




SAPIENZA  
UNIVERSITÀ DI ROMA  
DIPARTIMENTO DI INFORMATICA

## Register Interconnection

Prof. Daniele Gorla



SAPIENZA  
UNIVERSITÀ DI ROMA  
DIPARTIMENTO DI INFORMATICA

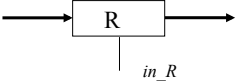
### Interconnecting registers

To be stored and handled, information must be moved from one register to another one within a computer.


This is done through **interconnection nets**.

REMARK: if registers have a parallel download (PIPO/SIPO), the moved information is actually *copied* from the source register to the destination one.

However, we don't care about uploading/downloading and assume that a register is simply




where thick lines denote  $n$  bits (if the register is made up by  $n$  FFs) and  $in\_R$  enables writing into the register (like the *load* line for the PIPO)



SAPIENZA  
UNIVERSITÀ DI ROMA  
DIPARTIMENTO DI INFORMATICA

### 4 interconnection kinds

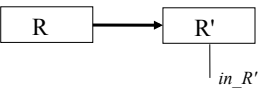
	Fixed destination	Variable destination
Fixed source	(point-2-point) through logic gates or buffer tri-state	With decoder
Variable source	With multiplexer	Through mesh or bus



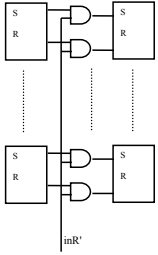
SAPIENZA  
UNIVERSITÀ DI ROMA  
DIPARTIMENTO DI INFORMATICA

### Fixed source and destination: logic gates

To move the content of R into R', it suffices to connect the output of R to the input of R' and set  $in\_R'$  whenever the info must be moved:



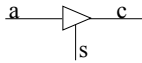
This is actually a schematic way to represent the following interconnection, by assuming that registers are sequences of SR FFs:



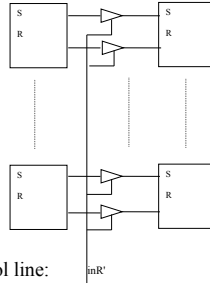
**Fixed Source and destination:  
tri-states Buffers**



A tri-state buffer is an electronic interruptor,  
Graphically designed as



that can assume three states (hence the name):  
 - open circuit:  $s = 0$ ;  
 - closed circuit and output 0:  $s=1 \ \& \ a=0$ ;  
 - closed circuit and output 1:  $s=1 \ \& \ a=1$ .



Instead of using AND gates, in the previous circuit  
we can use tri-states buffers with bit  $in\_R'$  as control line:

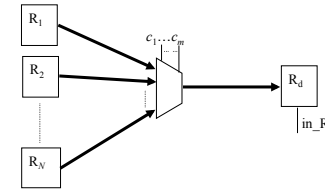
5

**Variable Source & Fixed destination:  
interconnection through multiplexers**



The source can be any of  $N$  registers  $R_i$ , whereas the destination register  $R_d$  is fixed.

This can be realized through a MUX whose data lines are the outputs of the  $N$  sources and whose output is given in input to the destination:



The MUX's control lines  $c_1, \dots, c_m$  are  $m = \lceil \log_2 N \rceil$  and provide the binary encoding of the index  $i$  of the register  $R_i$  whose content must be copied in  $R_d$ .

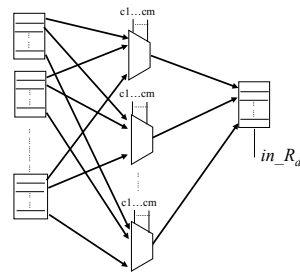
6

**Interconnection through MUX:  
details (single lines)**



In the previous representation, thick lines are actually  $n$  bits; hence, also the MUXs actually are  $n$ !!

- The **first** FF of every source register is connected with the **first** MUX, whose output goes to the **first** FF of  $R_d$ ;
- The **second** FF of every source register is connected with the **second** MUX, whose output goes to the **second** FF of  $R_d$ ;
- ...



The control lines are the same for all MUXs.

7

**Fixed Source & variable destination:  
interconnection through a DEC**

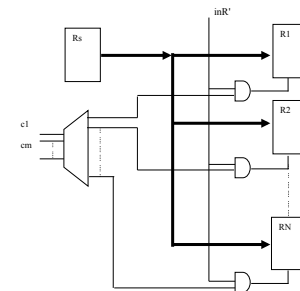


The source  $R_s$  is fixed, whereas the destination can be any  $R_i$  of a set of  $N$  registers.

This can be done by sending in input of every destination the output of the source and by using a decoder to enable writing in the chosen destination:

Inputs of the decoder  $c_1, \dots, c_m$  are  $m = \lceil \log_2 N \rceil$  and provide the binary encoding of the index  $i$  of the register  $R_i$  where the info in  $R_s$  must be moved.

The outputs of the DEC are put in AND with a "global" writing signal ( $in\_R'$ ) and provide signals  $in\_R_i$  for the destinations.



8

**Common mistake!**

It could be tempting to realize a one-to-many interconnection with a DEMUX, i.e. something like this:

This does NOT work well; indeed, the DEMUX, on the non-selected lines, outputs a 0 that, if not properly controlled, would put 0 in all non-selected destinations.

- controlling the writing on the destinations requires also in this case a DEC, for properly setting lines  $in_{R_i}$
- Hence, the DEMUX would be useless!!

**Many-to-many Interconnection through mesh**

Interconnection between  $M$  source registers and  $N$  destination ones.

To implement the net, we need  $N$  (groups of  $n$ ) multiplexers, one for every destination register:

REMARK: Every MUX has its own control lines (that are  $m = \lceil \log_2 M \rceil$ ), to enable moving info from every source to every destination

**Esempio di trasferimento in una mesh**

Vogliamo trasferire  $Rs_i$  in  $Rd_{(i+1) \text{ MOD } 3}$ .

$Rs_0 \rightarrow Rd_1$ :  
 $c_1^1 c_2^1 = 00, in_{Rd_1} = 1$

$Rs_1 \rightarrow Rd_2$ :  
 $c_1^2 c_2^2 = 01, in_{Rd_2} = 1$

$Rs_2 \rightarrow Rd_0$ :  
 $c_1^0 c_2^0 = 10, in_{Rd_0} = 1$

**Non-distinct sources and destinations**

**Mesh: advantages and disadvantages**

Pro: parallel transfers

Con: costs

Ex.: 128 registers (few!!!) with 32 bits

- $128 \times 32 = 4096$  MUX 128-to-1
- every MUX 128-to-1 is made up by a 7-to-128 DEC, 128 AND gates and 127 OR gates
- A 7-to-128 DEC is made up by 7 NOT gates and 128 AND gates

TOTAL:  $4096 \times ((7+128)+128+127)$   
 $\approx 1,6 \text{ millions gates!!!!}$

**A cheaper mesh**

Con: reduced parallelism (only transfers with the same source)

Pro: much cheaper!

Ex.: 128 register with 32 bits

- 32 MUX 128-to-1
- every MUX 128-to-1 is made up by a 7-to-128 DEC (7 NOT and 128 AND), 128 AND gates and 127 OR gates

TOTAL:  $32 \times ((7+128)+128+127)$   
 $\approx 12,5 \text{ thousands gates}$

**Many-to-many Interconnection through bus**

If we reduce parallelism, we can realize the previous interconnection in an even cheaper way through a *bus*, i.e. a stream of  $n$  lines that interconnect all registers, both in input and in output.

REMARK: to avoid conflicts in the bus access, every output is controlled by a (set of  $n$ ) tri-states buffers, each controlled by signal  $from_{R_i}$  that holds 1 if the source is  $R_i$ . Of course, we must ensure that at most one of such signals holds 1 at every time.

Ex. (128 register with 32 bits):  $128 \times 32 \approx 4100$  tri-states buffers


**Bus vs mesh**

2 kinds of memory:

- *central* (in the microprocessor): a few very quick registers
- *mass*: millions of register made up with cheaper technology

2 kinds of interconnection:

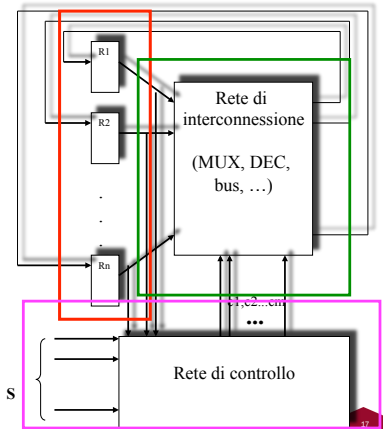
- mesh between the microprocessor registers
- bus to move data from mass memory to the central one


**SAPIENZA**  
 UNIVERSITÀ DI ROMA  
 DIPARTIMENTO DI INFORMATICA

### Design an interconnection net

We have 3 parts:

- The involved **registers** ( $R_1..R_N$ )
- The **interconnection net** that, in the general case, allows to move the content of every register into every other register
- The **control combinatorial net** that checks some internal condition (content of registers) or external one (signals  $S$ ) and generates the commands for the circuits that regulate the interconnection



The diagram illustrates the components of an interconnection net. On the left, a vertical stack of registers is labeled  $R_1, R_2, \dots, R_n$ . These registers are connected to a central block labeled 'Rete di interconnessione (MUX, DEC, bus, ...)'. Below this central block is another block labeled 'Rete di controllo'. The control net receives external signals  $S$  and sends control signals to the interconnection net. The interconnection net routes data from the registers.