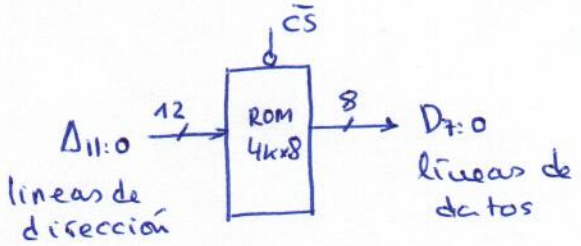
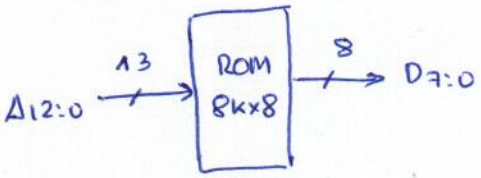
Apartado 1

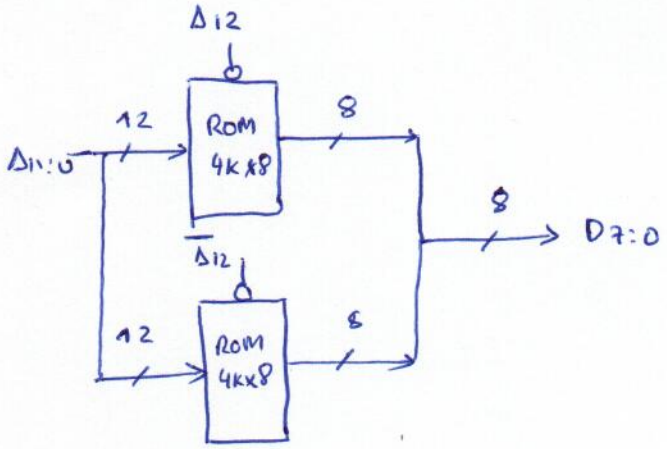
- memoria ROM 4Kx8 con $\overline{CS}$



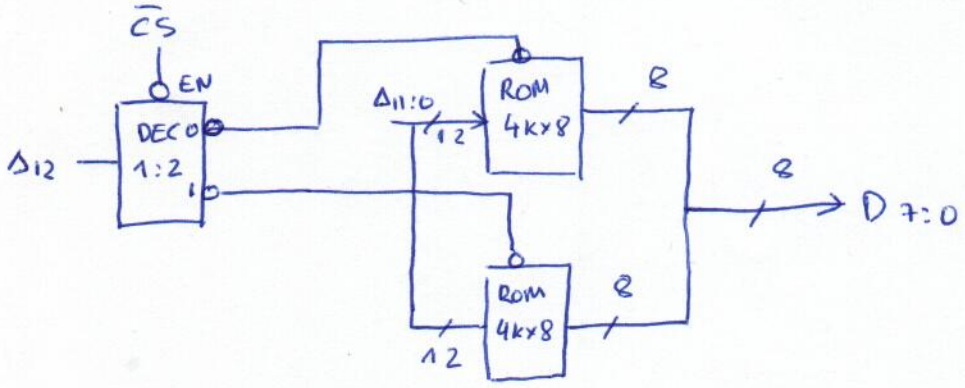

$4k = 2^2 \cdot 2^{10} = 2^{12}$

- obtener una ROM 8kx8 con dos ROM 4kx8




si queremos que la ROM 8kx8 también tenga $\overline{CS}$:



En las dos soluciones mostradas una de las ROM 4K x 8 cubre todas las direcciones en que $A_{12} = 0$ y la otra ROM 4K x 8 cubre todas las direccion en que $A_{12} = 1$.

La ROM 8K x 8 obtenida es una memoria de acceso aleatorio con 8192 palabras de 8 bits de anchura. Las entradas $A_{12:0}$ reciben el nombre de lineas de dirección y las salidas $D_{7:0}$ se denominan lineas de datos.

## Apartado 2:

Una descripción procedimental para el MUX2:1

```
module mux(
        input a,
        input b,
        input s,
        output reg z
        );

always @(a,b,s)      // también es válido @*//
  z = s ? b : a;

endmodule
```

Otra posibilidad:

```
module mux(
        input a,
        input b,
        input s,
        output reg z
        );

always @(a,b,s)
        if (s==0)
              z=a;
        else if (s==1)
              z=b;

endmodule
```

**Apartado 3:**

Una descripción procedimental para el registro de 8 bits:

```
module registro(
        input xr,
        input inicio,
        input shr,
        input cla,
        input ck,
        output zr
        );

reg [7:0] q;

always @(posedge ck,cla)
        if (cla)
                q<= 0;
                else if (inicio)
                        q<= 8'hd2;      // también podría ser q<=8'b11010010; //
                        else if (shr)
                                q <= {xr,q[7:1]};
assign zr = q;

endmodule
```

**Apartado 4:**

Descripción estructural del encriptador:

```
module encriptador(
                input serie_in,
                input circular,
                input reset,
                input inicio,
                input shr,
                input ck,
                input mensaje,
                output out
                );

// llamo cablemux al que va desde la salida del mux al registro y cablexor al otro //
wire cablemux,cablexor;

mux mux1 (.a(serie_in), .b(cablexor), .s(circular), .z(cablemux));
registro registro1 (.xr(cablemux), .cla(reset), .inicio(inicio), .shr(shr), .ck(ck),
                        .zr(cablexor));
xor xor1 (out,mensaje,cablexor);

endmodule
```

**Apartado 5:**

El testbench para encriptador:

```
module testcorto_tb;

// Inputs
	reg serie_in;
	reg circular;
	reg reset;
	reg inicio;
	reg shr;
	reg ck;
	reg mensaje;

// Outputs
	wire out;


encriptador uut (
	.serie_in(serie_in),
	.circular(circular),
	.reset(reset),
	.inicio(inicio),
	.shr(shr),
	.ck(ck),
	.mensaje(mensaje),
	.out(out)
	);

always begin
	#10;
	ck = ~ck;
end

initial begin
	serie_in = 0;
	circular = 0;
	reset = 0;
	inicio = 0;
	shr = 0;
	ck = 0;
	mensaje = 0;
```

```
        @(negedge ck);
        reset = 1;

        @(negedge ck);
        reset = 0;
        inicio = 1;

        @(negedge ck);
        inicio = 0;
        shr = 1;
        circular = 1;
        mensaje = 1;

        @(negedge ck);
        mensaje = 0;

        repeat (2) @(negedge ck);
        mensaje = 1;

        repeat (3) @(negedge ck);
        $finish;
end

endmodule
```