumes flip-flops)



Realising Multiplexers

There are many ways to realise Multiplexers.

On modern CMOS integrated circuits, multiplexers are realized using CMOS transmission gates. Each channel only needs a pair of transistors and is thus very small.

- A Cyclone IV FPGA contains 4-input Lookup Tables. (LUTs)
  - A single LUT can implement a 2-channel 1-bit MUX.
  - A pair of LUTs can implements a 4-channel 1-bit MUX. How can this be possible?

Obviously we can extrapolate from here.

An 8-channel Mux uses 5 LUTs and is a 3-level circuit.

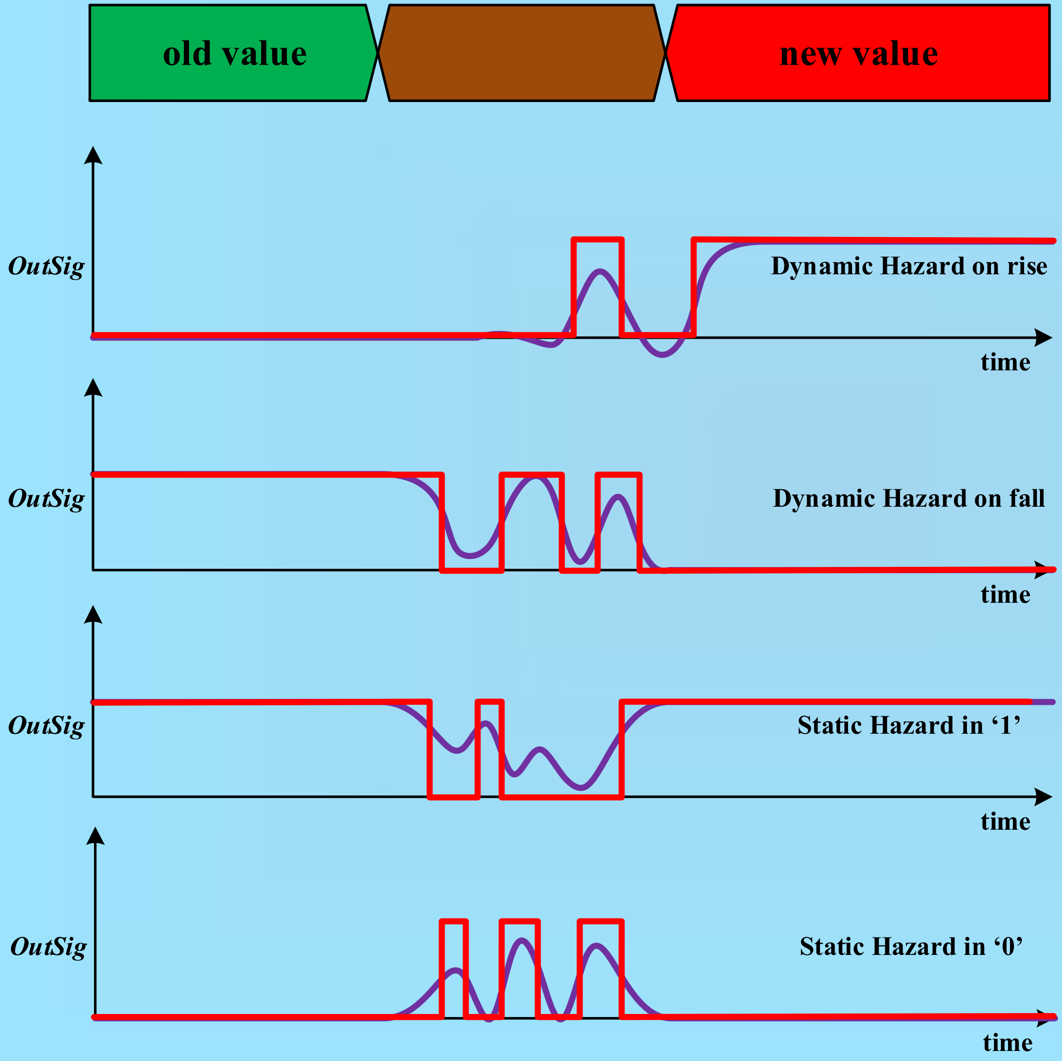
A 16-channel MUX uses 10 LUTs and is a 4-level circuit.

A 32 channel MUX uses

A Typical Designer's Workflow

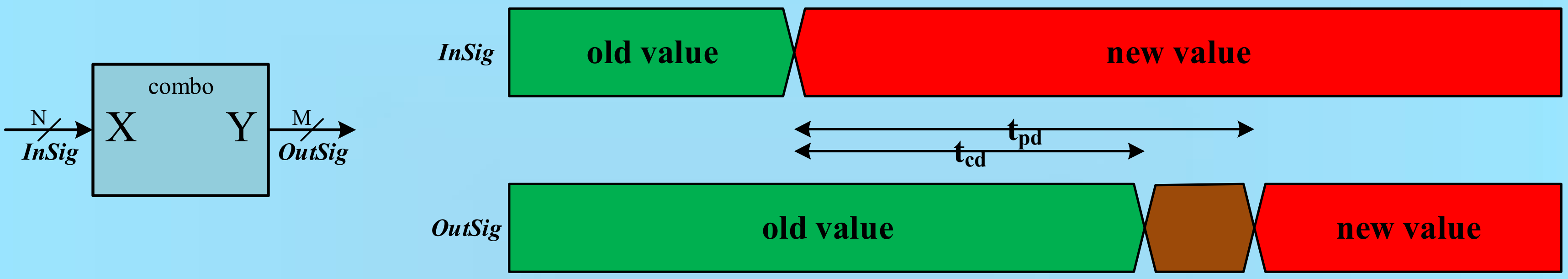
Typical Response of Output Signal Bits

Waveforms at the **output** of a combinational circuit in response to **Events at the input**.



- **Glitches** are the observed spurious pulses. (**symptom**)
- The underlying problem is referred to as a **Hazard**.(**disease**)

Combinational Circuit Timing Parameters



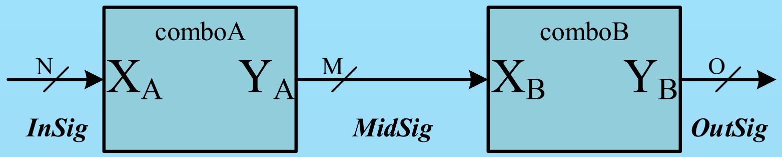
We characterise the timing behaviour of the circuit with two parameters called

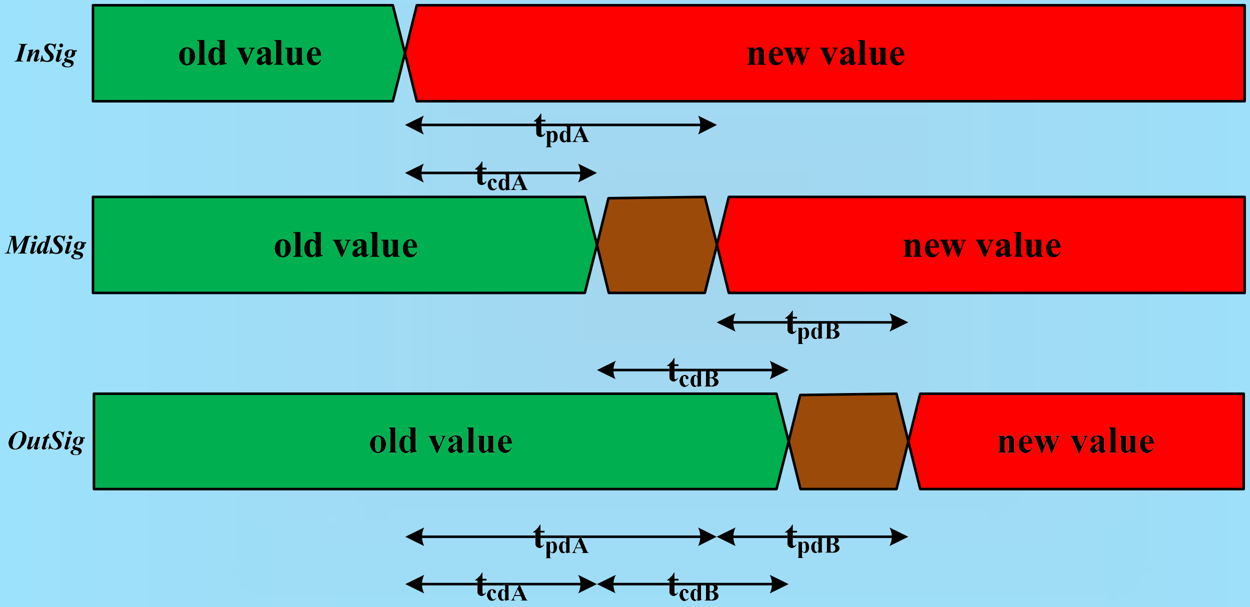
- the propagation delay, $\mathbf{t_{pd}}$ and
- the contamination delay, $\mathbf{t_{cd}}$.

$\mathbf{t_{pd}}$ = longest time before the circuit produces the stable new output value.

$\mathbf{t_{cd}}$ = smallest time that the output value will retain its old value after an input event.

Cascading Combinational Circuits

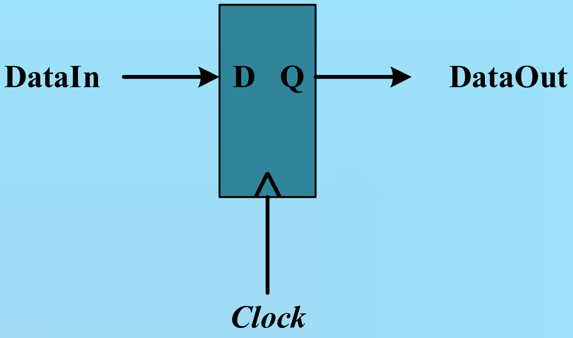


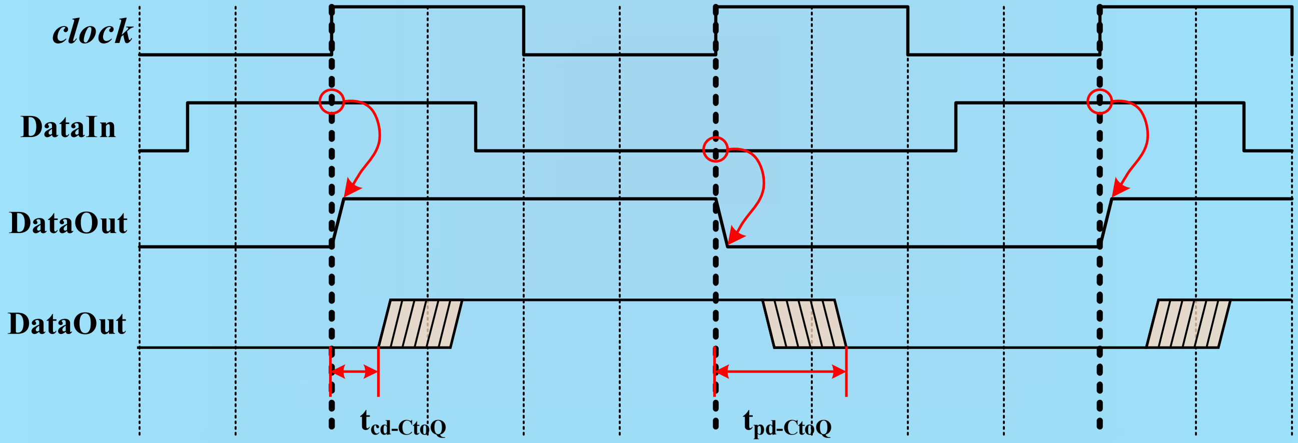


When **cascading** combinational circuits we calculate the overall $\mathbf{t_{pd}}$ and $\mathbf{t_{cd}}$ by simply summing the respective timing parameters of the individual stages.

Flip-Flop Timing Parameters

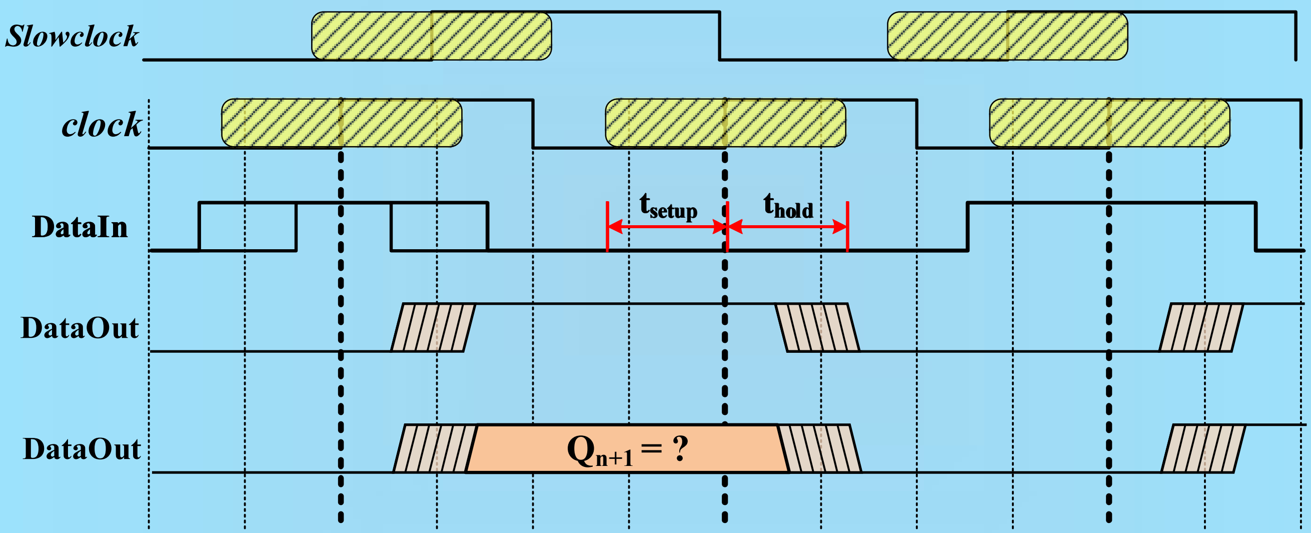
As with combinational circuits, Flip-Flops have a time delay between the clock event and the response at the Q-output.





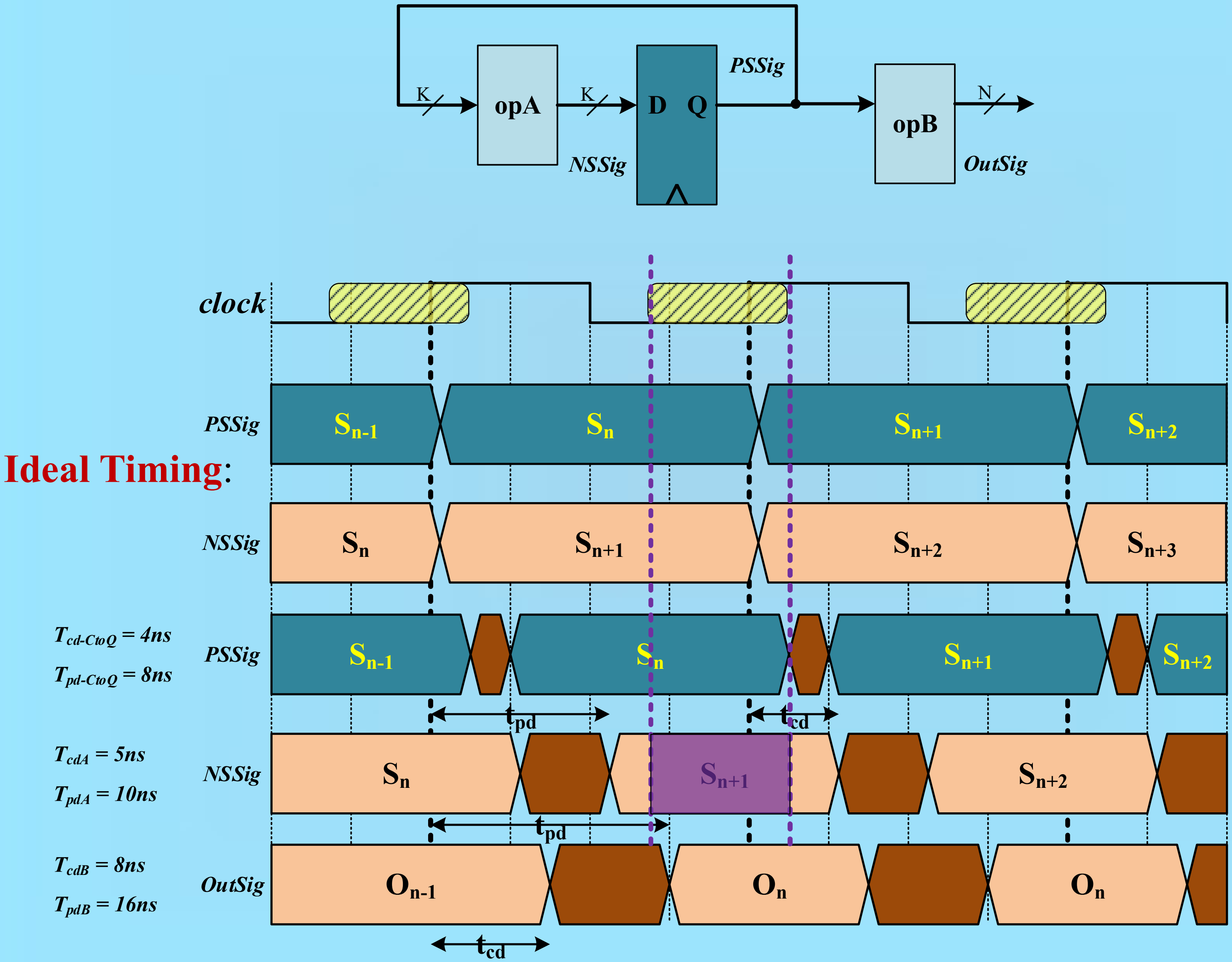
Setup and Hold Requirements

- The Q-output becomes metastable if the **D-input** changes **just before** or **just after** the clock edge event.
- The **setup** time and **hold** time are **timing requirements**; they are NOT delays.



- By slowing the clock (increasing the period) we reduce the relative amount of time between the “Call window” and the Clock period.

Timing for Counters (Sequencers)



Ideal Timing:

$T_{cd\text{-}CtoQ} = 4ns$

$T_{pd\text{-}CtoQ} = 8ns$

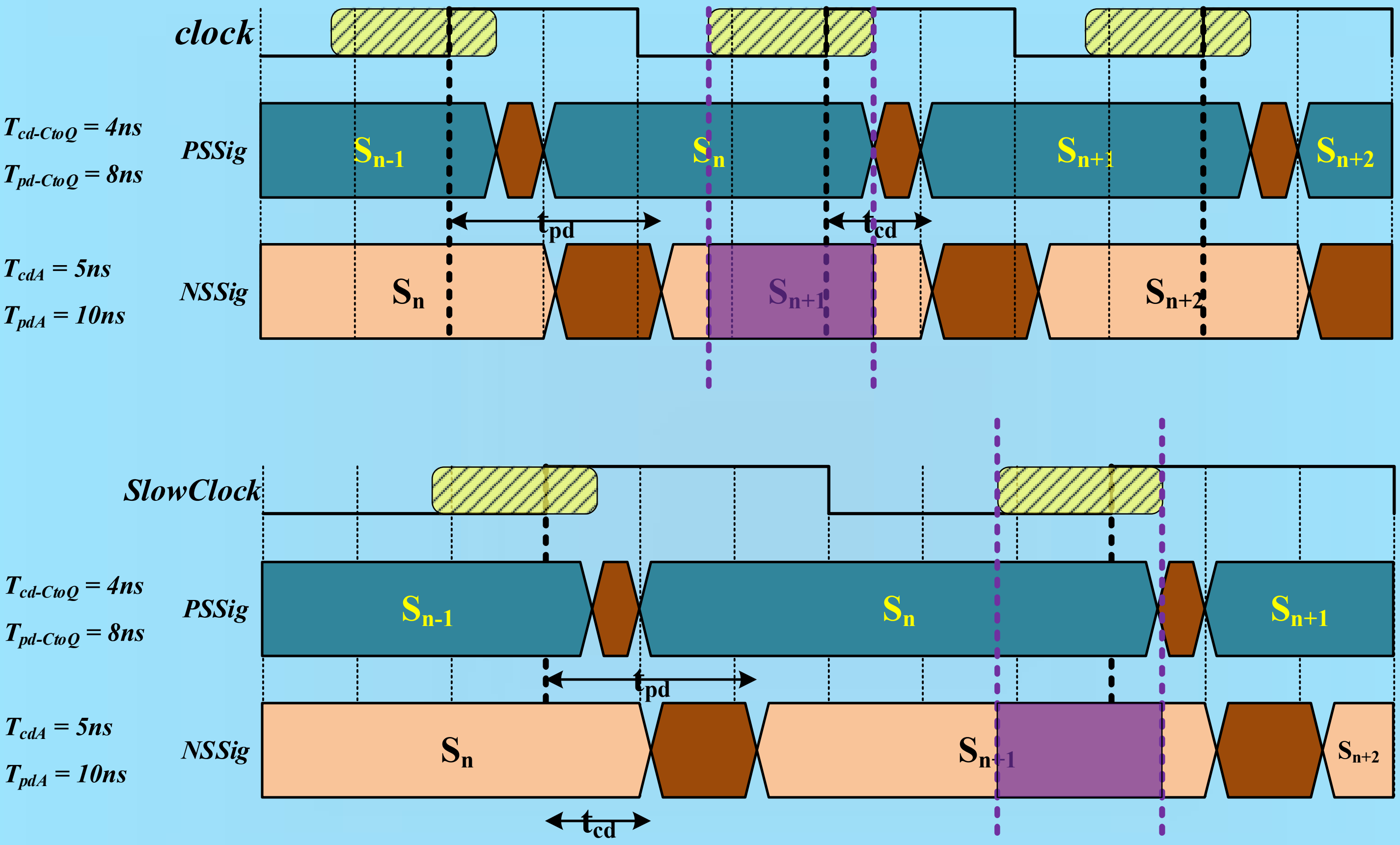
$T_{cdA} = 5ns$

$T_{pdA} = 10ns$

$T_{cdB} = 8ns$

$T_{pdB} = 16ns$

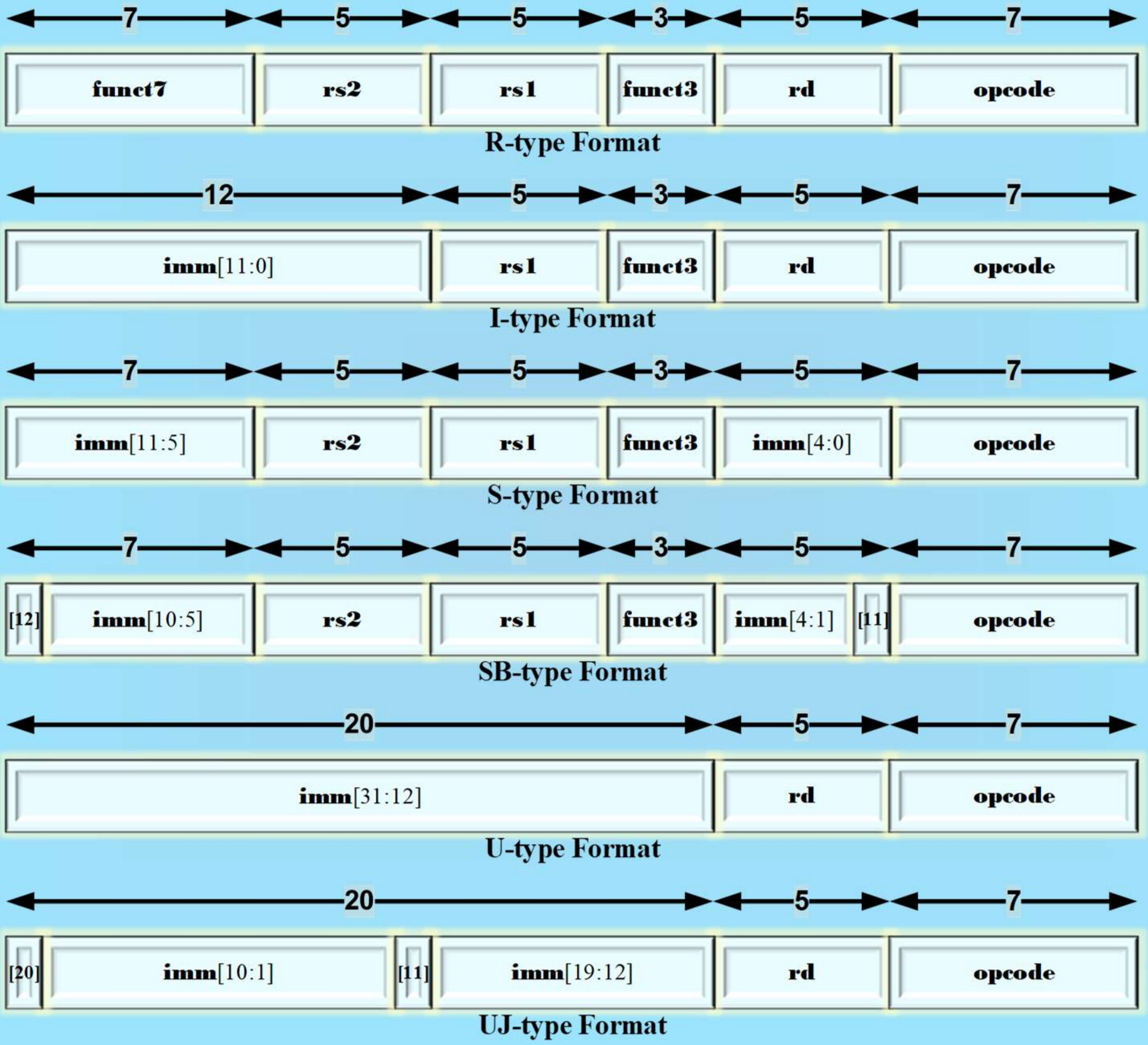
Calculating Maximum Clock Frequency



We can Avoid Setup time violations by simply slowing the clock frequency.

$$T_{clock} > t_{pd\text{-}CtoQ} + t_{pdA} + t_{setup}$$

The RV64I Instruction Formats



7	5	5	3	5	7
funct7	rs2	rs1	funct3	rd	opcode

R-type Format

12	5	3	5	7
imm[11:0]	rs1	funct3	rd	opcode

I-type Format

7	5	5	3	5	7
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

S-type Format

7		5	5	3	5		7
[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	[11]	opcode

SB-type Format

20	5	7
imm[31:12]	rd	opcode

U-type Format

20				5	7
[20]	imm[10:1]	[11]	imm[19:12]	rd	opcode

UJ-type Format

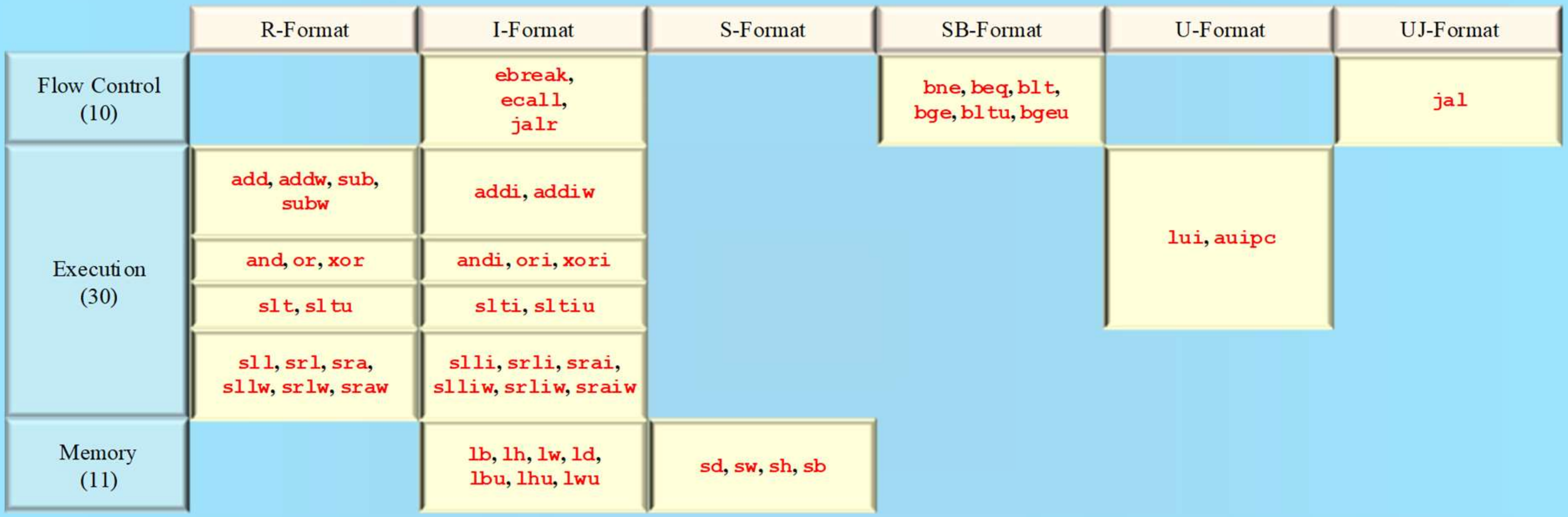
Instruction Classes

There are 59 instructions in rv64I.

I have ignored 8 of these instructions. { fence, fenci.i } (thread synchronization) &

{ csrrs, csrrsi, csrrw, csrrwi, csrrc, csrrci } (control/status register - read/write, read/set, read/clear)

We will design a processor that can execute the 51 remaining instructions.



	R-Format	I-Format	S-Format	SB-Format	U-Format	UJ-Format
Flow Control (10)		ebreak, ecall, jalr		bne, beq, blt, bge, bltu, bgeu		jal
Execution (30)	add, addw, sub, subw	addi, addiw			lui, auipc	
	and, or, xor	andi, ori, xori				
	slt, sltu	slti, sltiu				
	sll, srl, sra, sllw, srlw, sraw	slli, srli, srai, slliw, srliw, sraiw				
Memory (11)		lb, lh, lw, ld, lbu, lhu, lwu	sd, sw, sh, sb			

The (51) Instructions

	funct6	funct3	opcode
sllw	0000000	001	0111011
srlw	0000000	101	0111011
sraw	0100000	101	0111011
slliw	0000000	001	0011011
srliw	0000000	101	0011011
sraiw	0100000	101	0011011

	funct6	funct3	opcode
sll	0000000	001	0110011
srl	0000000	101	0110011
sra	0100000	101	0110011
slli	0000000	001	0010011
srli	0000000	101	0010011
srai	0100000	101	0010011

	funct7	funct3	opcode
xor	0000000	100	0110011
or	0000000	110	0110011
and	0000000	111	0110011
xori	-	000	0010011
ori	-	010	0010011
andi	-	011	0010011

	funct7	funct3	opcode
add	0000000	000	0110011
addw	0000000	000	0111011
sub	0100000	000	0110011
subw	0100000	000	0111011
slt	0000000	010	0110011
sltu	0000000	011	0110011

	funct7	funct3	opcode
addi	-	000	0010011
addiw	-	000	0011011
slti	-	010	0010011
sltiu	-	011	0011011
lui	-	-	0110111
auipc	-	-	0010111

	funct7	funct3	opcode
lb	-	000	0000011
lh	-	001	0000011
lw	-	010	0000011
ld	-	011	0000011
lbu	-	100	0000011
lhu	-	101	0000011
lwu	-	110	0000011

	funct7	funct3	opcode
sb	-	000	0100011
sh	-	001	0100011
sw	-	010	0100011
sd	-	011	0100011

	funct7	funct3	opcode
beq	-	000	1100011
bne	-	001	1100011
blt	-	100	1100011
bge	-	101	1100011
bltu	-	110	1100011
bgeu	-	111	1100011

ebreak	imm=0x000	000	1110011
ecall	imm=0x001	000	1110011

jal	-	-	1101111
jalr	-	000	1100111