Exercises on the topics of class 25

Exercises with solutions

Ex. 1. Design a circuit that, given four source registers S_i and four destination registers D_i (for i = 1,…,4), allows the following parallel transfers:

(a) $S_1 \rightarrow D_1, D_2, D_4$ and $S_2 \rightarrow D_3$

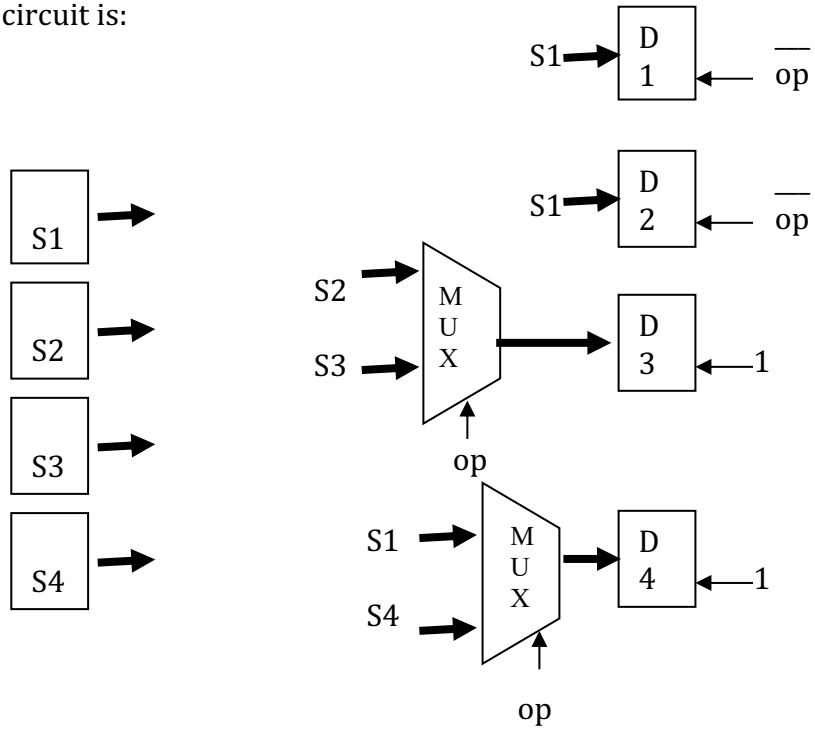
(b) $S_3 \rightarrow D_3$ and $S_4 \rightarrow D_4$

If we would identify D_1 with D_3 and D_2 with D_4 (that is, the new circuit only has destinations D_1 and D_2), would the above transfers be still legal? Why?

SOLUTION:

First of all, let's observe that D1 and D2 only receive from S1, whereas D3 and D4 receive from either S2 or S3, and from either S1 or S4, respectively. Hence, we can use a point-to-point interconnection among S1 and D1, and among S1 and D2; by contrast, D3 and D4 would require a MUX 2-to-1 to select their input. According to the operation requested (that is specified by a bit *op*), the control line will activate or not D1 and D2 for writing (op = 0 activates them, op = 1 does not; hence, *in_D1* and *in_D2* are the negation of *op*). Instead, registers D3 and D4 are always enabled in writing (*in_D3* and *in_D4* always hold 1); what is needed is a proper selection of the source register for them (that is, a control bit for the two MUXs): when *op* = 0, it must select S2 for D3 and S1 for D4; when *op* = 1, it must select S3 for D3 and S4 for D4. So, for both MUXs the control signal is the bit *op*.

To conclude, the circuit is:



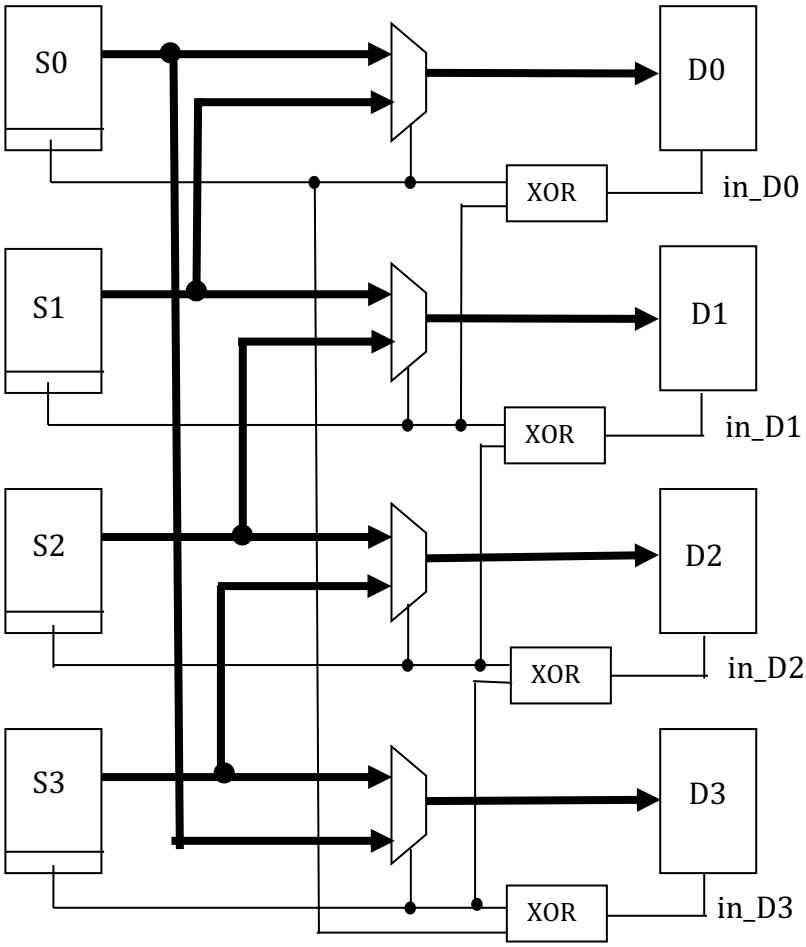
If we now change the net to have just D1 and D2 as destination (with D3 that become s D1 and D4 that becomes D2), the transfer (b) would still be legal, whereas transfer (a) no: indeed, in the first case no conflict arises since sources and destinations are disjoint, whereas in the second case there is a conflict on D1 that should simultaneously receive the datum coming from S1 and S2.

Ex. 2. Design an interconnection net with 4 source registers S0, S1, S2, S3 and 4 destination registers D0, D1, D2, D3 in which the following transfers can be done in parallel: if the content of Si is even then Si is copied into Di; otherwise $S_{(i+1) \bmod 4}$ is copied into Di. Moreover, Di is enabled to writing if and only if Si + $S_{(i+1) \bmod 4}$ is odd. How can we realize the same interconnection (in a cheaper way) if the transfers should not happen in parallel?

SOLUTION:

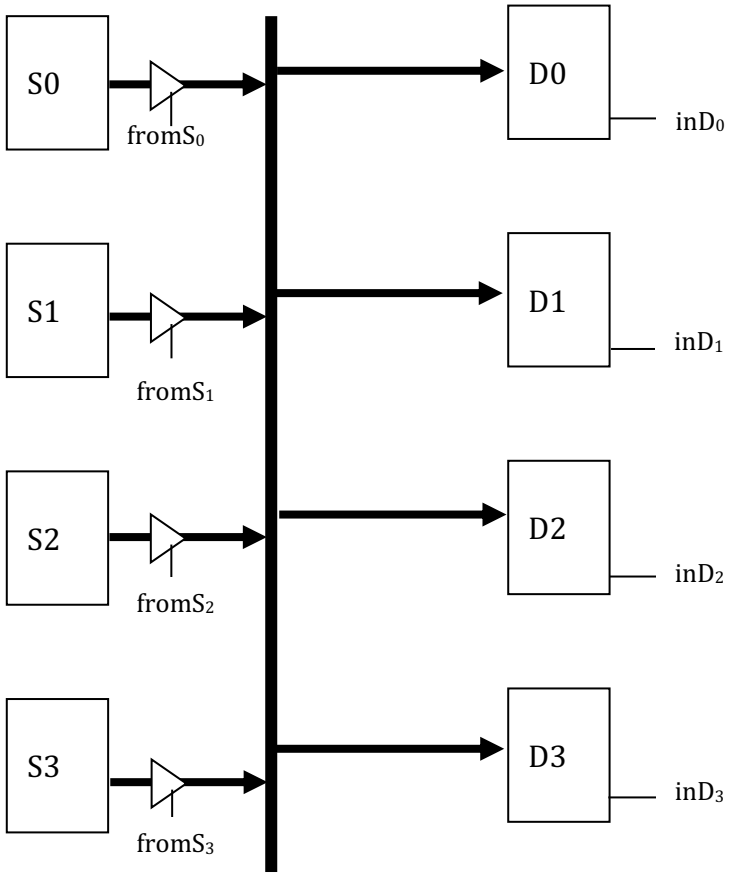
The interconnection net is a many-to-many one, with a MUX for every destination register that selects one among the two possible inputs Si and $S_{(i+1) \bmod 4}$ according to the parity of register Si; this requires just one single control signal. To check whether the content of a register is even or odd, we just need to check its less signifying bit: if 0, then the content of Si is even, otherwise its is odd. The value of the less signifying bit is then used as the control signal for the MUX. Signal *in_Di* is simply obtained by noting that the sum of two numbers is odd if and only if exactly one of the two summands is odd; hence, this can be done by performing the XOR among the less signifying bits of Si and $S_{(i+1) \bmod 4}$. The needed transfers are then:

D0 is enabled only if **S0+S1** is odd; it receives **S0** if S0 is even, **S1** if S0 is odd;
D1 is enabled only if **S1+S2** is odd; it receives **S1** if S1 is even, **S2** if S1 is odd;
D2 is enabled only if **S2+S3** is odd; it receives **S2** if S2 is even, **S3** if S2 is odd;
D3 is enabled only if **S3+S0** is odd; it receives **S3** if S3 is even, **S0** if S3 is odd.



Without parallel transfers, we can use a bus. However, we must ensure the exclusive access to the bus, so we must select which transfer has to be performed (via 2 input signals, call them

a1 and *a0*, with 00 enabling the first transfer, 01 the second, 10 the third, and 11 the fourth) or sequentialize the transfers (by using a counter modulo 4, that produces the bits *a1* and *a0*). In both cases, the interconnection is the following one:



where (we denote with *li* the LSB of Si)

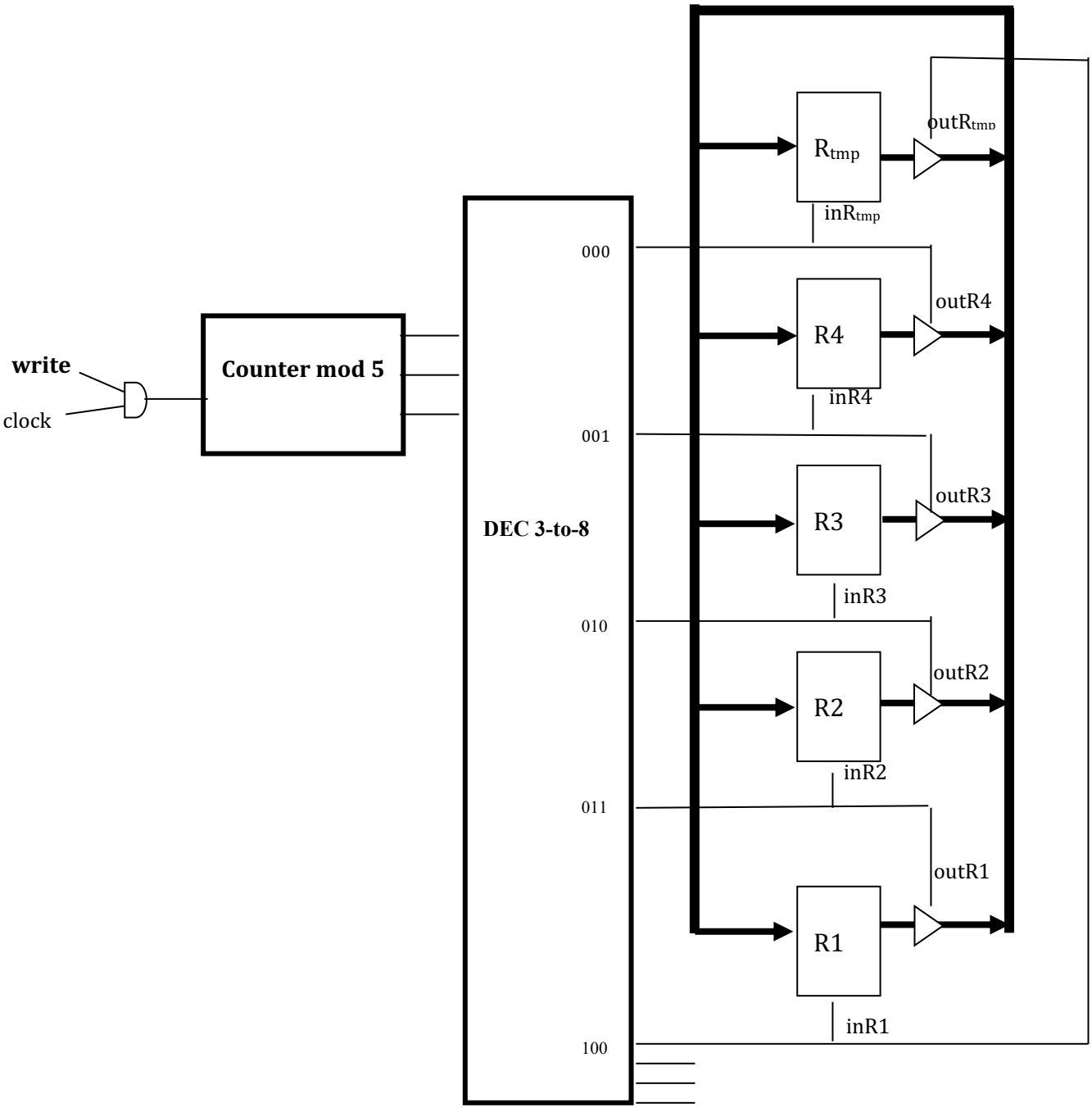
$$\text{inD0} = \overline{a1}\ \overline{a0}(l0 \oplus l1) \qquad \text{fromS0} = \overline{a1}\ \overline{a0}\ \overline{l0} + a1\ a0\ l3$$
$$\text{inD1} = \overline{a1}\ a0(l1 \oplus l2) \qquad \text{fromS1} = \overline{a1}\ a0\ \overline{l1} + \overline{a1}\ \overline{a0}\ l0$$
$$\text{inD2} = a1\ \overline{a0}(l2 \oplus l3) \qquad \text{fromS2} = a1\ \overline{a0}\ \overline{l2} + \overline{a1}\ a0\ l1$$
$$\text{inD3} = a1\ a0(l3 \oplus l0) \qquad \text{fromS3} = a1\ a0\ \overline{l3} + a1\ \overline{a0}\ l2$$

Ex. 3. Registers R1, R2, R3 and R4 are connected with a bus. If signal **write** is 1, the following transfers must be sequentially performed: R1 → R2, R2 → R3, R3 → R4 e R4 → R1. Tenendo conto che utilizzando un bus non è possibile eseguire trasferimenti in parallelo, progettare quanto necessario a produrre **tutti** i segnali di controllo e la loro temporizzazione fino al dettaglio di porte logiche e flip-flop (N.B. i registri non vanno dettagliati).

SOLUTION:

Since the bus doesn't allow parallel transfers, whenever **write**=1 we have to perform them one after the other, by logically performing R1 --> R2 , R2 --> R3 , R3 --> R4 and R4 --> R1 in

some sequence. In doing this, we must ensure that the previous contents of the registers are not lost (e.g., if we first move R1 into R2 and then R2 into R3, we have to somehow save the previous value of R2 to be moved into R3). To this latter aim, we use a temporary register Rtmp and perform the following sequence of transfers: R4 → Rtmp (so that the content of R4 is saved), R3 → R4, R2 → R3, R1 → R2 and Rtmp → R1. Then, we use a counter mod 5 to sequentially enable such five transfers, by using the first 5 outputs of a 3-to-8 decoder. Such outputs control the writing signals *inR* and the control *outR* of the tri-state buffers:



Ex. 4. Let's have two 16-bits source registers S0 and S1 that contain floating point numbers in the IEEE half-precision standard. We also have two 10-bits destination registers D0 and D1. Design the following interconnection:

- If the integer part of the number stored in S0 is even, then copy the mantissa of the number stored in Si into Di
- Otherwise copy the mantissa of Si into $D_{(i+1) \bmod 2}$

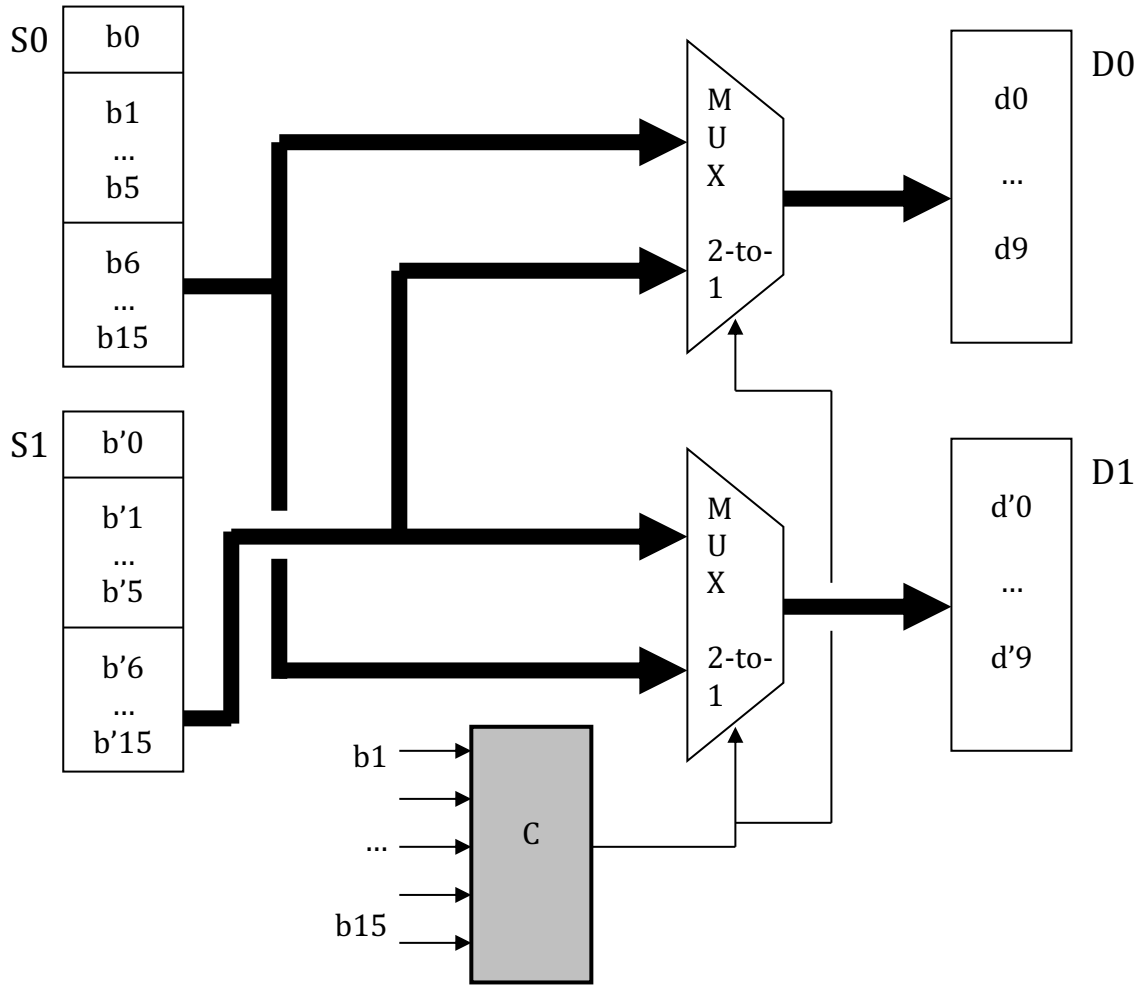
REMARK: it is not required that the mantissa of the number in S0 to be even, but it must be even the integer part of the number itself!

SOLUTION:

Source registers can be (logically) divided as follows:

b0	b1 ... b5	b6 ... b15

where b0 is the sign bit, b1/…/b5 give the (biased) exponent and b6/../b15 are the (normalized) mantissa. Since this is a many-to-many interconnection, its structure is the following:

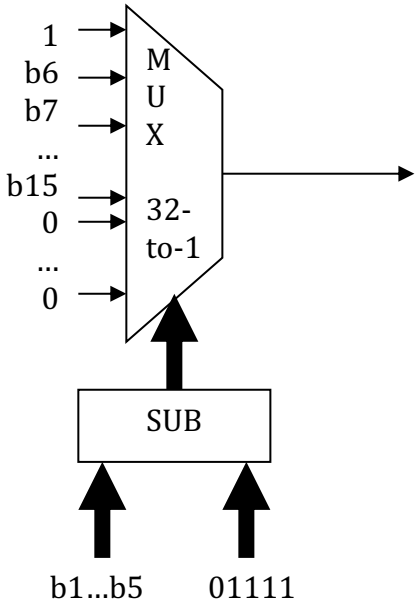


where the thick lines represent the 10 bits of the mantissas. As usual, the MUXs select the upper input if the control signal is 0, the lower input otherwise.

We are now left with the control circuit C. To design it, first recall that a number is even if and only if its LBS is 0. With this in mind, the easiest way to implement the behaviour of C is by considering b1…b5 – 01111 (as a number in 2-complement with 5 bits); if it is

- 00000, then the number is 1,b6…b15 and so the LSB of its integer part is 1;
- 00001, then the number is 1b6,b7…b15 and so the LSB of its integer part is b6;
- 00010, then the number is 1b6b7,b8…b15 and so the LSB of its integer part is b7;
- …
- 01001, then the number is 1b6…b15 and so the LSB of its integer part is b15;
- 01010, then the number is 1b6…b150 and so its LSB is 0;
- …
- 01111, then the number is 1b6…b15000000 and so its LSB is 0;
- 10000, then the number is 0,0…01b6…b15 and so its integer part is 0;
- …
- 11110, then the number is 0,01b6…b15 and so its integer part is 0;
- 11111, then the number is 0,1b6…b15 and so its integer part is 0.

So, we can use the result of the difference as the 5 control lines of a 32-to-1 MUX that provides the LSB of the integer part of the number stored in S0:



where here the thick lines represent numbers of 5 bits. This circuit gives in output 0 if and only if the integer part of the number stored in S0 is even, as desired.

Exercises without solutions

Ex. 1. Given 3 source registers R0, R1 and R2 and a destination register R' design the interconnection net controlled by the signals *inR'* and *op* that can perform the following trasfers:
if *inR'* = 0 , R' doesn't change;
if *inR'* = 1 and *op* = 0 , R0 is copied into R';
if *inR'* = 1 and *op* = 1 , in R' we store the bit-wise exclusive OR of the content of R1 and R2.

Ex. 2. Let's have 4 2-bits source registers $S_1, \dots, S_4$ and 6 3-bits destination registers $D_1, \dots, D_6$. The MSB of D_i (that we denote with d_i^3) is a relevance indicator of the info stored in the register; in particular, $d_i^3 = 0$ denotes a non-relevant info (that can hence be overwritten), whereas $d_i^3 = 1$ denotes a relevant info (that cannot be canceled).
Design the circuit that allows the following trasfers:

a) $S_1 \rightarrow D_3, \underline{D_4}$
b) $S_2 \rightarrow D_1, D_2$
c) $S_3 \rightarrow D_5$
d) $S_4 \rightarrow D_6$

where $S_i \rightarrow \underline{D_k}$ denotes that the transfer yields a relevant storing into D_k (that is, after the transfer, we must have $d_k^3 = 1$), always provided that d_k^3 was not already at 1 (in this case, the transfer $S_i \rightarrow \underline{D_k}$ cannot happen). Moreover, if the transfer from S_i to D_k (relevant or not) happens, after it we shall have $d_k^1 = s_i^1$ and $d_k^2 = s_i^2$ (i.e., the two LSBs of D_k store the info present in S_i). Finally assume that at the beginning the destination registers have all their bits at 0.

Ex. 3. Given th source registers A and B that contain values in the 2-complement representation, a destination register R, and two control signals c1c0, design the circuit such that:
- if c1c0=(0,0) copies into R the successor of B;
- if c1c0=(0,1) copies into R the maximum between (the values stored in) A and B;
- if c1c0=(1,0) copies into R the result of the sum between A and B;
- if c1c0=(1,1) copies into R the predecessor of A.