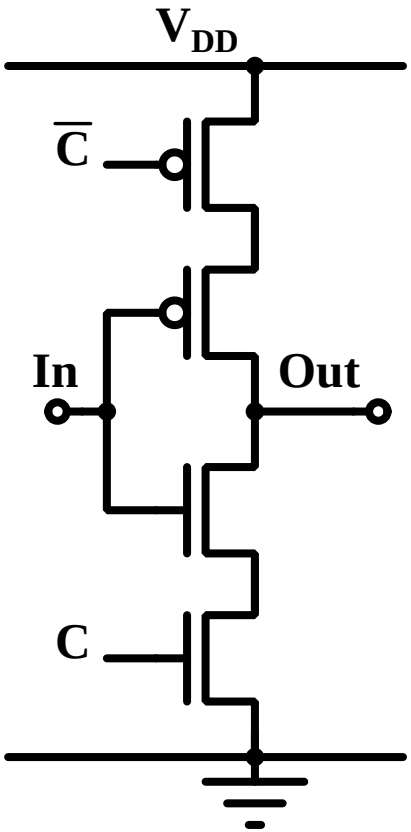
MIPS Data Path Design

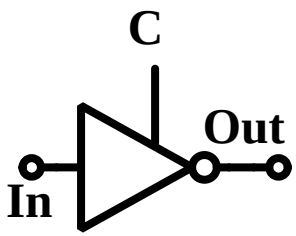
Dinesh Sharma

EE Department
IIT Bombay, Mumbai

April 19, 2021

Tristateable Inverter



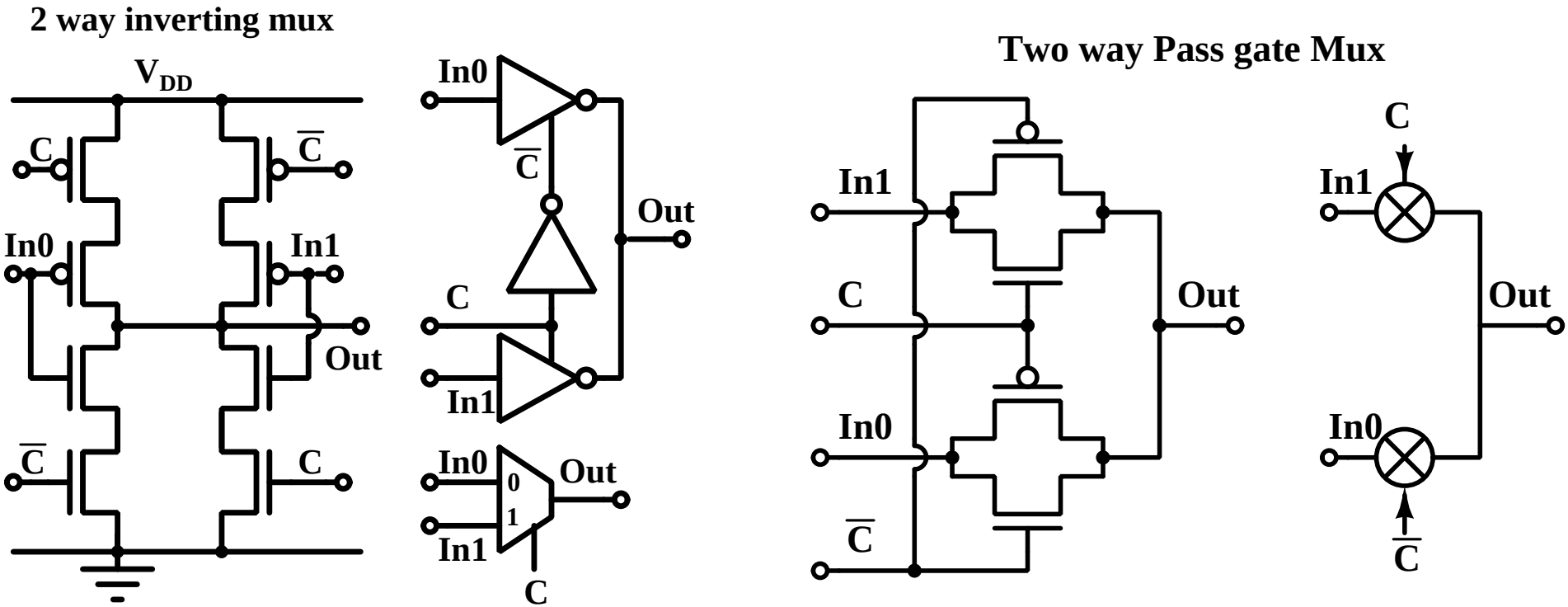


Truth Table

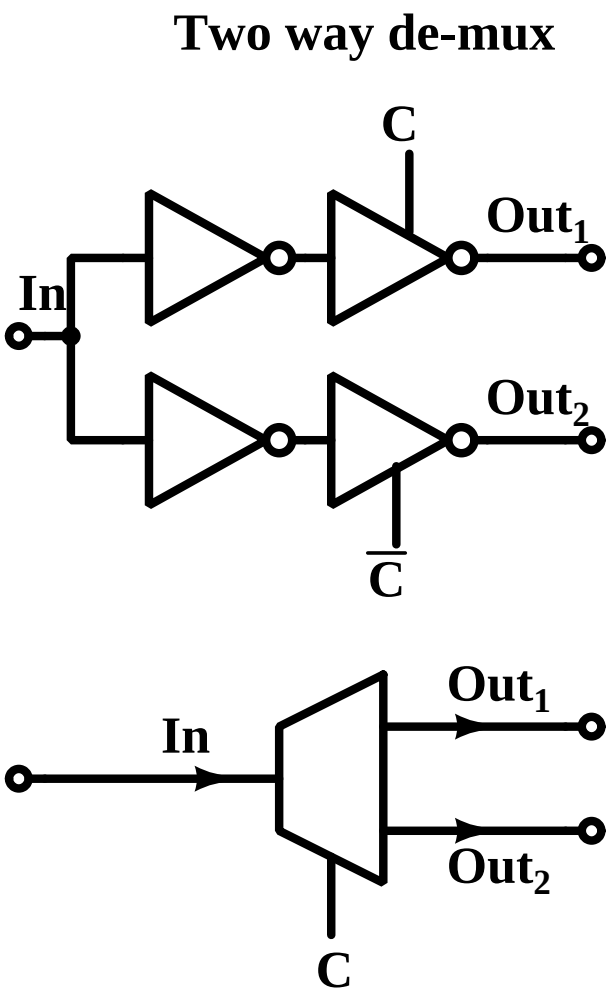
C	In	Out
0	0	Z
0	1	Z
1	0	1
1	1	0

- When C=0, the top pMOS and the bottom nMOS are off. So there is no drive to the output (tri-stated).
- When C=1, the top pMOS and the bottom nMOS are on and connect the middle two transistors, which act line an inverter, to supply.

A multiplexer puts data from one out of many input lines on the output.

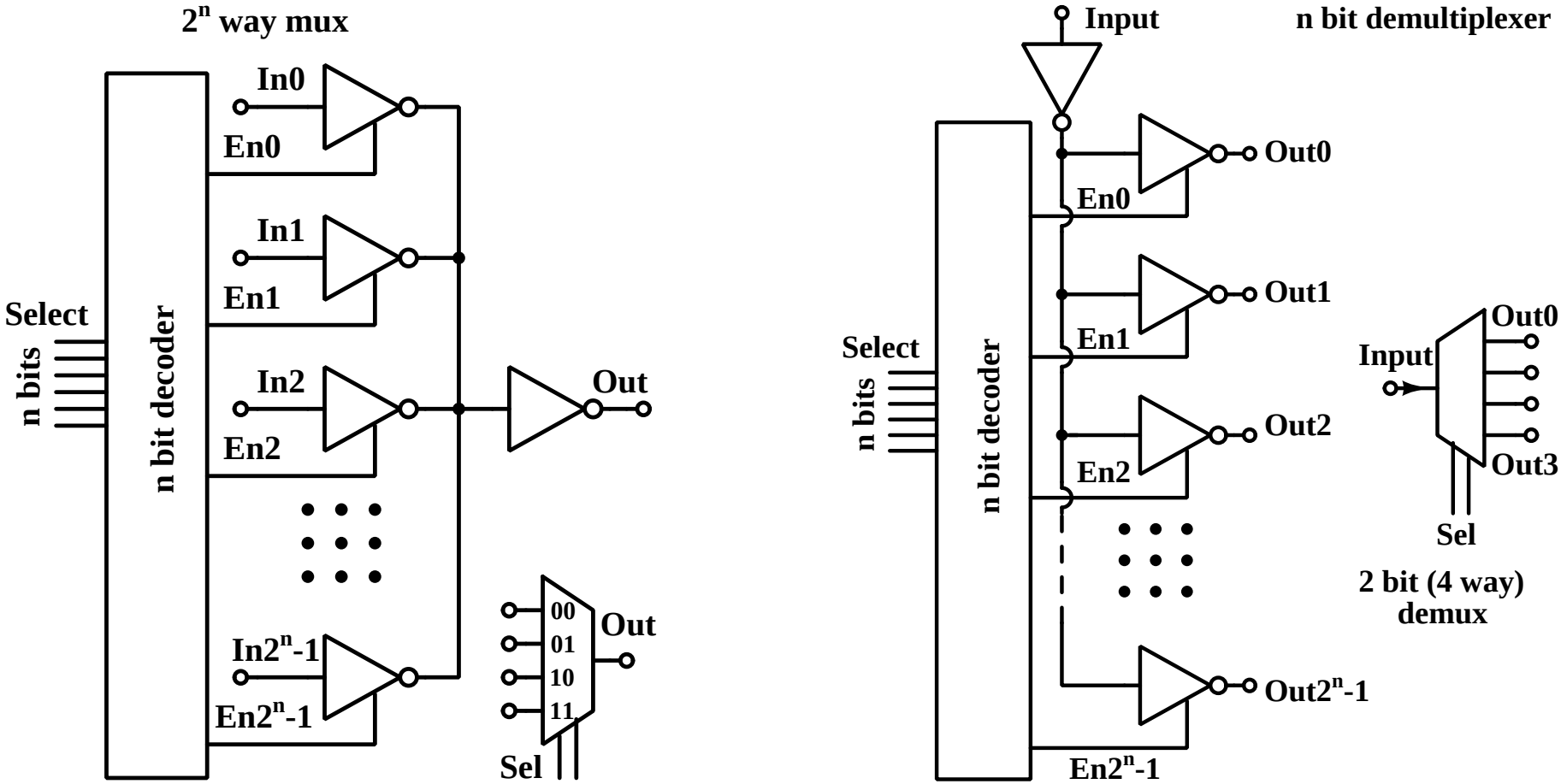


The trapezoidal symbol shown above is used to depict a multiplexer.



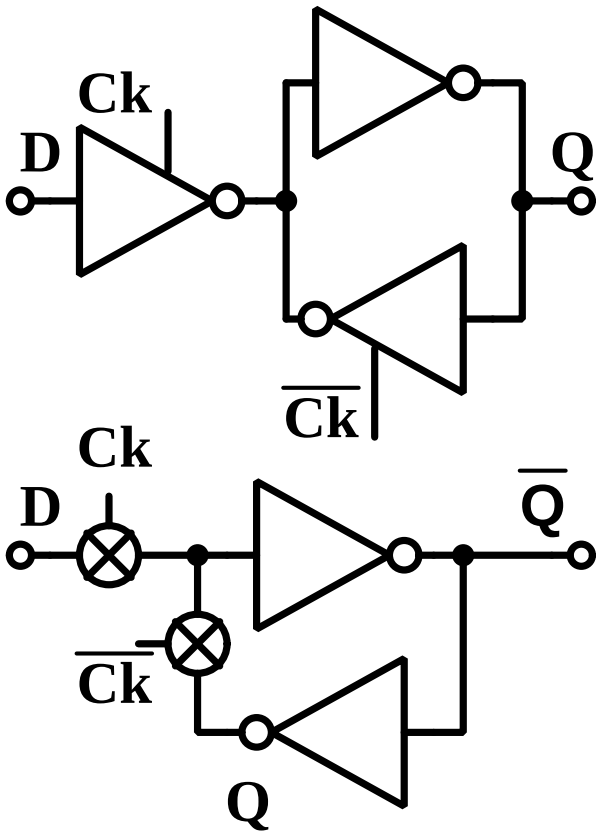
- A demultiplexer copies a common input to one out of many output lines.
- For a two way demux, the control input and its inverted version provide the enable signals for the two output lines.
- A trapezoidal symbol (narrower at the input, wider at the output) is used for a demultiplexer.

Multiplexers can use a decoder to select data out of more than two channels for copying to the output.



Similarly, a demultiplexer can use a decoder to provide a choice of copying data to one of multiple output lines from a common input line.

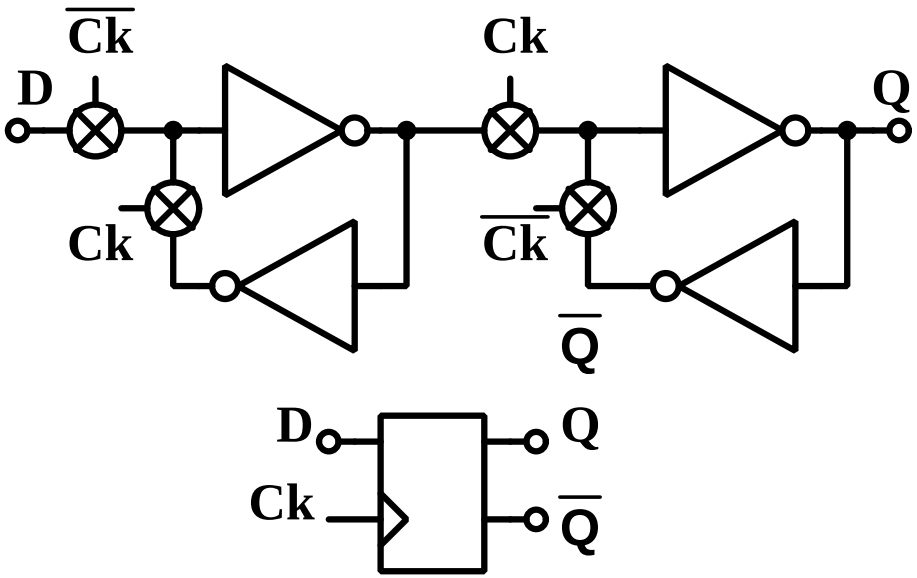
Transparent D latch



- When C = 1, the forward inverter from D is enabled and a buffered version of input appears at the output.
- When C = 0, the inverter from D is tri-stated, while the feedback inverter is enabled. It forms a latch with the forward inverter.
- Data at the input when the input goes from 1 to 0 is stored in the latch.
- Output follows input when C=1 (transparent) and remains latched when C=0.
- A transparent D latch can also be constructed using pass gates.

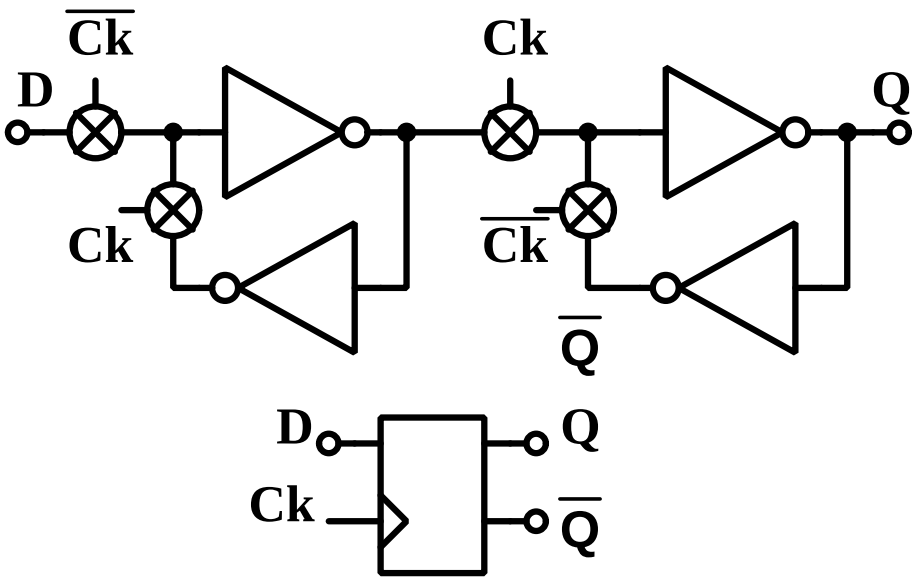
We can form an edge sensitive D flipflop by connecting two transparent latches using complementary clocks connected in master-slave fashion.

Edge Sensitive D flipflop



- When Ck is low, the master latch is transparent, so $\overline{D}$ appears at its output.
- However, the slave is disconnected from its input and is latched to its previous value.

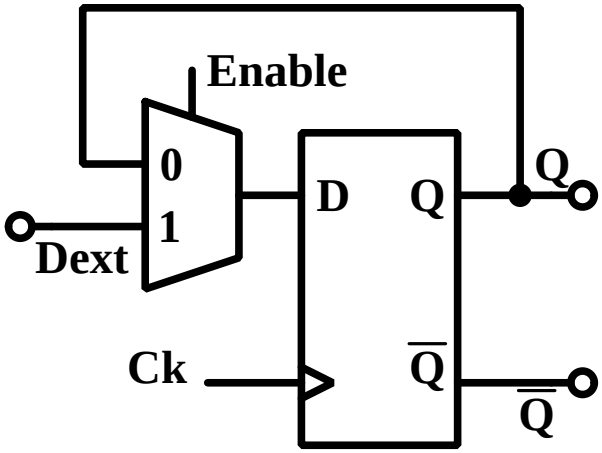
Edge Sensitive D flipflop



- When Ck transitions to high, the master latch goes to the latched state. Its output is latched to the (complement of) the value of D which existed when clock went from 0 to 1 (positive edge).
- At this time, the slave latch goes to follow mode. Thus, the output Q remains at the value that D had at the time of the positive edge on the clock.

When the clock returns to 0, The master follows D, but the slave output remains latched at the sampled value.

A variant of D flipflop is the Enabled or E flipflop

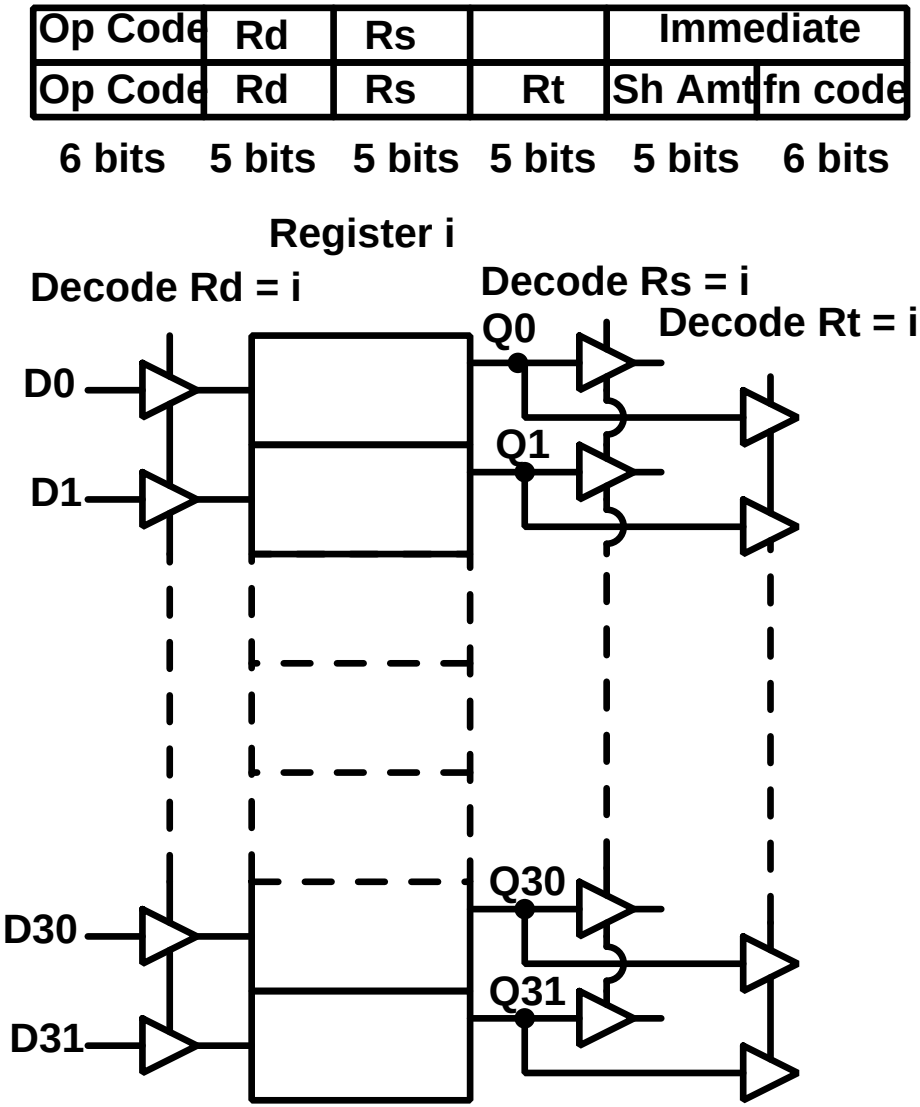


- When Enable = 0, the Q output is connected to D input.
- The flipflop samples its own Q at every positive edge of the clock and thus retains its original value.

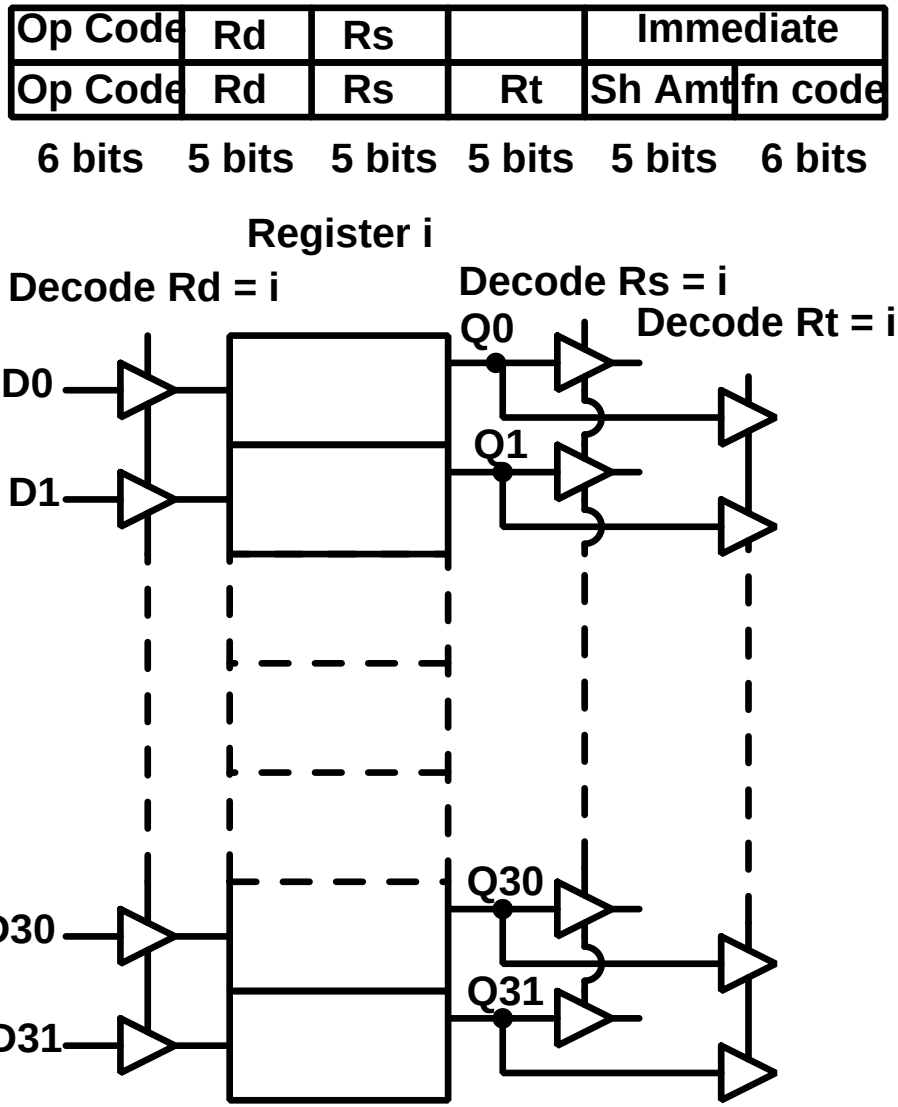
When Enable=1, the external value of D is connected to the D input of the flipflop. Now the circuit acts like a normal d flipflop.

The E flipflop permits one to enable or disable the flipflop from sampling its input without gating the clock.

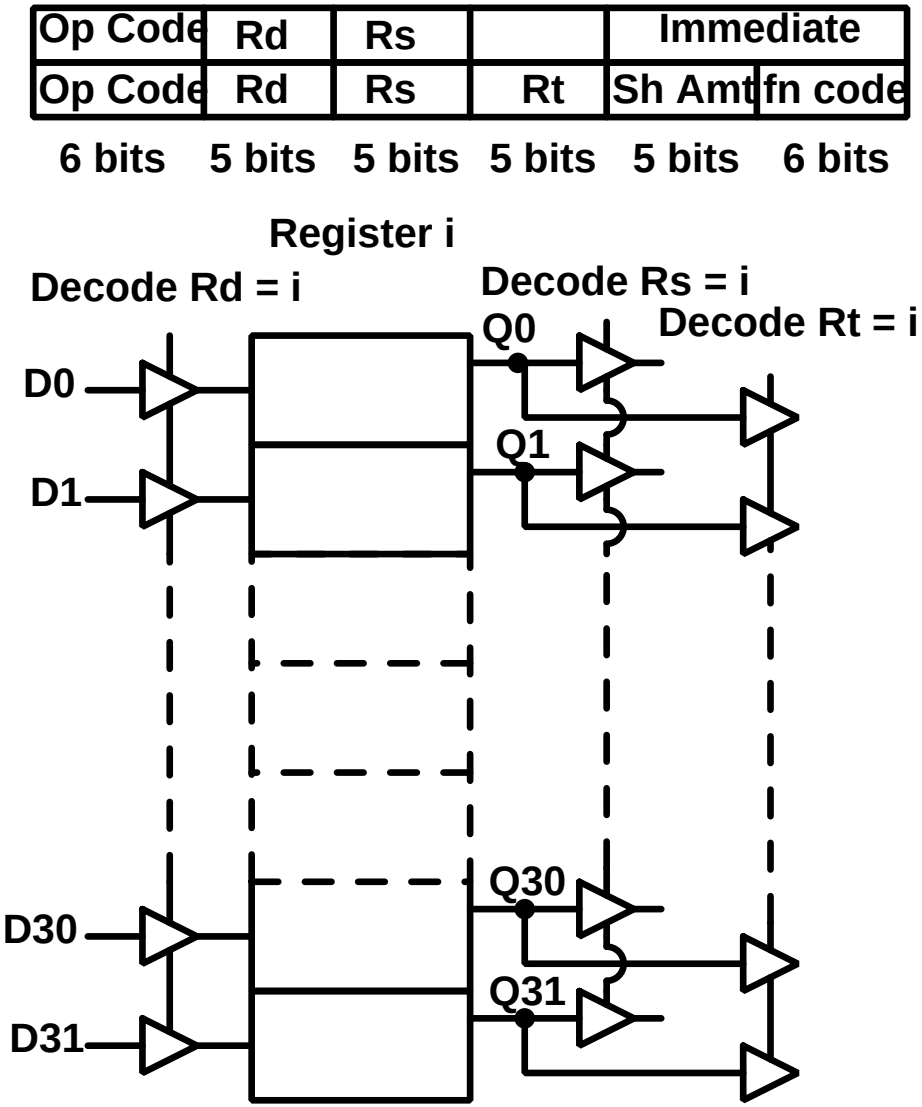
Gating the clock introduces skew in clock arrival time, which is not desirable.



- A data register can be formed by putting together multiple flipflops.
- The circuit shown here implements one of the 32 bit registers used by the MIPS processor.
- Notice that tristateable buffers are used here instead of inverters.
- The circuit provides two independent output ports and an input port.

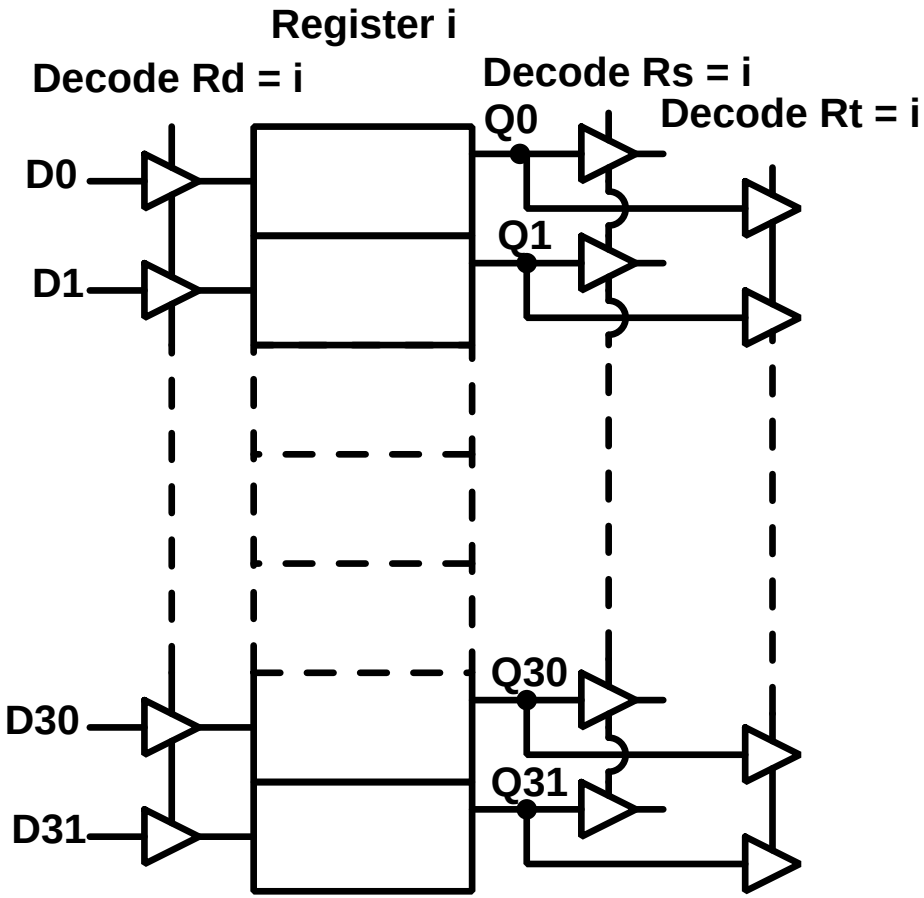


- Each Q output of the flipflops is connected to two tristateable buffers.
- One set of output buffers is enabled by decode output from the source register Rs specified in the instruction.
- The other set of output buffers is enabled by the decode output from the second source register specified in R type instructions.

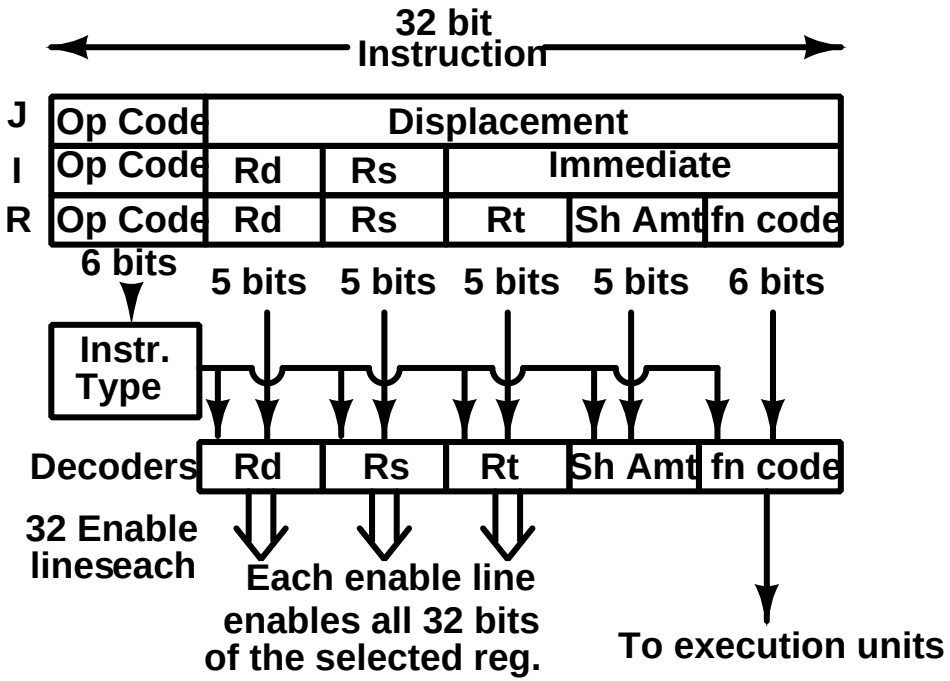


- When the 5 bits specified in Rs or Rt register fields of the instruction match the index of this register, the enable signal for the corresponding set of tristateable buffers goes high, enabling the outputs to appear at the corresponding port.
- Similarly, the input tristateable buffers are enabled when the Rd field of the instruction contains the 5 bit index of this register.

Op Code	Rd	Rs		Immediate	
Op Code	Rd	Rs	Rt	Sh Amt	fn code
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

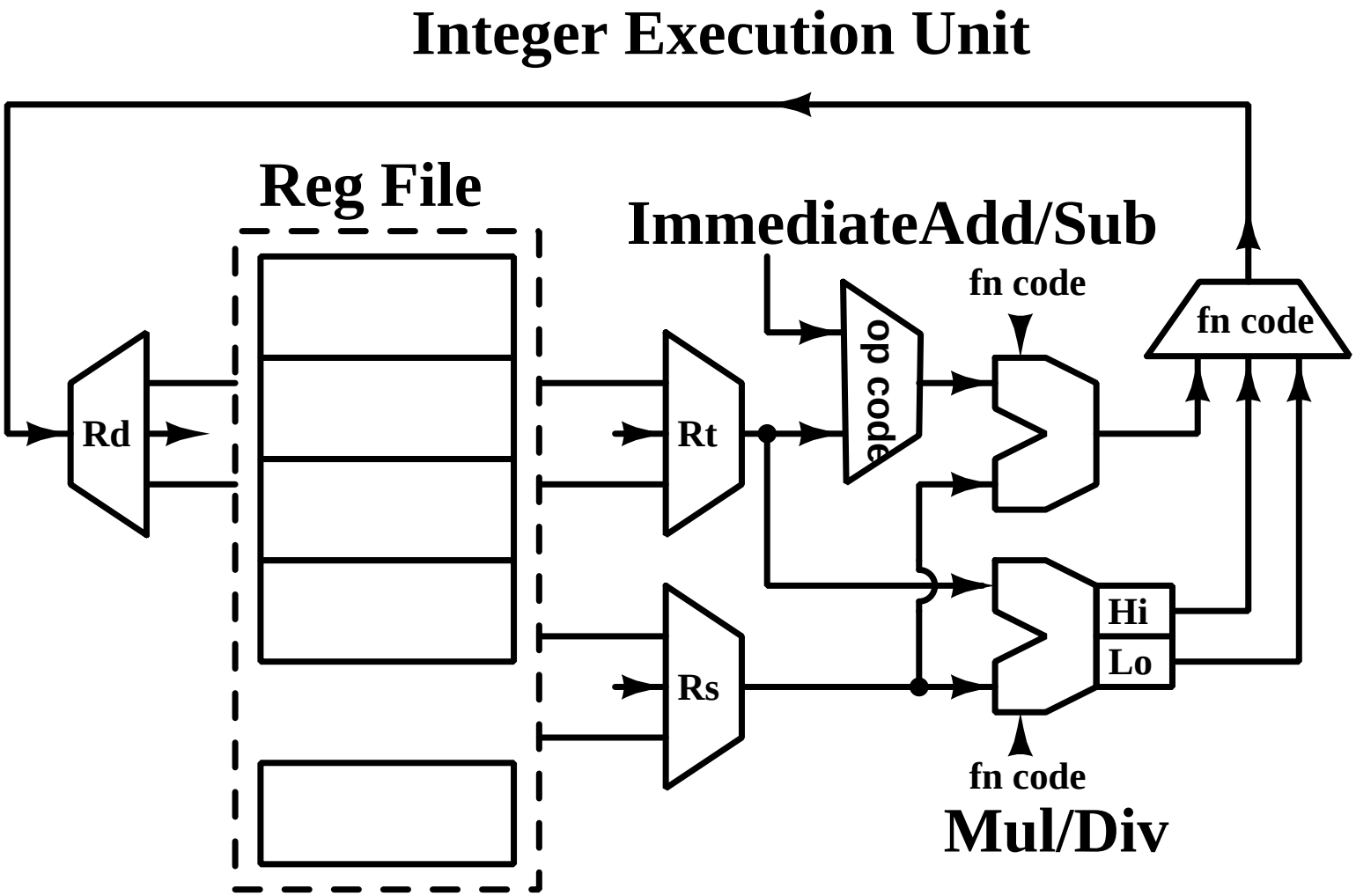


- Outputs of corresponding bits of all registers are shorted together.
- Buffers of only one selected register are enabled by the decoder, so the output of that port corresponds to the data stored in the selected register.
- Similarly, the D inputs of only the selected register are connected to the Rd port.



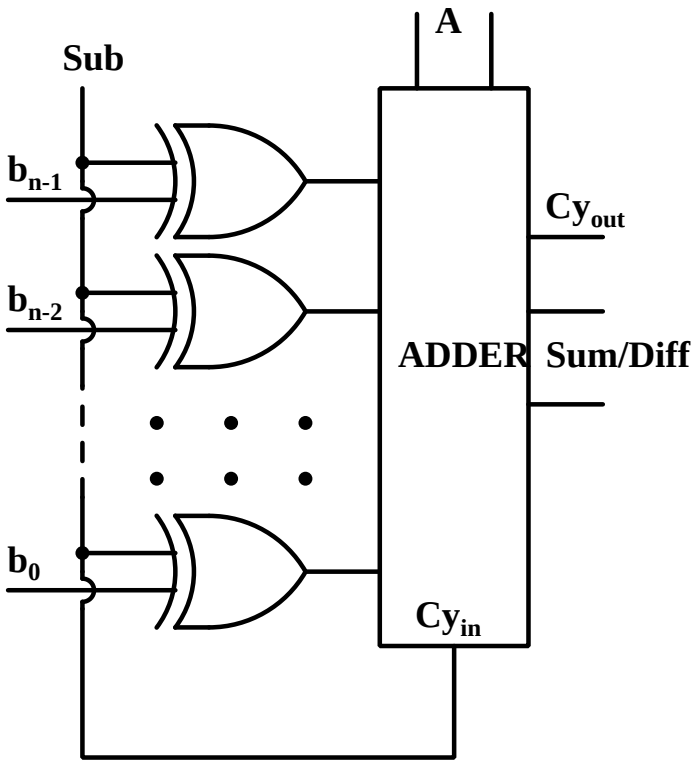
- The register decode circuit first decodes the top 6 bits of the instruction to determine the format of the instruction as R, I or J type.
- Decoders of a particular field are enabled only if that field exists in the instruction format.
- Each decoder output enables all 32 bits of the selected register.

The integer execution unit of MIPS is shown below:



Inside the ALU: Adder/Subtracter

You would have done several kinds of adders in your course on digital design. We can convert an adder to an adder/subtracter.



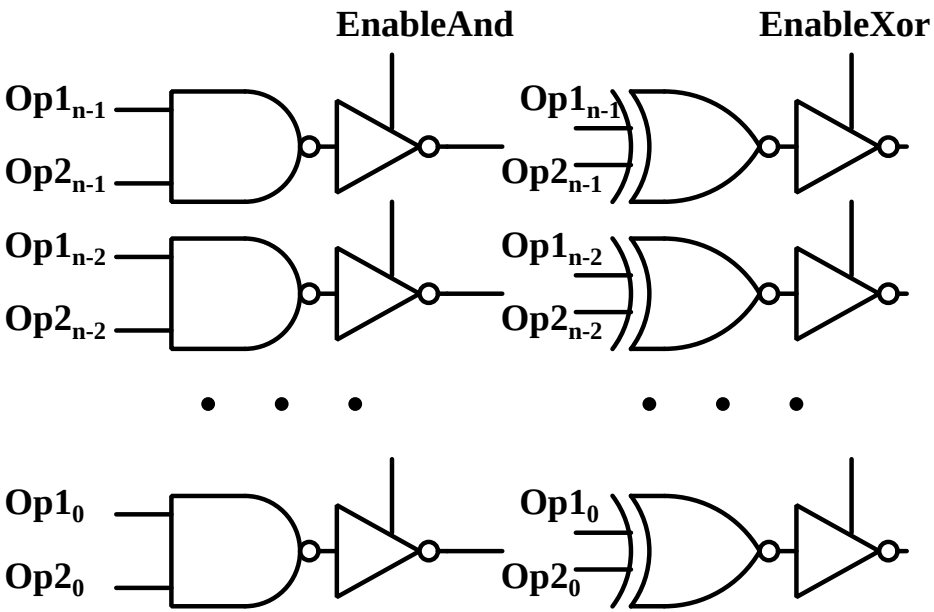
- If we wish to subtract rather than add, we add the 2's complement of the second operand to the first.
- We complement all bits of the second operand using XOR gates. Now we need to add 1 to it to convert it to its 2's complement.
- This can be done by setting the input carry to to the adder to 1.
- This circuit adds when sub=0 and subtracts when sub=1.

Inside the ALU: Incrementer

- The program counter is incremented by 4 at each instruction. A dedicated incrementer for this operation is worth while.
- An incrementer can be viewed as addition of zero with carry in of 1.
- Carry out at each bit, which is normally given by $ab + (a + b).C_{in}$ becomes $a.C_{in}$, as all b bits are zero. The sum is just $a \oplus C_{in}$.
- To increment by 4, we simply treat PC as a 30 bit register, keeping bits 1 and 0 always at 0. Incrementing this 30 bit register by 1 is equivalent to incrementing the full 32 bit value by 4.
- All techniques used for fast adders (such as carry look ahead or logarithmic addition) can also be applied to incrementers with much simpler logic.

Inside the ALU: Logic operations

Logic operations can be carried out easily using banks of logic gates.



- All outputs are buffered through tristateable inverters/buffers.
- Outputs for the same bit index for all logic functions are shorted together.
- The function code in the instruction selects which output will appear at the shorted outputs.

Barrel Shifters

- Shifters which produce outputs as select operations are called barrel shifters.
- The name comes from viewing the inputs as well as outputs as a circular arrangement of bits.
- The shifter then connects the input circle to the output circle like the sections of a barrel.
- A brute force implementation will require n multiplexers of n bits each, where the control inputs for each multiplexer are generated from the amount and type of shift/rotate.
- This is quite complex and puts a heavy load on data bits.

Logarithmic Barrel Shifters

- The brute force barrel shifter places a heavy load on input data lines because each input bit is a candidate for each output position.
- The control logic is complex because the amount of shift is variable.
- The loading on data lines and control logic complexity can be reduced if we break up the shift process into parts.
- We can carry out shifts in different stages, each stage corresponding to a single bit of the binary representation of the **shift amount**.
- Thus a shift by 6 (binary: 110) will be carried out by first doing a 4 bit shift and then a 2 bit shift.

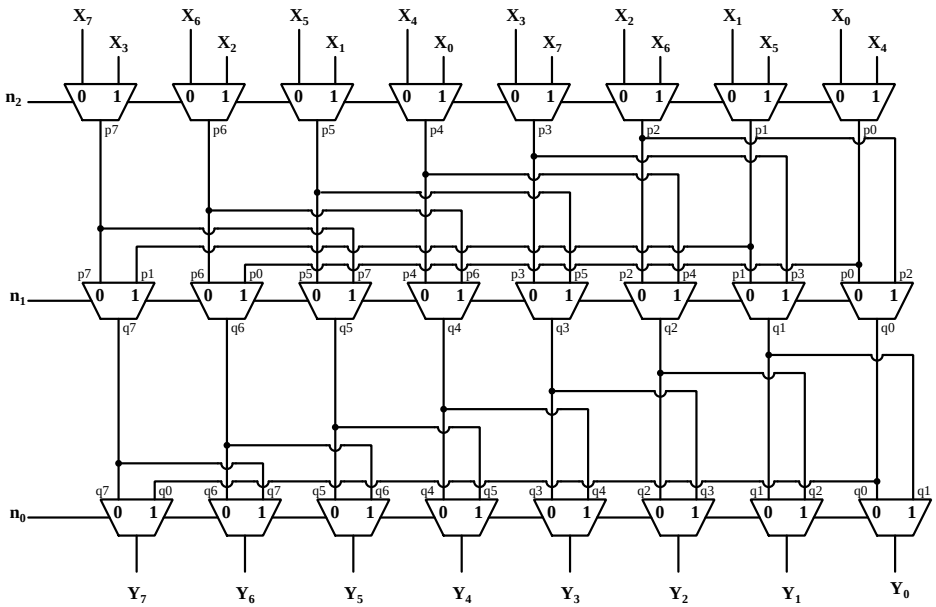
Logarithmic Barrel Shifters

- We need n bits to represent a maximum shift amount of $2^n - 1$ places.
- So the number of bits to express the shift amount (and hence the number of shift stages required) is logarithmic in the maximum shift desired.
- That is why such shifters are called Logarithmic Barrel Shifters.
- We can optionally buffer the outputs after each stage.

Logarithmic Barrel Shifter Stages

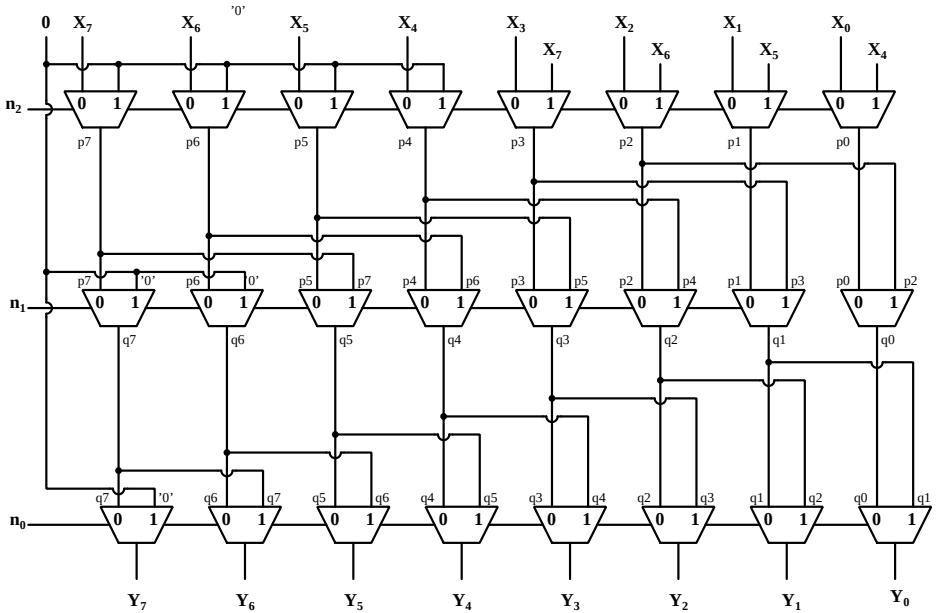
- Bit i of the **shift amount** represents
  - no shift (if it is 0)
  - a constant shift by 2^i places (if it is 1).
- If the shift amount is fixed, we do not need any electronics. The output can just be wired from the input bits.
- Using a 2 way mux controlled by bit i of **shift amount**, we can choose either the unshifted operand bit or the operand bit 2^i places away from it.
- This can be done for all bits of the operand in parallel.
- This constitutes one stage of the logarithmic shifter.
- The output can then be shifted again in the next stage, controlled by the next significant bit.

Right Rotate for an 8 bit Operand



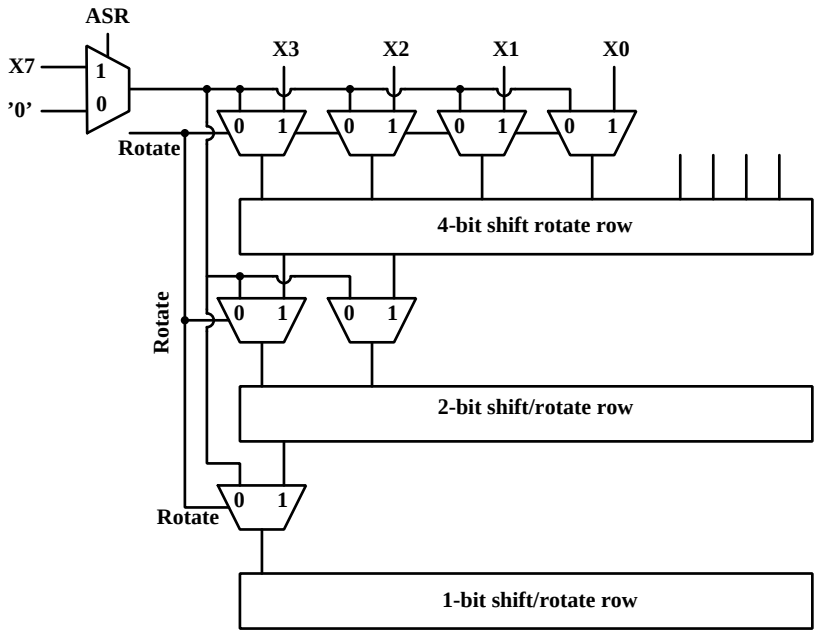
- Each input bit drives just two muxes, each with just 2 inputs.
- At each stage, the muxes select either the unshifted bit or a bit 2^n places from it.
- 3 stages are required for 0 to 7 bits of shift.

8 bit Logical Shift Right



- If we need a shift instead of a rotate, we feed a 0 instead of the corresponding bit.
- This has to be done for 4 muxes in the first stage, 2 in the second stage and 1 in the last stage.

Combining Rotate and Shift

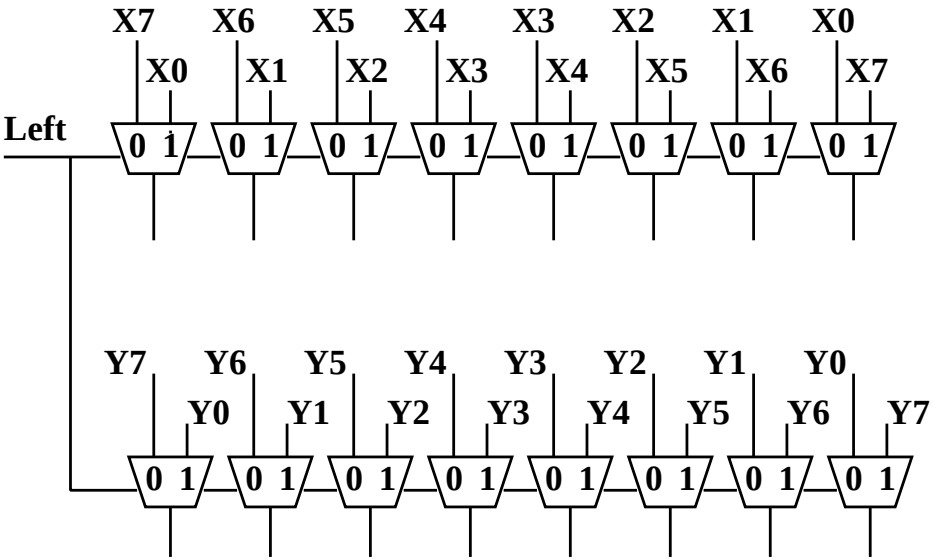


- We can combine the circuits for rotate and shift functions by putting muxes where different inputs need to be presented for the two functions.
- We can include the Arithmetic Shift function by choosing between 0 or X7 as the bit to be inserted.

Rotate and Shift by Masking

- We can also combine the rotate and shift functions by masking.
- We use the rotate function, which does not lose any information.
- Now we can mask n bits at the left to 0 if a right shift operation was desired instead.
- In case of an arithmetic shift, n bits on the left have to be set to the same value as X7.
- Shift/Rotate Left case is similar, except that the Logical and Arithmetic shifts are the same.

Combining Left and Right Shift/Rotate



- We can use the same hardware for left and right shift/rotate operations.
- This can be done by adding rows of muxes at the input and output which reverse the order of bits.

Combining Left and Right Shift/Rotate

- We can also make use of the fact that a left rotate by n places is the same as a right rotate by $2^n - n$ places.
- $2^n - n$ is just the 2's complement of n.
- By presenting the 2's complement of n at the mux controls, we can convert a right rotate to a left rotate.
- This can be followed by a mask operation, if a shift operation was required, rather than a rotate.