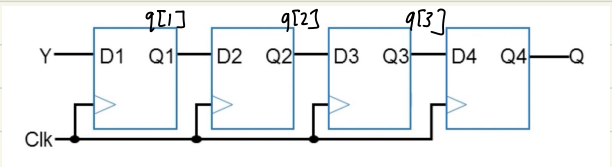
Shift Register

a shift register takes a single input and passes it through a chain of flip-flops. at each clock cycle, the input progresses one stage



internal signals are required to connect the registers and must be declared as reg type

```
module shiftreg (input clk, y,
                 output reg q);

  reg q1, q2, q3;

  always @ (posedge clk)
  begin
      q1 <= y;
      q2 <= q1;
      q3 <= q2;
      q  <= q3;
  end
endmodule
```

order doesn't matter

old values

another option, → next page

- using vectors ; shifting on command ; changing direction

```
module shiftreg (input clk, y, sh, rt,
                 output reg q_out);

reg [3:1] q;

always @ (posedge clk)
begin
  q[1] <= y;
  //q_out <= q[3];
  q[3:2] <= q[2:1];

assign q_out = rt? q[0] : q[4];

endmodule
```

can be assigned outside always block

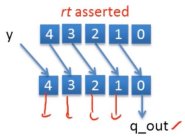
if (sh) begin ← if (rt) begin ← q[4] <= y; q[3:0] <= q[4:1];

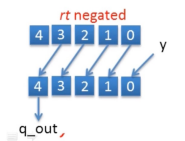
only 3 assignments required

```
reg [4:1] q;
assign q_out = q[4];

always @ (posedge clk)
begin
  q[4:1] <= {q[3:1], y};
end
```

- all outputs:

- We might want to be able to access the whole word






```
module shiftreg4 (input clk, y, sh, rt,
                  output q_out,
                  output [4:0] q_word);

reg [4:0] q;

always@(posedge clk)
begin
    if (sh) begin
        if (rt) begin
            q[4] <= y; q[3:0] <= q[4:1];
        end else begin
            q[0] <= y; q[4:1] <= q[3:0];
        end
    end
end

assign q_out = rt ? q[0] : q[4];
assign q_word = q;

endmodule
```

parallel in/out ⇒ [a:b] type I/O ← many
serial in/out ⇒ wire type I/O ← just one

not v. imp

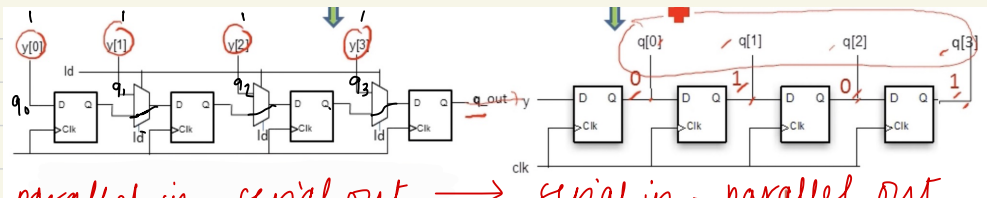
Serial Data Transfer

- one-bit at a type (sender)
  ⇓
  needs a parallel to serial converter (PISO)
  ↓
  serial to parallel converter (SIPO)
  ↓
  receiver

```
module piso4 ( input clk, ld,
               input [3:0] y,
               output q_out );
  reg [3:0] q;
  always @(posedge clk)
  begin
    if (ld) q <= y;
    else begin
      q[0] <= y[0];
      q[3:1] <= q[2:0];
    end
  end
  assign q_out = q[3];
endmodule
```

shift converter

```
module sipo4 (input clk, y,
              output reg [3:0] q);
always@(posedge clk)
begin
    q[0]   <= y;
    q[3:1] <= q[2:0];
end

endmodule
```

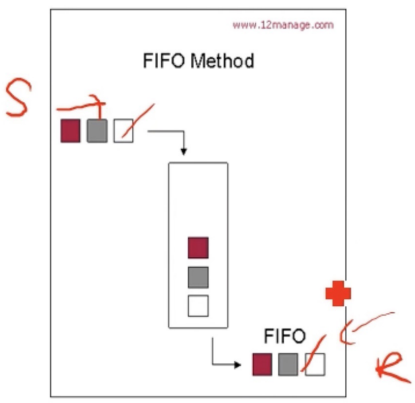


parallel in - serial out ⟶ serial in - parallel out

First-In-First-Out Buffer

- same as shift registers, except each register is multibit

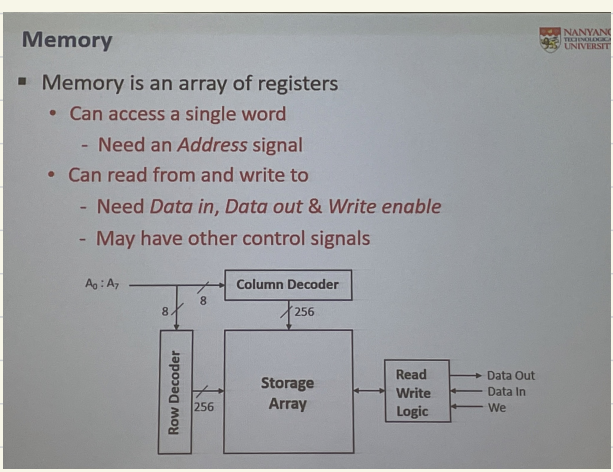
```
module fifo4 (input clk, input [3:0] y,
              output reg [3:0] q);

reg [3:0] q1, q2, q3;

always@(posedge clk)
begin
    q1 <= y;
    q2 <= q1;
    q3 <= q2;
    q  <= q3;
end

endmodule
```



Memories

- array of storage elements
- each register is a single storage element and has a unique address
- we should be able to select the register by its address and also read something out from the memories.

8:57

- can access a single word (needs an adress signal)
- 

defined as an array of type reg

8-bit x 1024 memory:

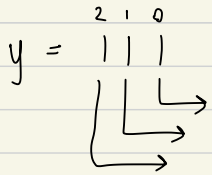
reg [7:0] mem_array [0:1023];

- A simple 8-bit x 1024 single port RAM module in Verilog

```
module dmem #(parameter WORD_SIZE = 8, ADD_SIZE = 10) (input clk, we
        input [WORD_SIZE-1:0] din, input [ADD_SIZE-1:0] add,
        output [WORD_SIZE-1:0] dout);
  reg [WORD_SIZE-1:0] RAM [0:(1<<ADD_SIZE)-1];   // Define mem arra
  assign dout = RAM[add];                        // Combinational read
  always @ (posedge clk)
    if (we) RAM[add] <= din;                     // Synchronous write
endmodule
```

→ shifts 1 by 10 Places ⇒ 10000000 = 2^{10} = 1024

memory initialisation



q <= y q = 1 1 1

q[