Lecture 12:

Registers, Multiplexers, Decoders, Comparators and What Nots

After Karnaugh maps - what next?

Since we already know how to design both combinational and sequential circuits, what else is there to learn about digital circuits?

Actually, most practical digital circuit design problems are too large to be solved by minimisation and Karnaugh maps. They have to be solved by:

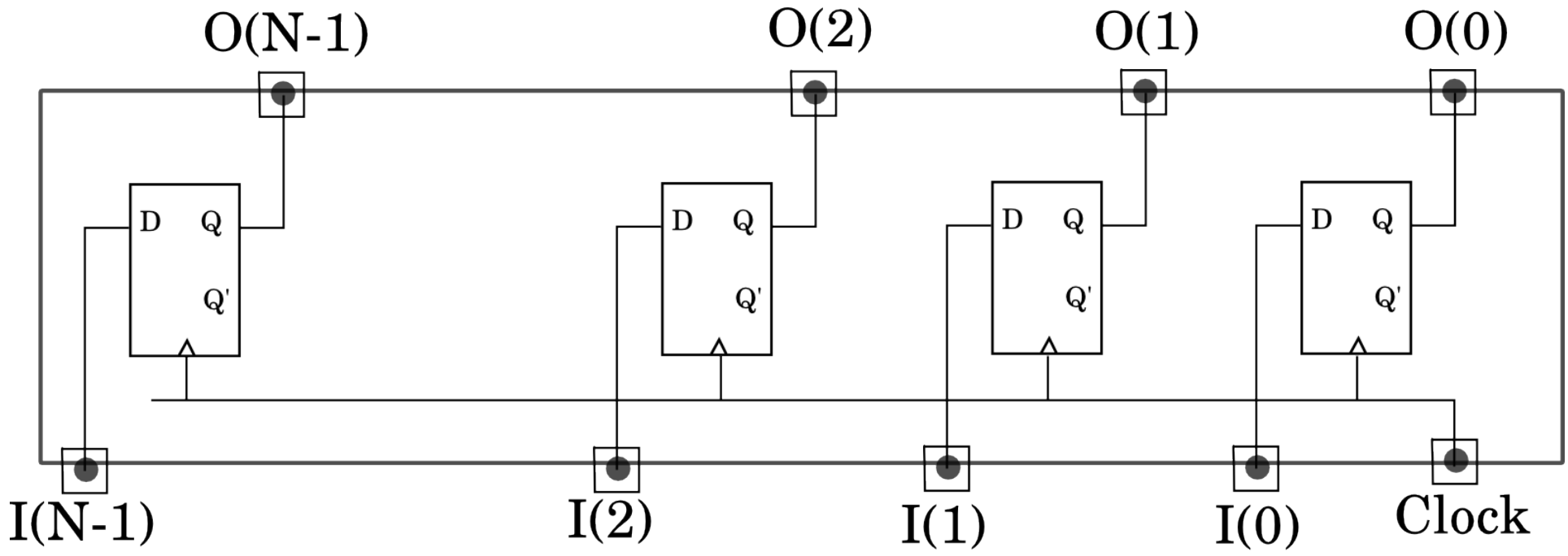
DIGITAL BUILDING BLOCKS

using

FUNCTIONAL DESIGN

Registers

The most fundamental digital building block of a digital computer is the register, which is an ordered group of single bit flip-flops with one clock signal connected to all of them.

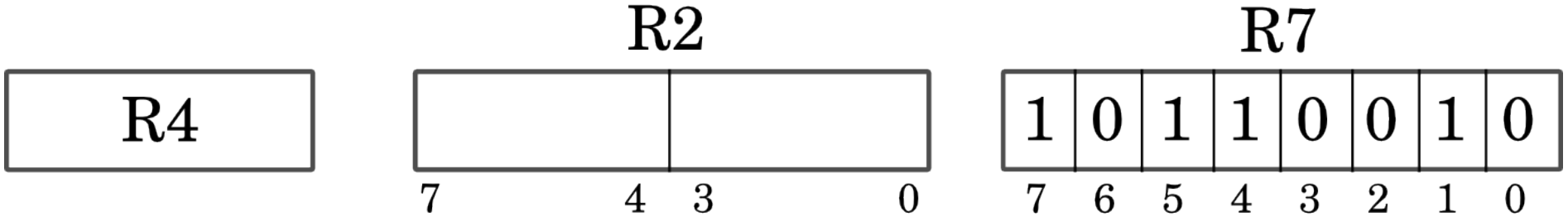


Registers (continued)

By convention we number the bits from 0 to N-1 for a N-bit register and may assign its content the positive numerical value Σ (2^n)*OB(n).

(However, it is important to remember that bits can be interpreted in any way you wish).

We also have short hand notations for registers. Three of the most common are:

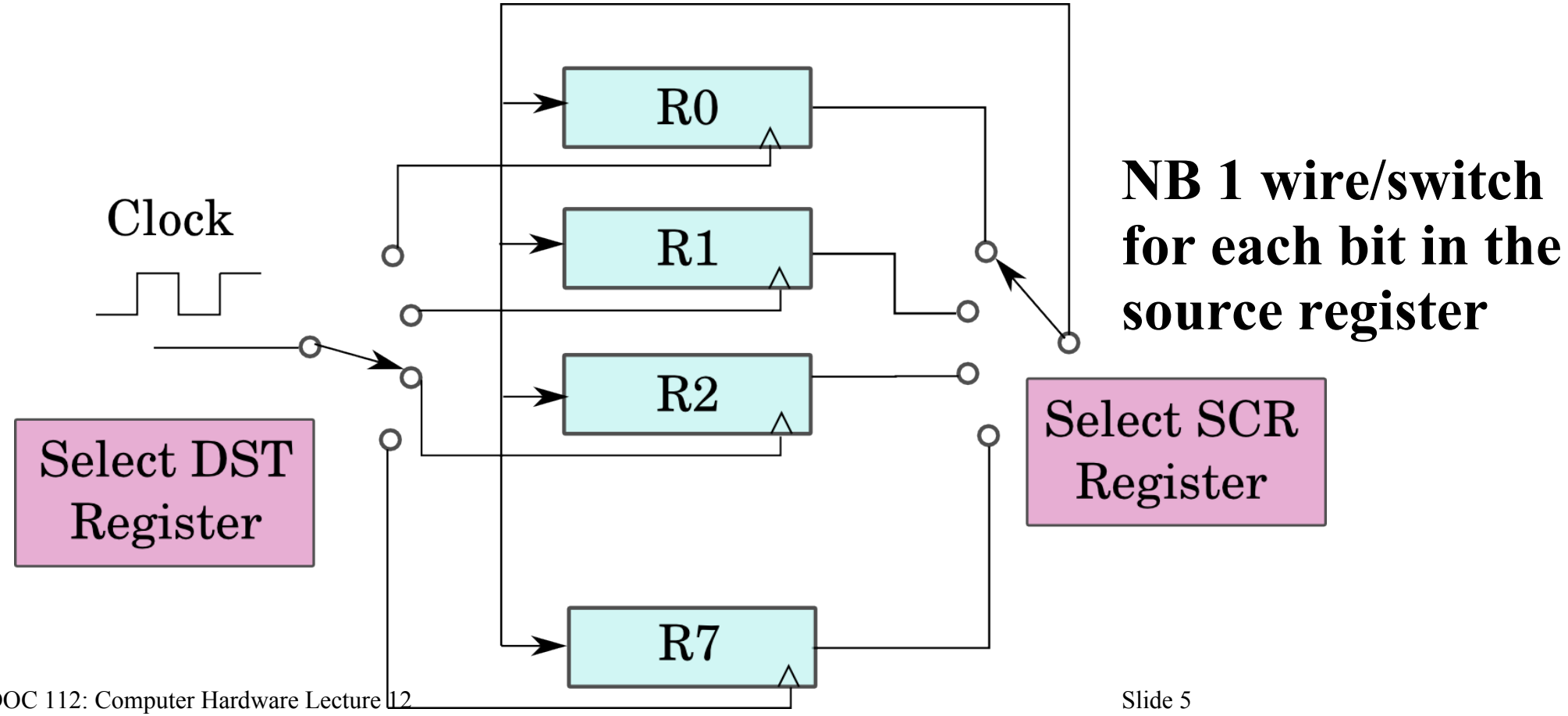


Registers can contain either data or control bits.

Register Transfer Operations

Simple Register Transfer:

the contents of one register (source) is copied into another (destination) without affecting the contents of the first.

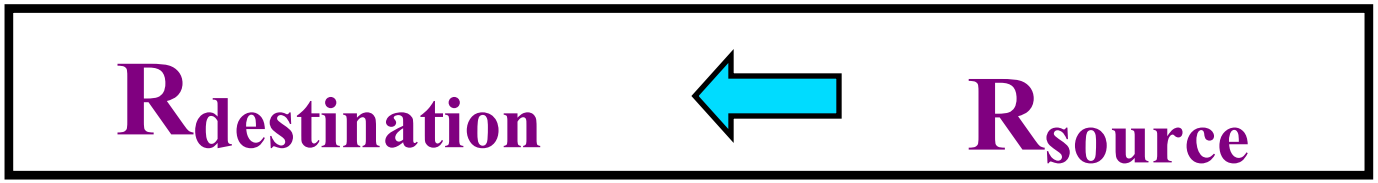


Operation of the Register Transfer Circuit

Select the Input register (Source)

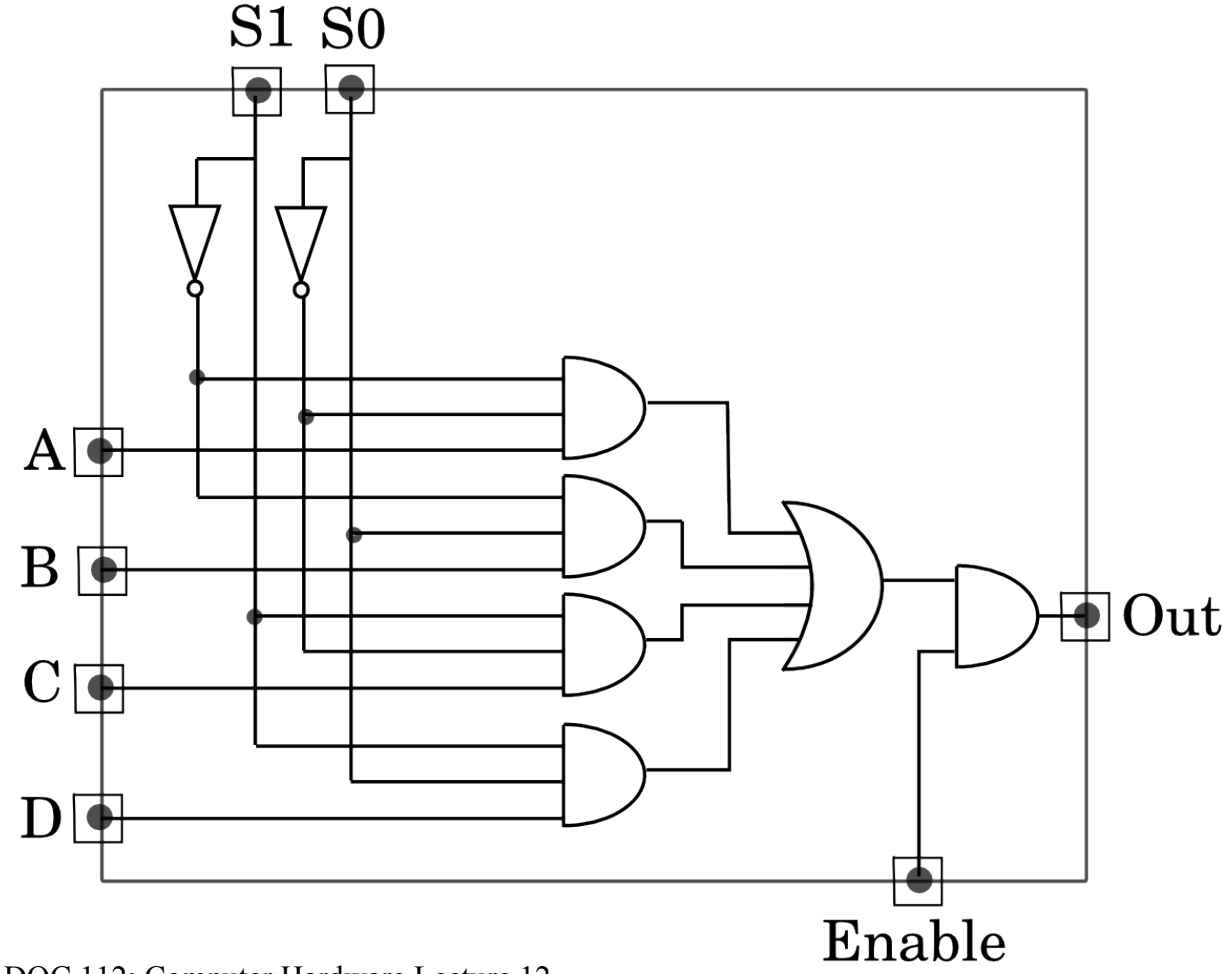
Select the Output Register (Destination)

Transfer data from the Source Register to the Destination Register



Select the Source Register

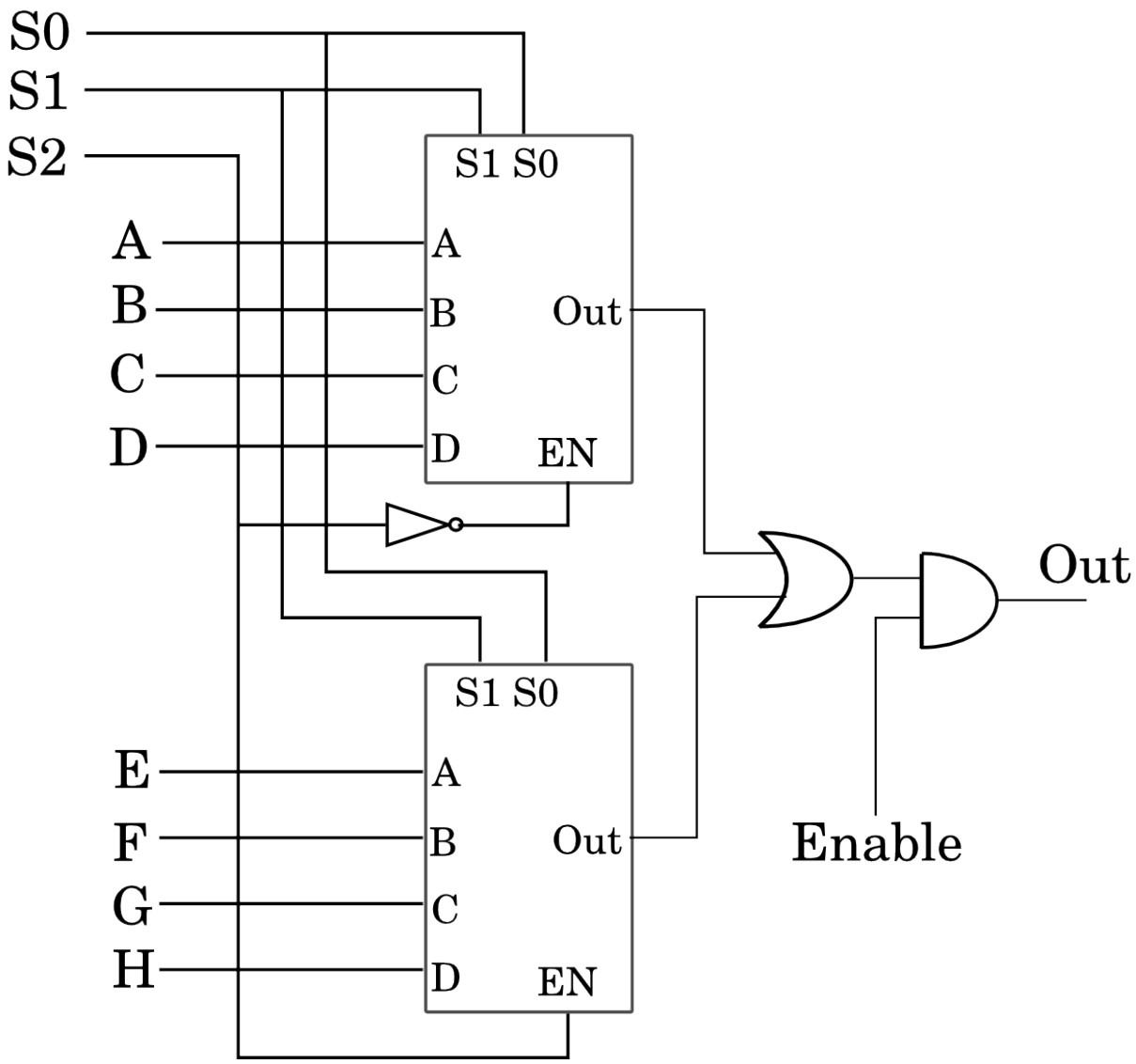
The source register may be selected by a Multiplexer circuit. (One multiplexer per bit)



A 4-to-1 Multiplexer:

ENBL	SEL0	SEL1	OUT
0	X	X	0
1	0	0	A
1	0	1	B
1	1	0	C
1	1	1	D

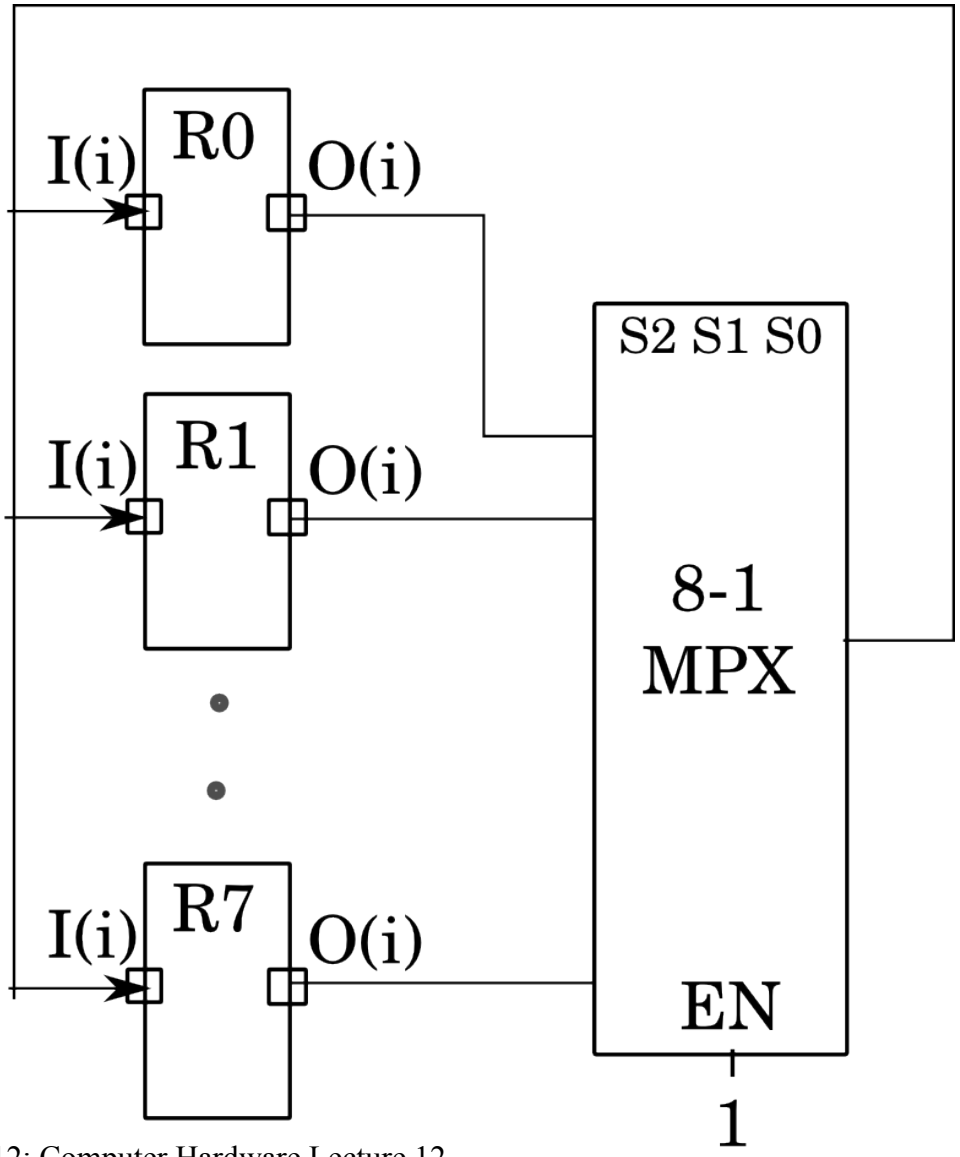
8-to-1 Multiplexer



Since we have 8 registers we need an 8-1 multiplexer.

We could design this at the gate level, but functional design is much simpler

Register Transfer using an 8-1 Multiplexer



We can now wire up each bit of the register transfer circuit.

The circuit for bit i is shown.

Another Possible Use of the Multiplexer

If you could use a 4-to-1 multiplexer, your assessed course work would be trivial!

Say, you needed AeorB for C1C2=00, B for C1C2=01, A'•B for C1C2=10, A'+B for C1C2=01.

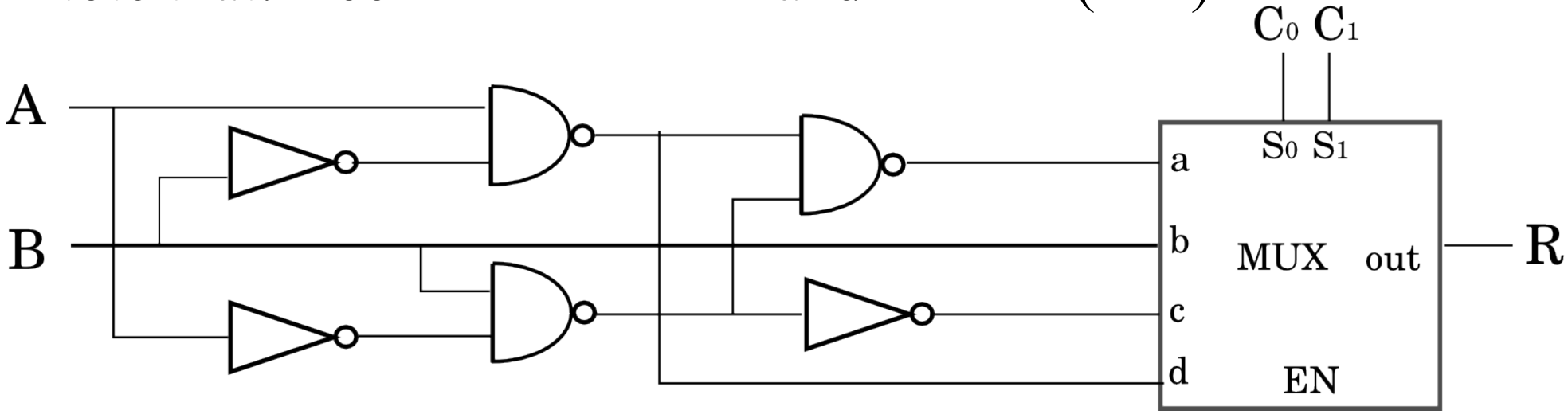
Note that: AeorB = AB'+A'B and A'+B = (AB')'

Another Possible Use of the Multiplexer

If you could use a 4-to-1 multiplexer, your assessed course work would be trivial!

Say, you needed AeorB for C1C2=00, B for C1C2=01, A'•B for C1C2=10, A'+B for C1C2=01.

Note that: AeorB = AB'+A'B and A'+B = (AB')'

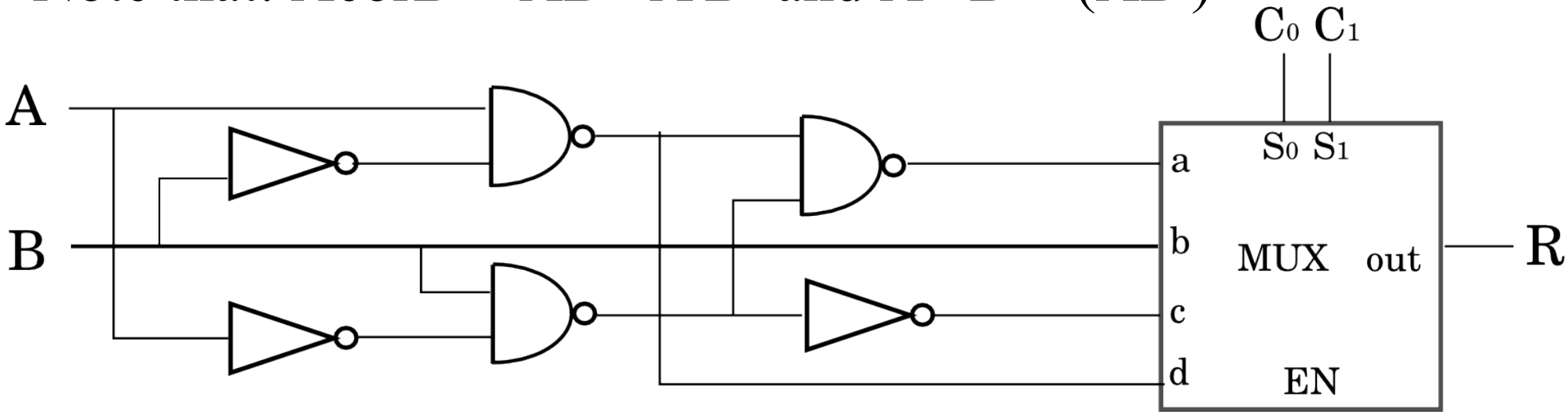


Another Possible Use of the Multiplexer

If you could use a 4-to-1 multiplexer, your assessed course work would be trivial!

Say, you needed AeorB for C1C2=00, B for C1C2=01, A'•B for C1C2=10, A'+B for C1C2=01.

Note that: AeorB = AB'+A'B and A'+B = (AB')'



This saves not only design time but £s as well!

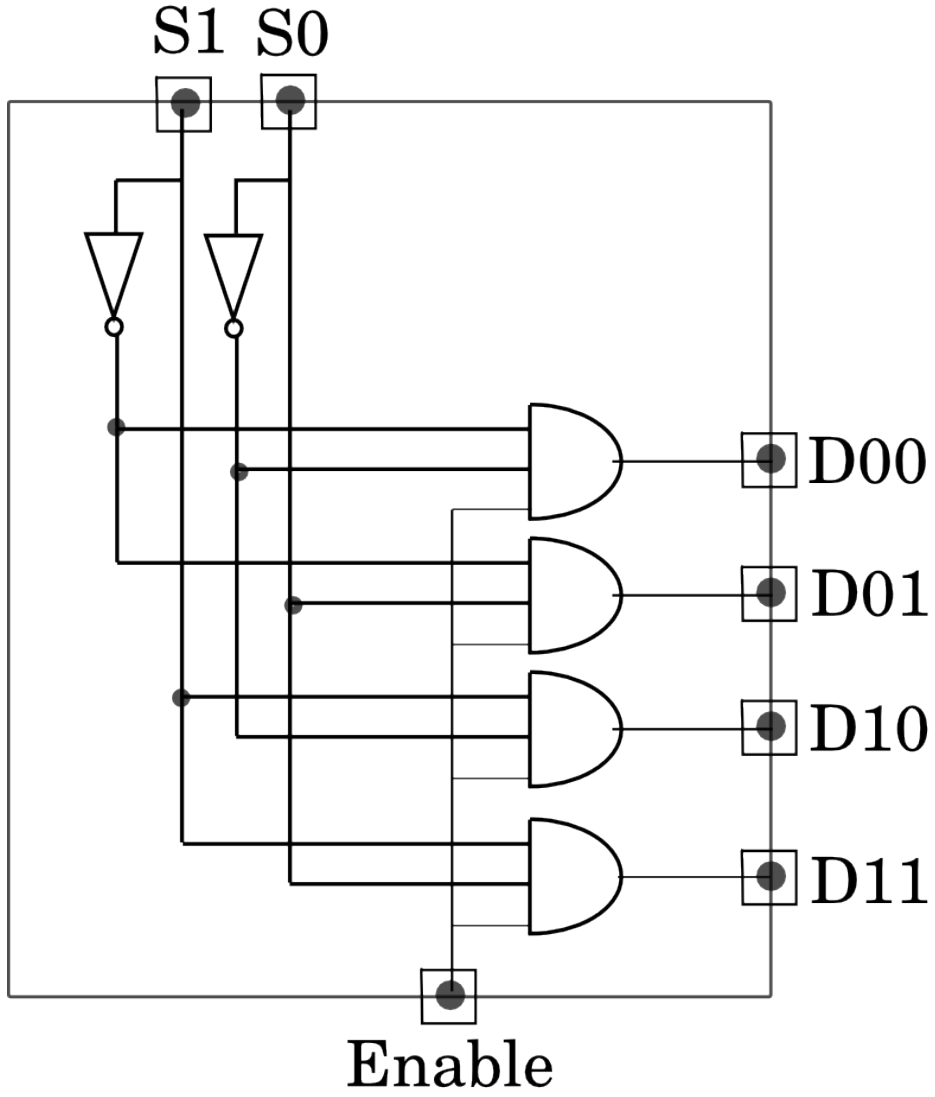
Output Control for The Register Transfer Circuit

So far, by using a multiplexer, we have been able to select the source register whose contents are fed back to the inputs of all the registers.

In order to complete the transfer, a clock pulse must be applied to the destination register. No clock pulse should be applied to any other register.

A Demultiplexer can be used to switch a clock pulse on one out of eight lines. This device has three selection inputs, one enable input and eight outputs.

The 2-to-4 Demultiplexer:



The demultiplexer is also called a decoder

It is a binary to unary converter

Operation of the 2-to-4 Demultiplexer

A good way to describe the operation of a decoder or demultiplexer is to look at its functional truth table:

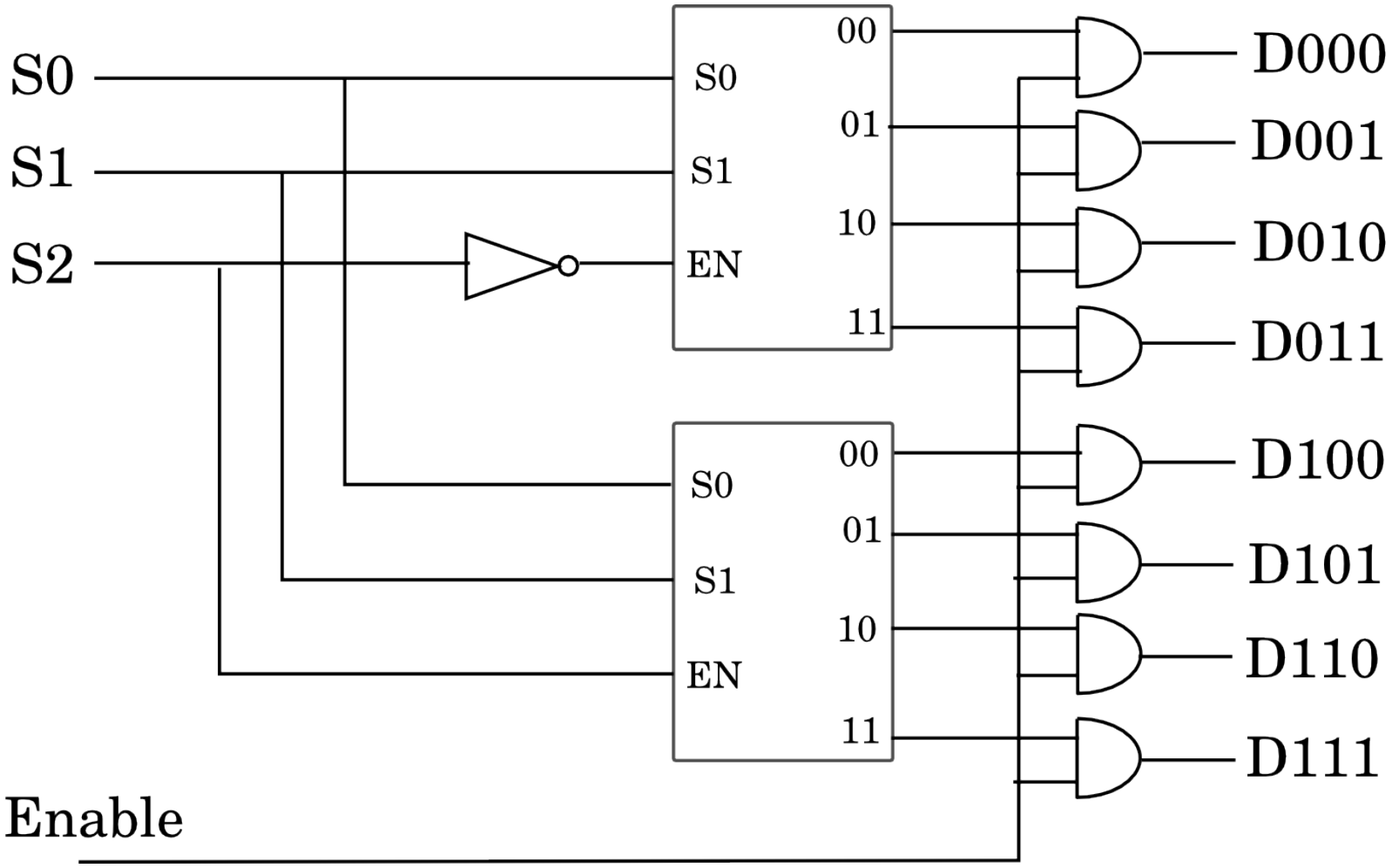
ENBL	SEL1	SEL2	D00	D01	D10	D11
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

When the enable input is: 0 all outputs are 0.

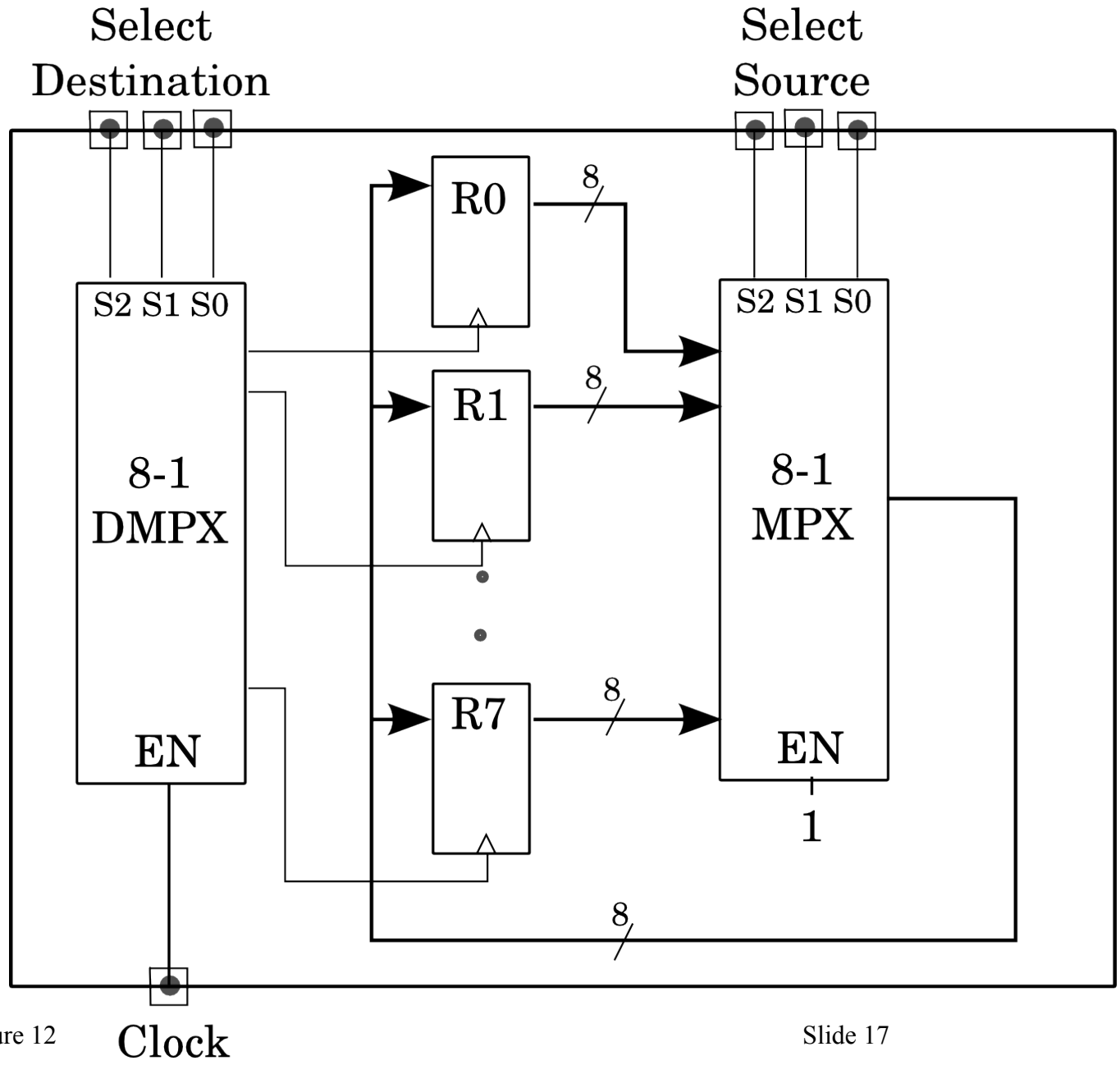
1 one of the outputs is 1.

The selection of the output is therefore determined by the values on the selection lines.

Expansion to a 3-to-8 Demultiplexer



The Final Circuit

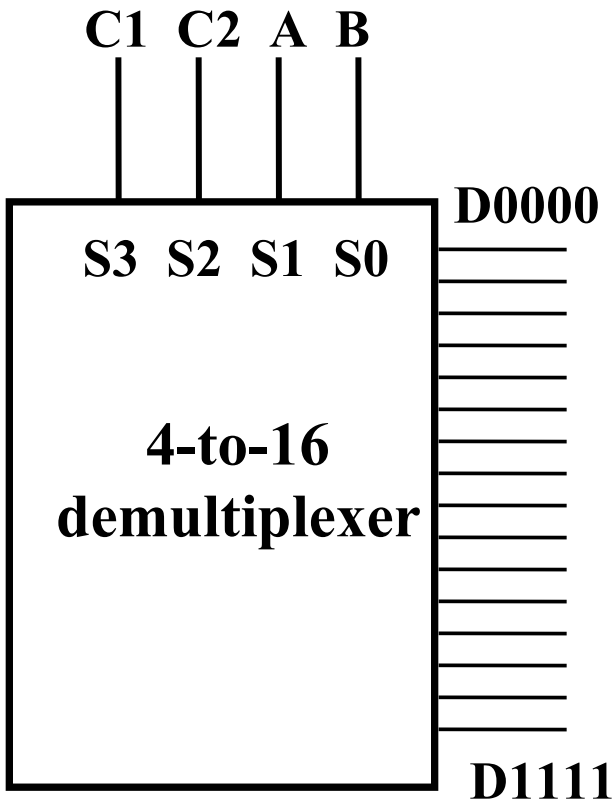


How to do your coursework in 30 seconds

Decoders are very powerful functional devices!

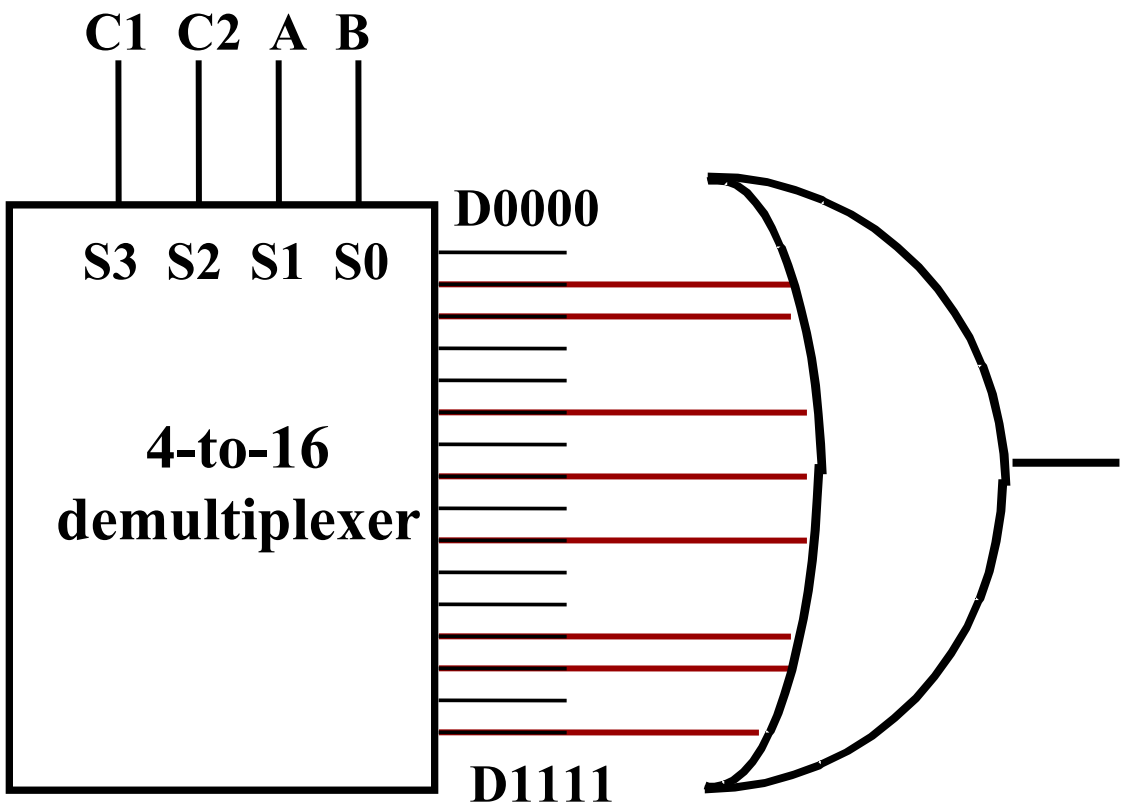
C1	C2	A	B	Out	
0	0	0	0	0	
0	0	0	1	1	A ⊕ B
0	0	1	0	1	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	B
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	1	1	A' · B
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	1	
1	1	0	1	1	A' + B
1	1	1	0	0	
1	1	1	1	1	

How to do your coursework in 30 seconds



C1	C2	A	B	Out	
0	0	0	0	0	
0	0	0	1	1	A ⊕ B
0	0	1	0	1	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	B
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	1	1	A' · B
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	1	
1	1	0	1	1	A' + B
1	1	1	0	0	
1	1	1	1	1	

How to do your coursework in 30 seconds



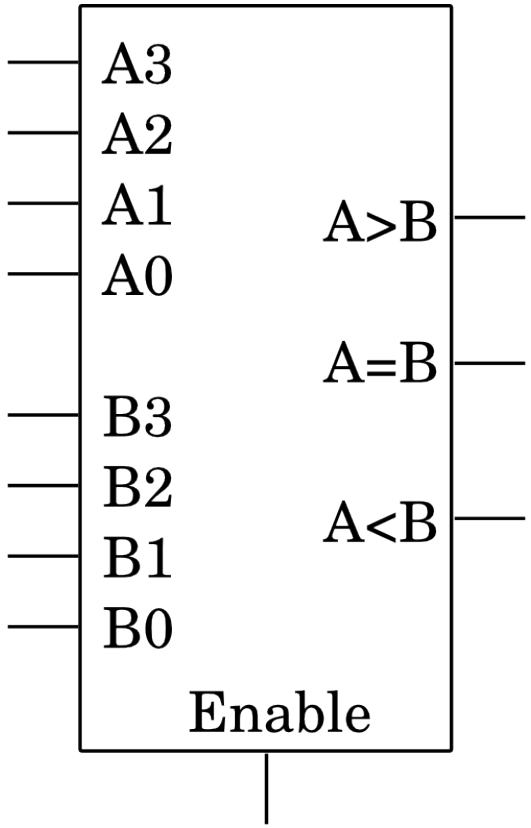
C1	C2	A	B	Out	
0	0	0	0	0	
0	0	0	1	1	A ⊕ B
0	0	1	0	1	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	B
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	1	1	A' · B
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	1	
1	1	0	1	1	A' + B
1	1	1	0	0	
1	1	1	1	1	

The decoder is a minterm generator!

Comparators (computer intelligence?)

Let's build a comparator circuit for two 4-bit positive binary numbers.

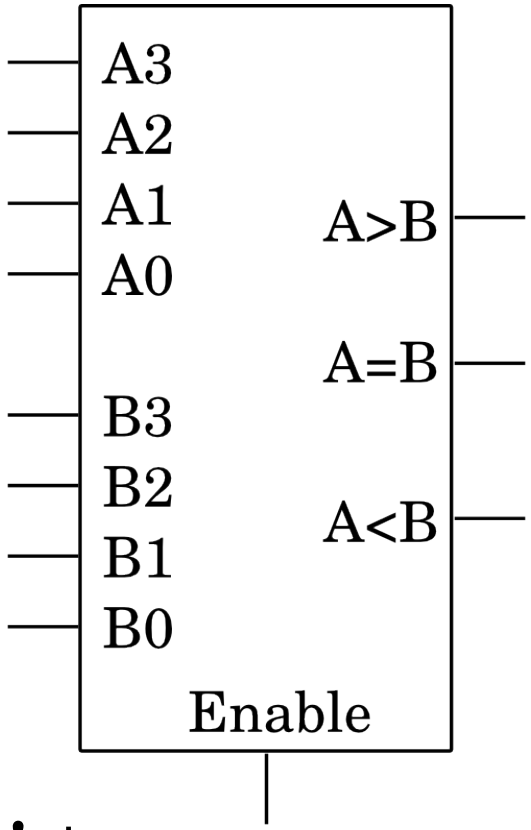
Nine inputs, three outputs (three 512 entries truth tables?)



Comparators (computer intelligence?)

Let's build a comparator circuit for two 4-bit positive binary numbers.

Nine inputs, three outputs (three 512 entries truth tables?)



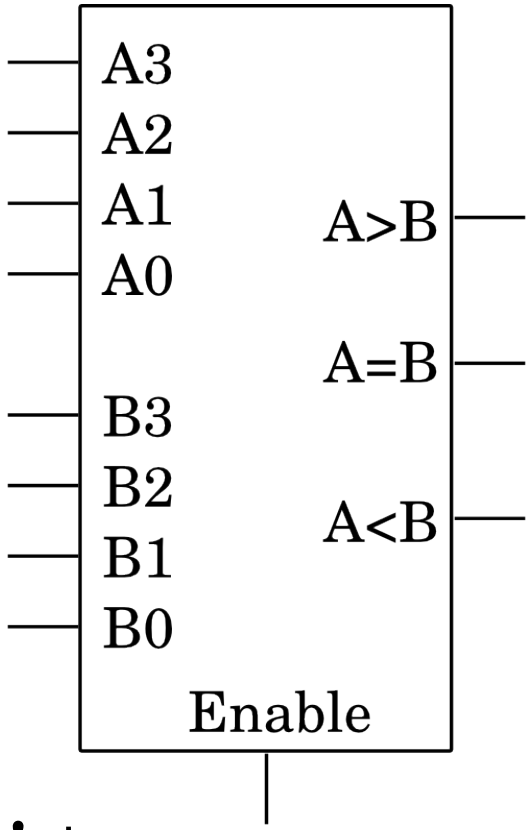
$$A>B = A_3B_3' + (A_3B_3+A_3'B_3')(A_2B_2' +$$

$$(A_2B_2+A_2'B_2')(A_1B_1' +(A_1B_1+A_1'B_1')A_0B_0'))$$

Comparators (computer intelligence?)

Let's build a comparator circuit for two 4-bit positive binary numbers.

Nine inputs, three outputs (three 512 entries truth tables?)



$$A>B = A_3B_3' + (A_3B_3+A_3'B_3')(A_2B_2' + (A_2B_2+A_2'B_2')(A_1B_1' +(A_1B_1+A_1'B_1')A_0B_0'))$$

YUK!

Designing Comparators Functionally

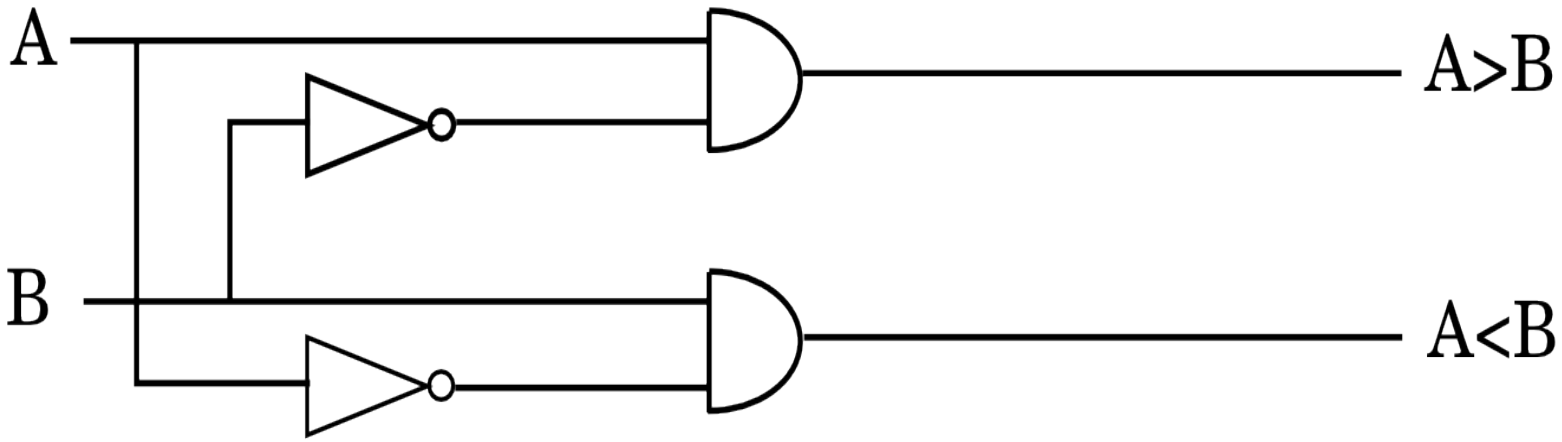
1. Build a one-bit comparator

A \ B	0	1
0	A=B	A<B
1	A>B	A=B

A>B : AB'

A<B : A'B

A=B : A'B' + AB



Designing Comparators Functionally

1. Build a one-bit comparator

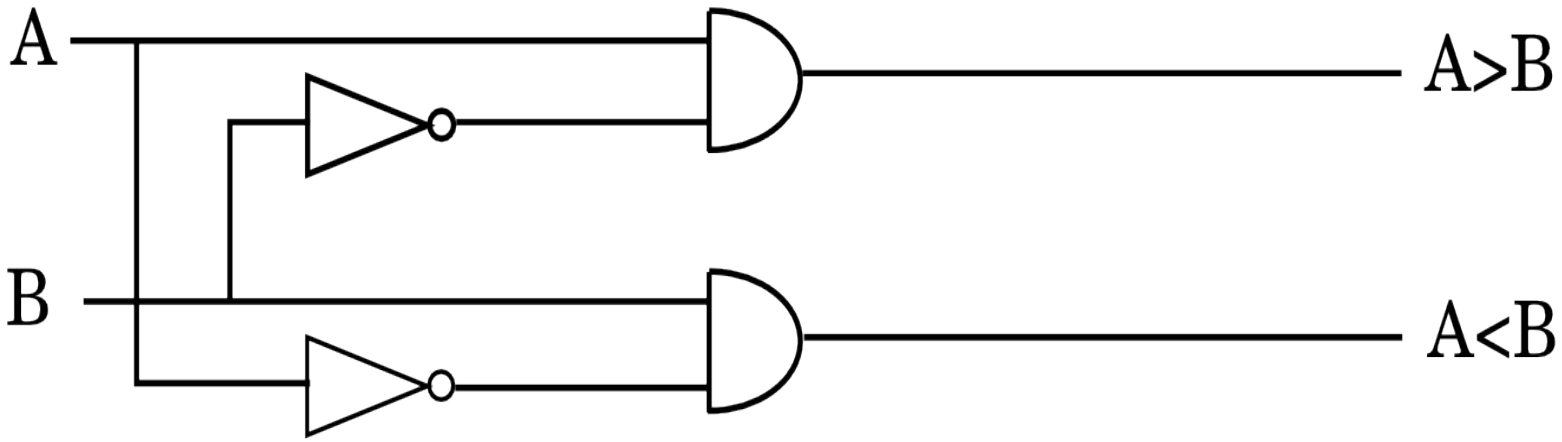
A \ B	0	1
0	A=B	A<B
1	A>B	A=B

A>B : AB'

A<B : A'B

A=B : A'B' + AB

OR What?



Designing Comparators Functionally

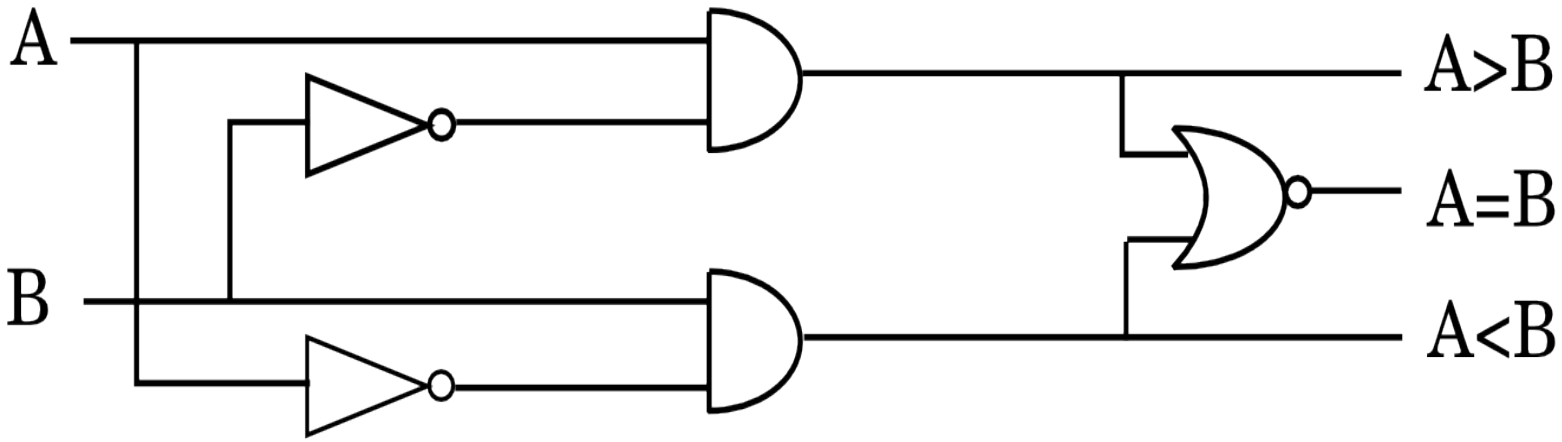
1. Build a one-bit comparator

A \ B	0	1
0	A=B	A<B
1	A>B	A=B

A>B : AB'

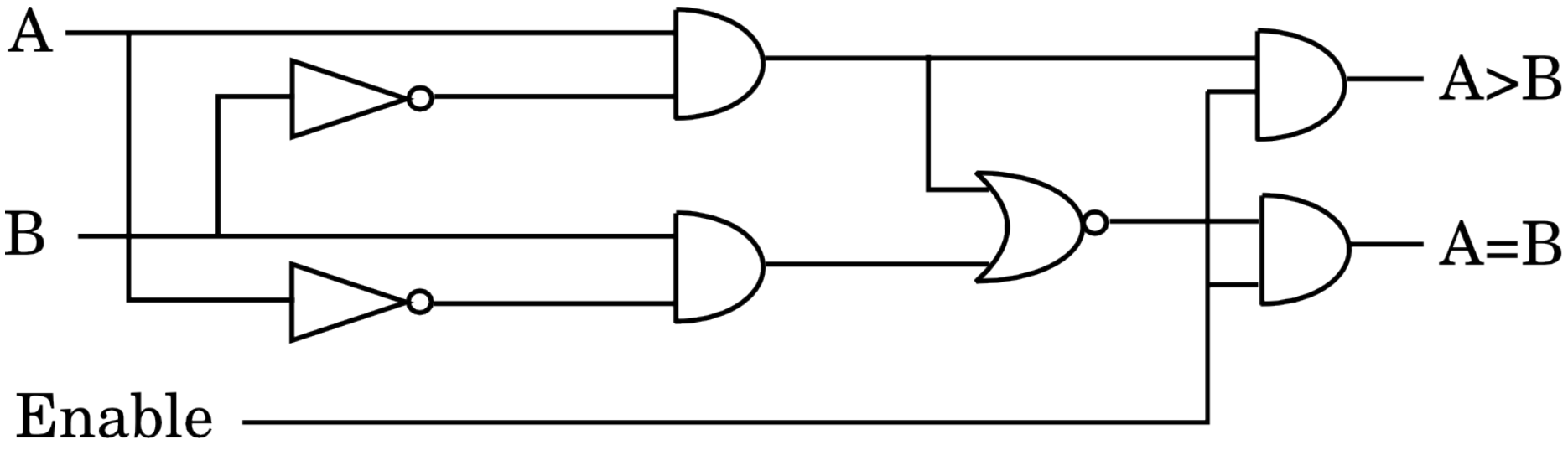
A<B : A'B

A=B : ((A>B)+(A<B))'

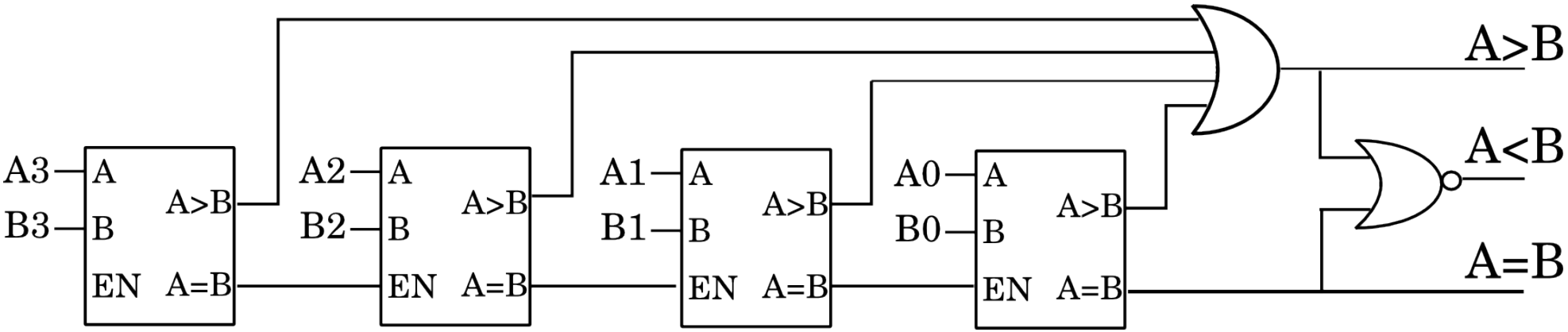


Designing Comparators Functionally

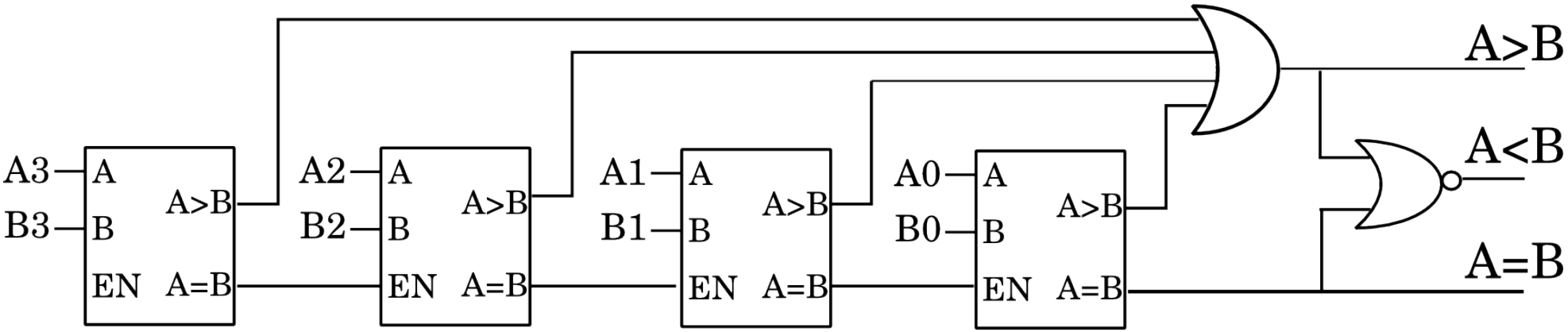
Add an enable line - and we will drop A<B since we won't need it.



Build a four-bit Comparator (functionally from four one-bit ones)



Build a four-bit Comparator (functionally from four one-bit ones)



Hooray for Functional Design !!!