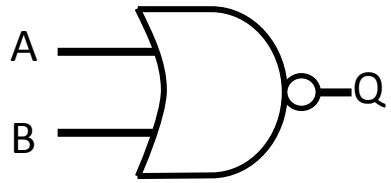
Today

- Today
  - Datapath modules
  - Complete processor using gates

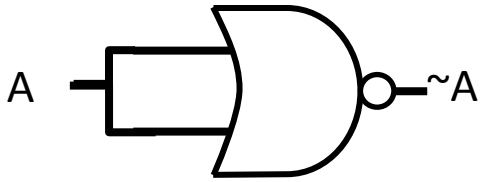
NOR Logic

NOR GATE

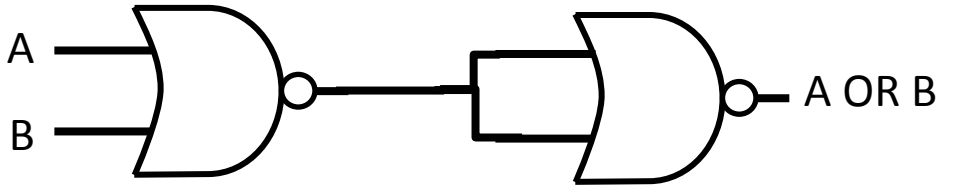


A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

NOT



OR



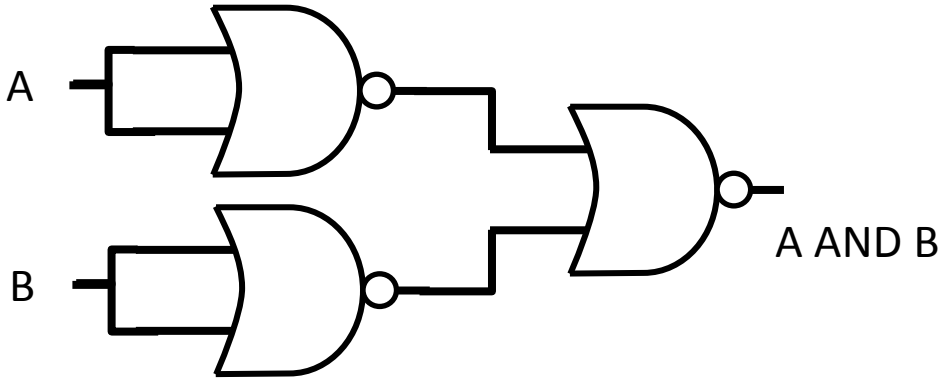
From De Morgan's Laws

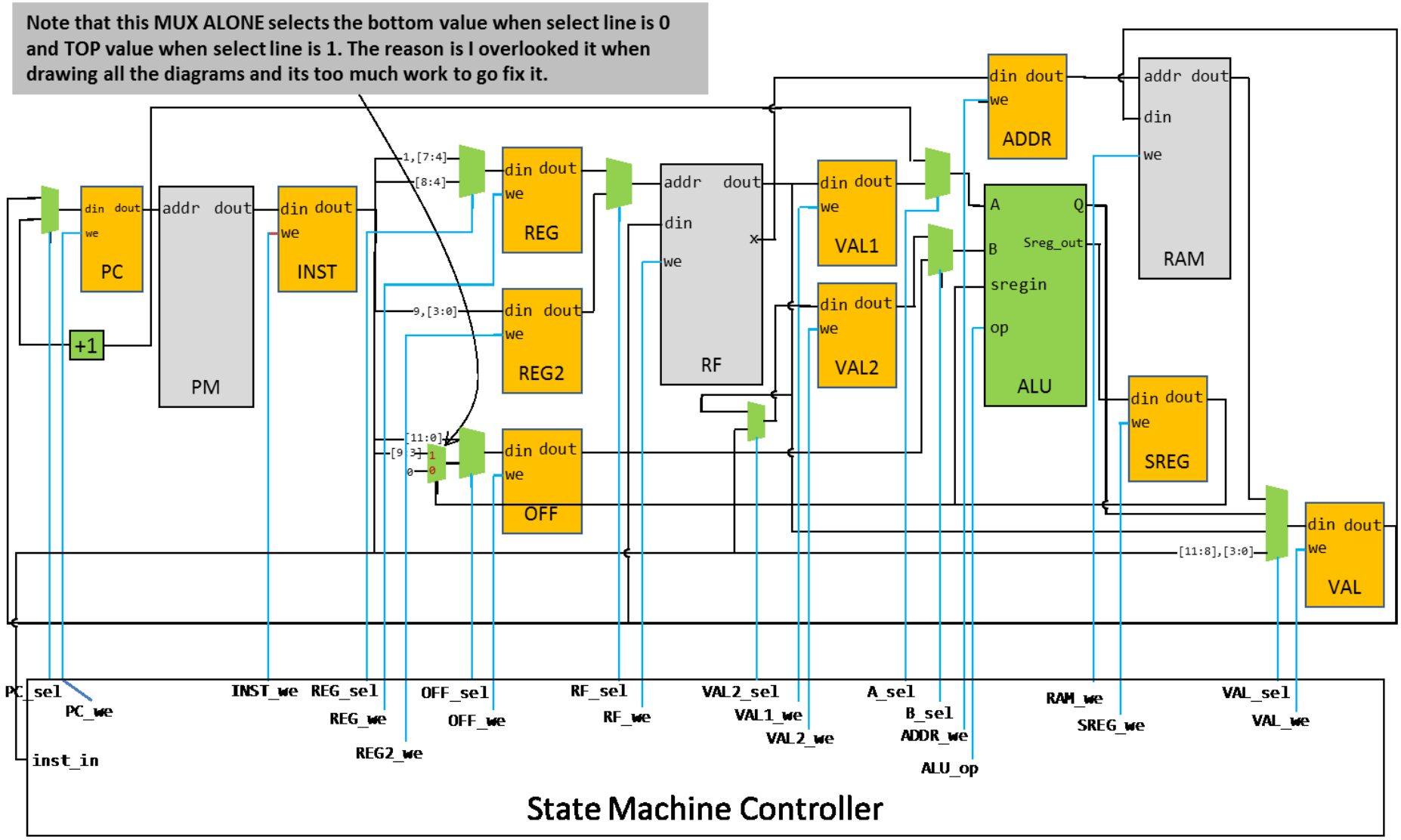
~(AB) = ~A OR ~B

~(~(A B)) = A B

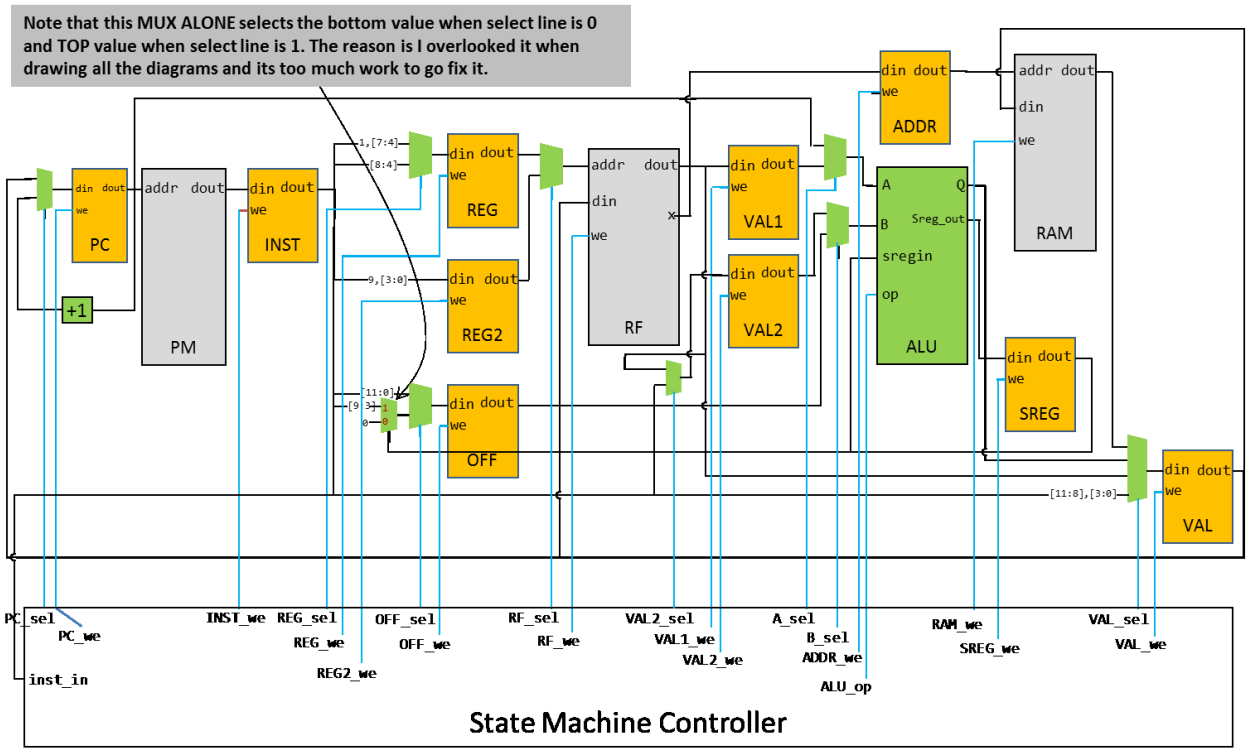
= ~ (~A OR ~B)

= ~A NOR ~B

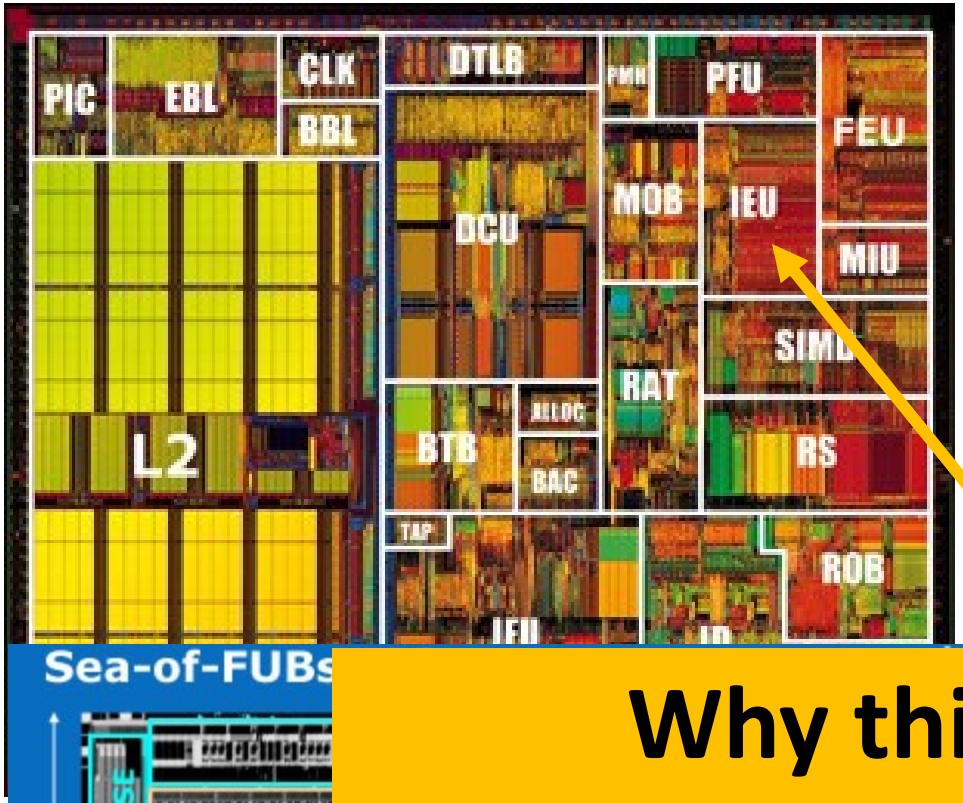


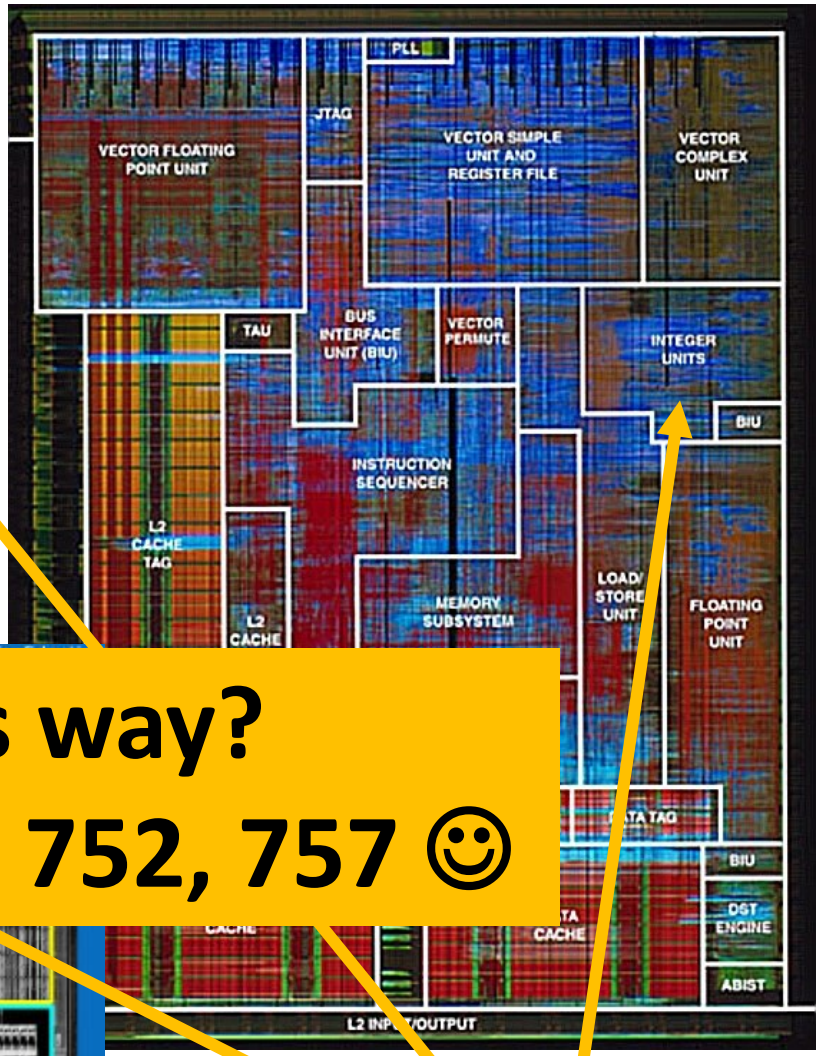


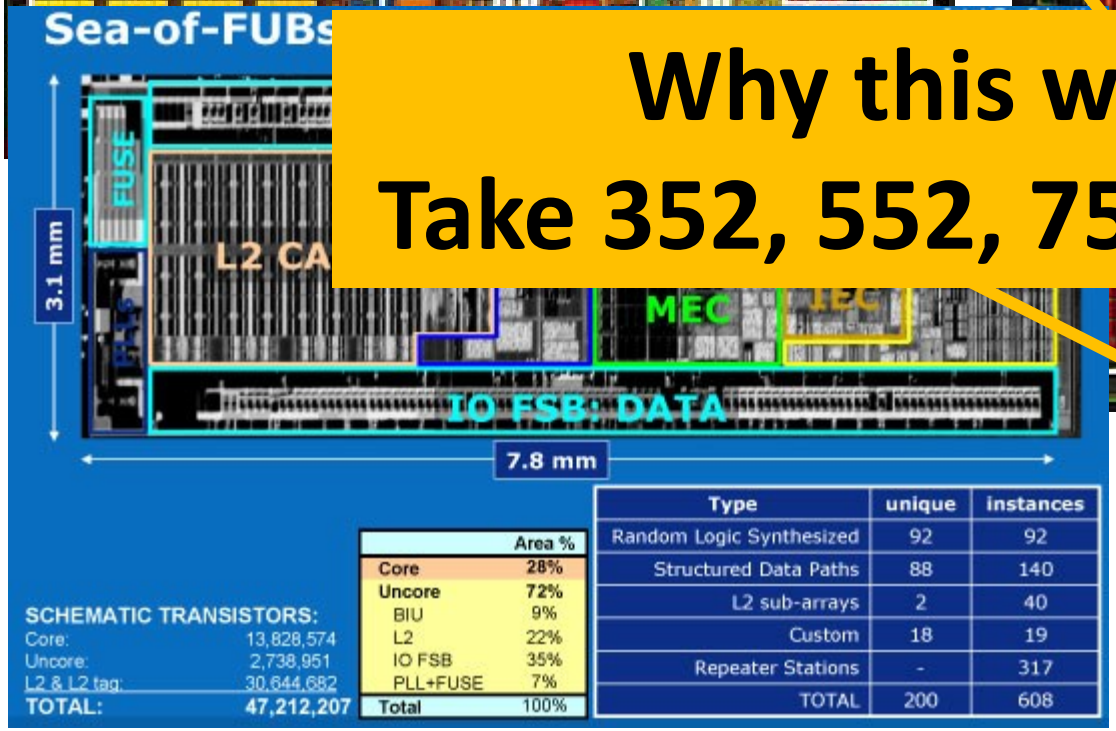
Goal: Build everything using only logic gates!



- Memories (PM, RF, RAM) – Friday
- Registers – Friday
- **MUX (ok!)**
- **ALU**
- **Incrementor (+1)**







	Area %
Core	28%
Uncore	72%
BIU	9%
L2	22%
IO FSB	35%
PLL+FUSE	7%
Total	100%

Type	unique	instances
Random Logic Synthesized	92	92
Structured Data Paths	88	140
L2 sub-arrays	2	40
Custom	18	19
Repeater Stations	-	317
TOTAL	200	608

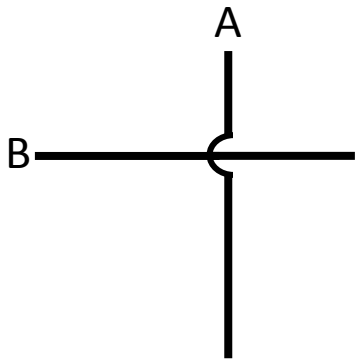
SCHEMATIC TRANSISTORS:	
Core:	13,828,574
Uncore:	2,738,951
L2 & L2 tag:	30,644,682
TOTAL:	47,212,207

Why this way?

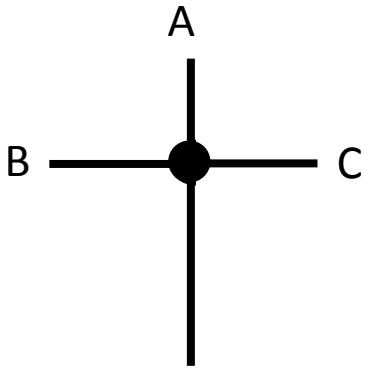
Take 352, 552, 752, 757 ☺

ALU

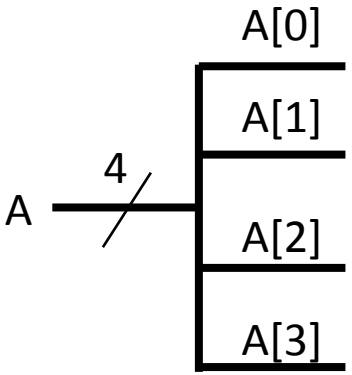
Drawing conventions



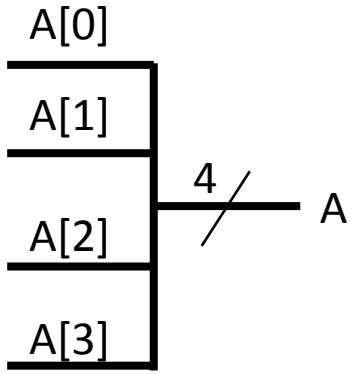
Means wires A and B are *not connected*



Means B and C are *copies* of A

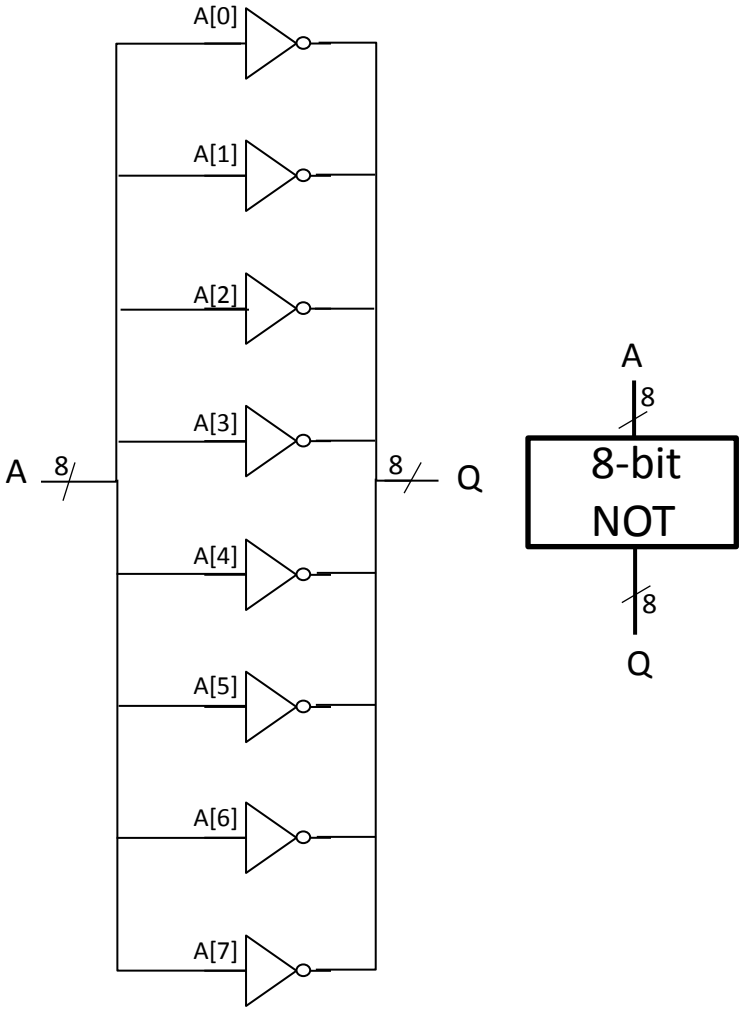


4 bits of A are split into *separate* 1-bit wires

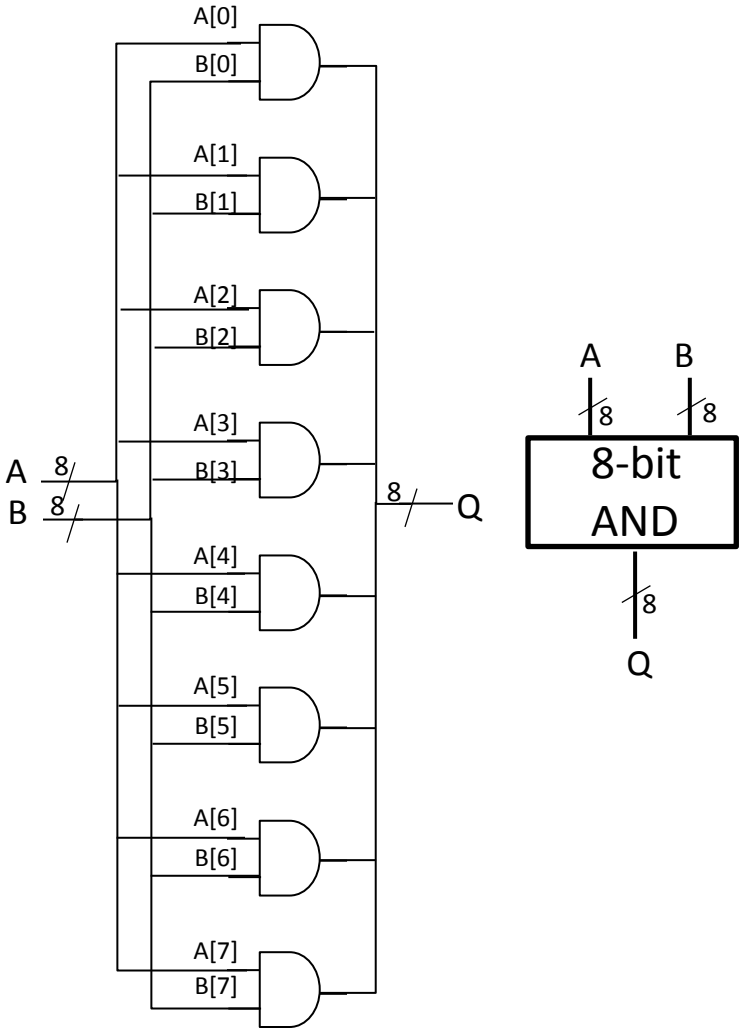


4 bits of A are *combined* into ane 4-bit wire

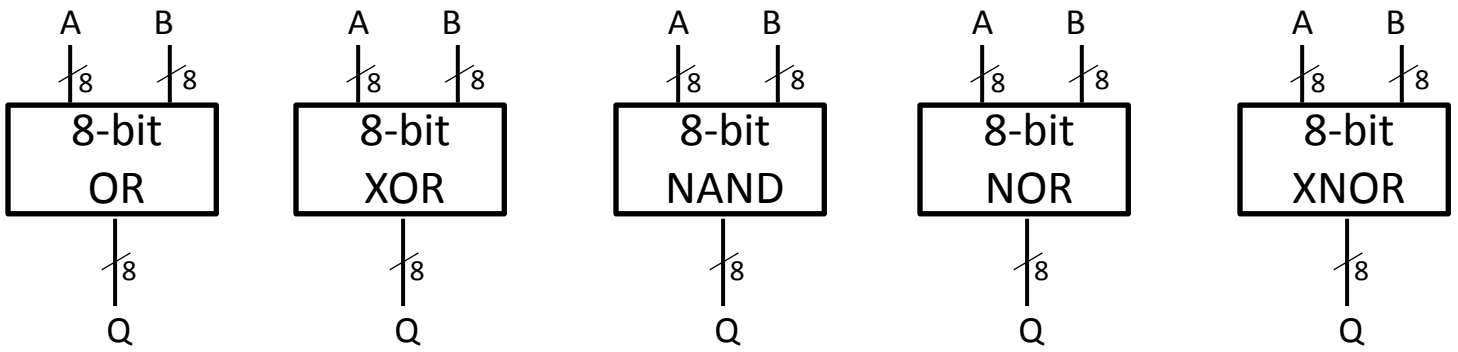
Multi-bit gates (NOT)



Multi-bit gates (AND)

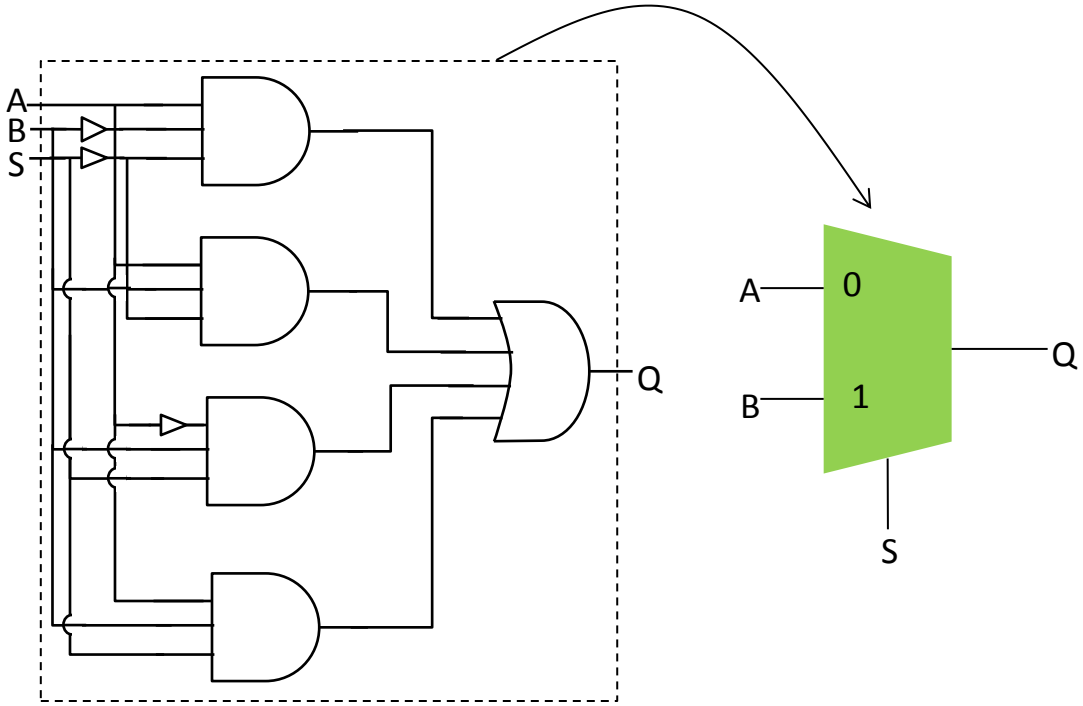


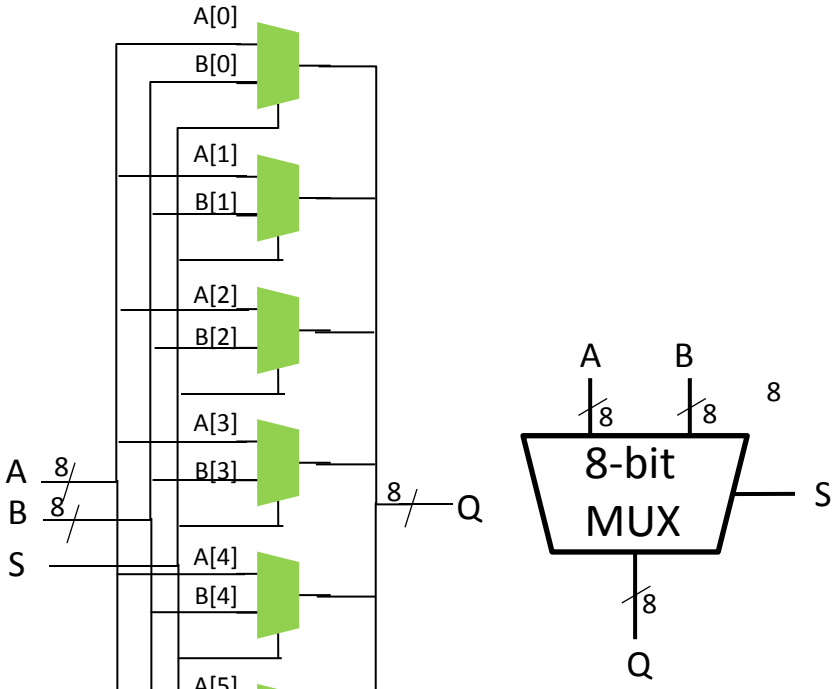
Other gates


A
B
8
8
8-bit
OR
8
Q
A
B
8
8
8-bit
XOR
8
Q
A
B
8
8
8-bit
NAND
8
Q
A
B
8
8
8-bit
NOR
8
Q
A
B
8
8
8-bit
XNOR
8
Q

Multiplexor

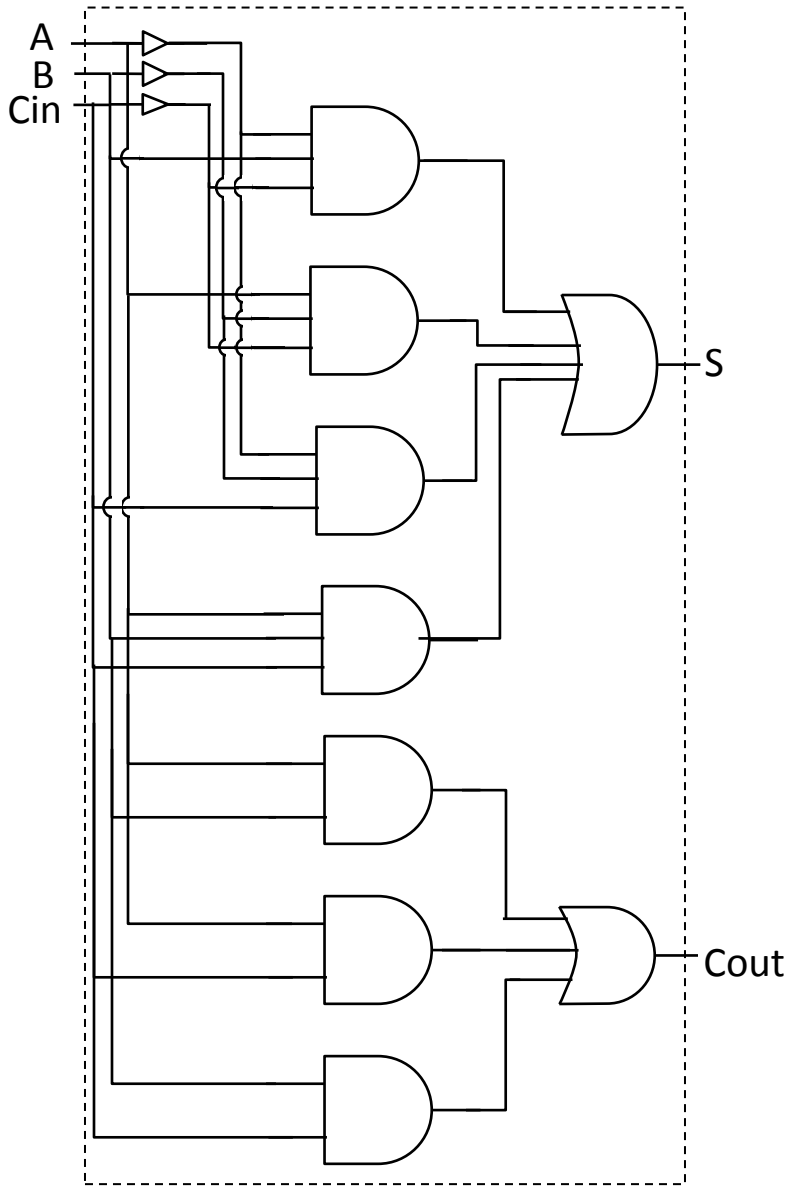
A	B	S	Q		
0	0	0	0		
0	1	0	0		
1	0	0	1		$A\ and \sim B\ and \sim S$
1	1	0	1		$A\ and\ B\ and \sim S$
0	0	1	0		
0	1	1	1		$\sim A\ and\ B\ and\ S$
1	0	1	0		
1	1	1	1		$A\ and\ B\ and\ S$

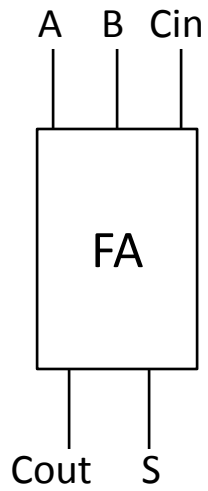



A[0]
B[0]
A[1]
B[1]
A[2]
B[2]
A[3]
B[3]
A[4]
B[4]
A[5]
B[5]
A[6]
B[6]
A[7]
B[7]
A
B
S
8
8
8
Q
A
B
8
8
8
8-bit
MUX
S
8
Q

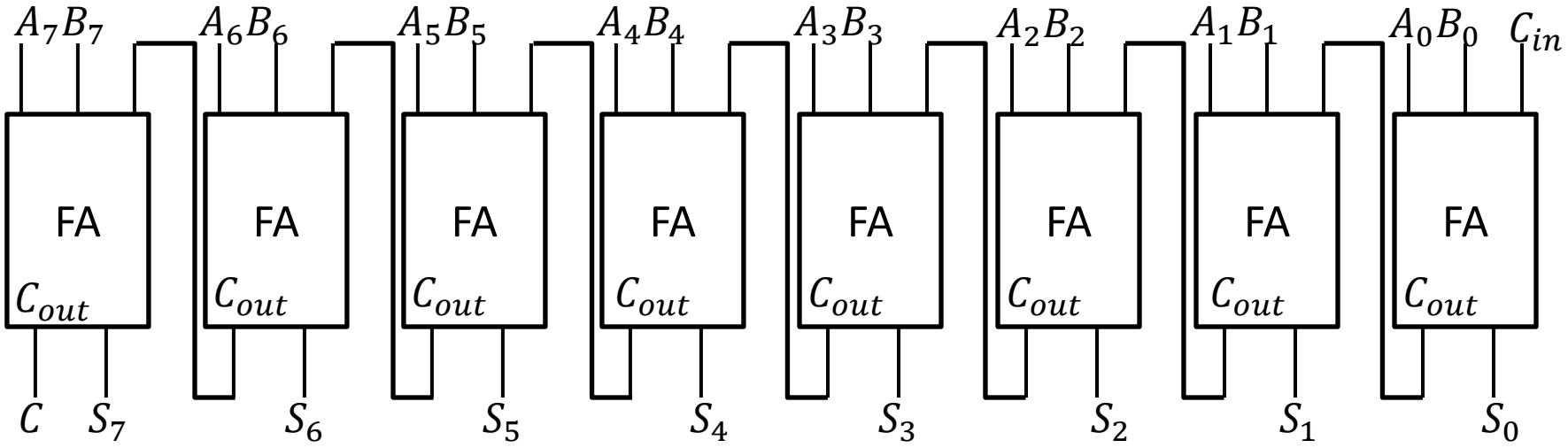
1-bit adder (FULL ADDER)

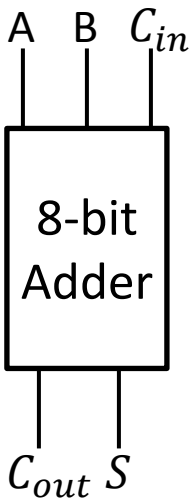
A	B	Cin	S	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1



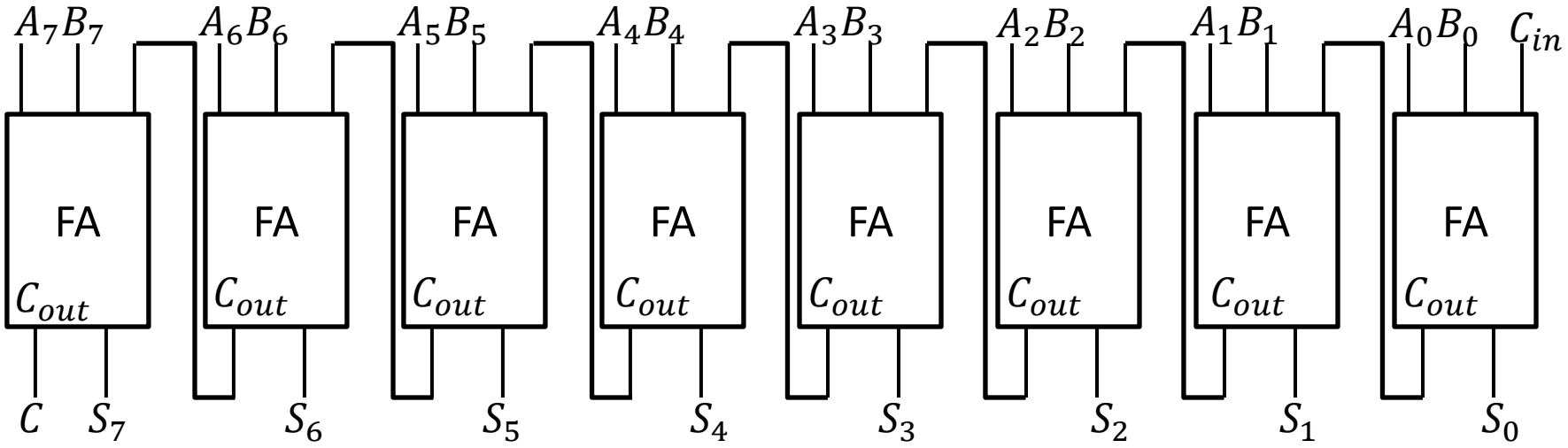


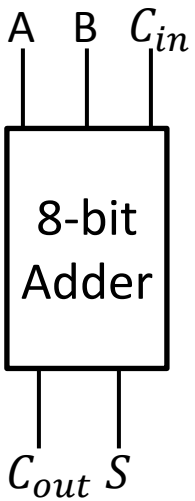
8-bit adder!


A_7B_7
A_6B_6
A_5B_5
A_4B_4
A_3B_3
A_2B_2
A_1B_1
A_0B_0
C_{in}
FA
C_{out}
C
S_7
S_6
S_5
S_4
S_3
S_2
S_1
S_0

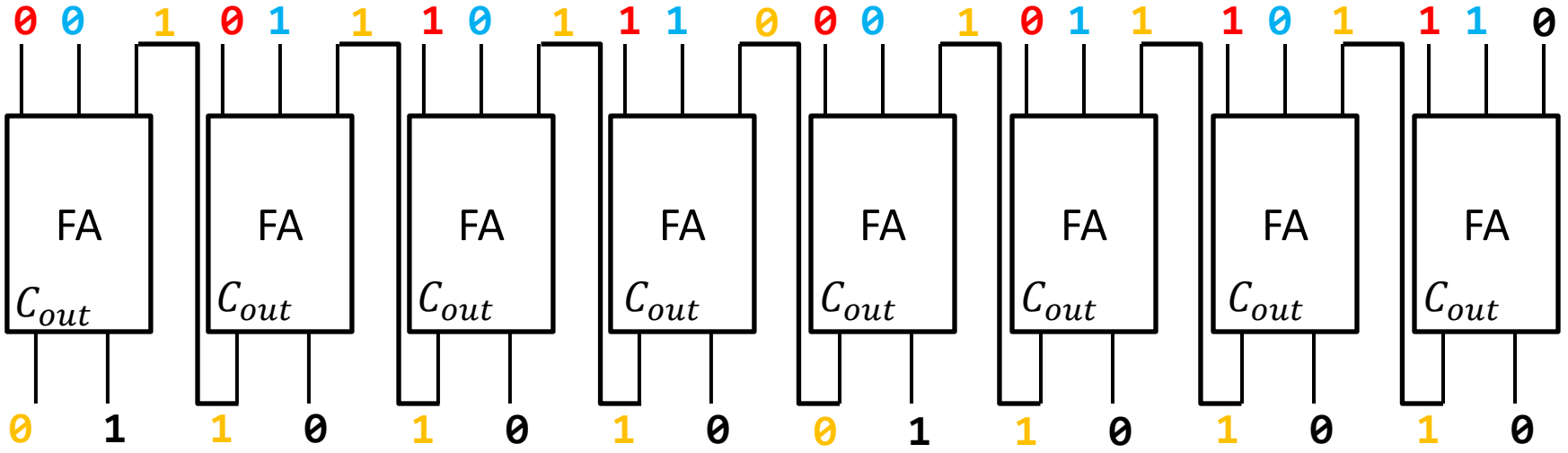

A
B
C_{in}
8-bit
Adder
C_{out}
S

8-bit adder!


A_7B_7
A_6B_6
A_5B_5
A_4B_4
A_3B_3
A_2B_2
A_1B_1
A_0B_0
C_{in}
FA
C_{out}
C
S_7
S_6
S_5
S_4
S_3
S_2
S_1
S_0


A
B
C_{in}
8-bit
Adder
C_{out}
S

8-bit adder: example

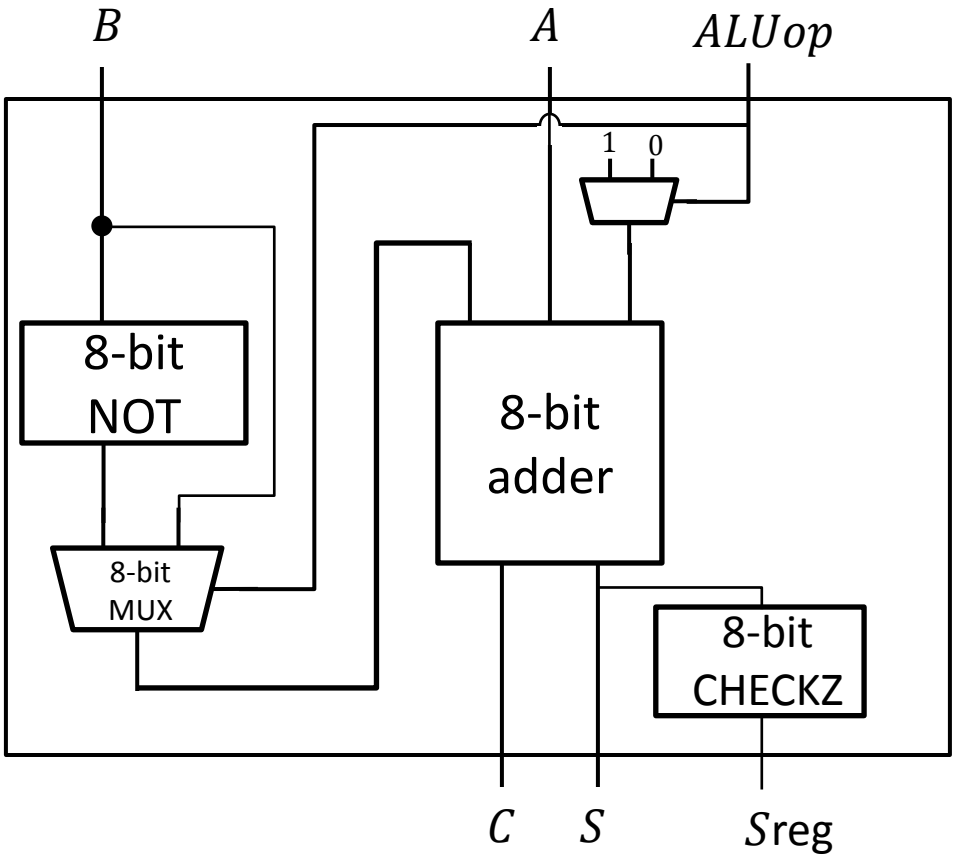


A = 0 0 1 1 0 0 1 1

B = 0 1 0 1 0 1 0 1

S = 1 0 0 0 1 0 0 0

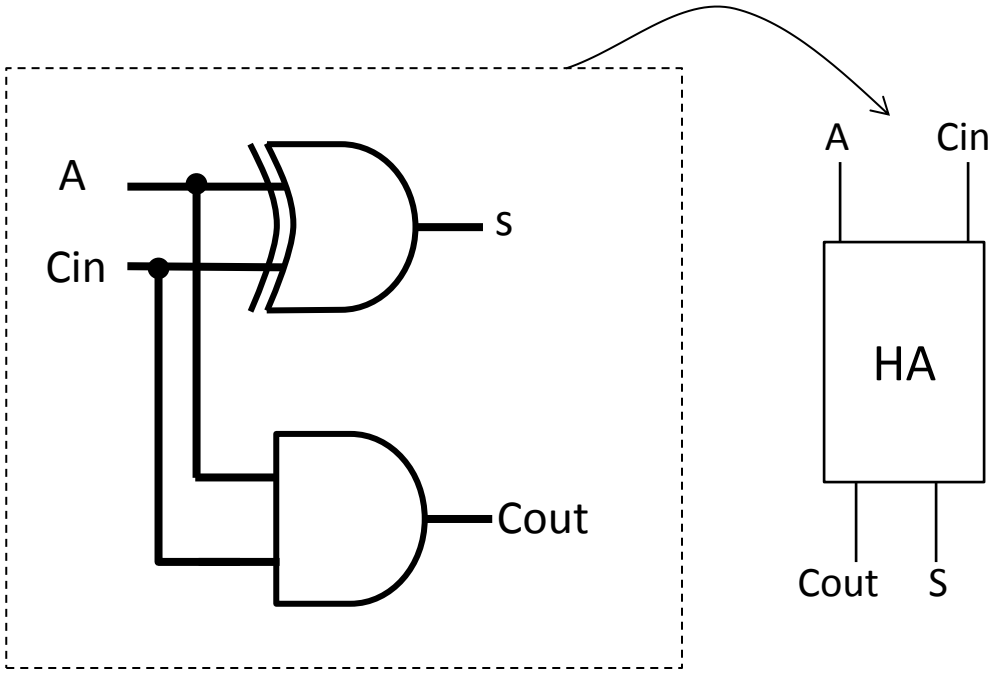
ALU



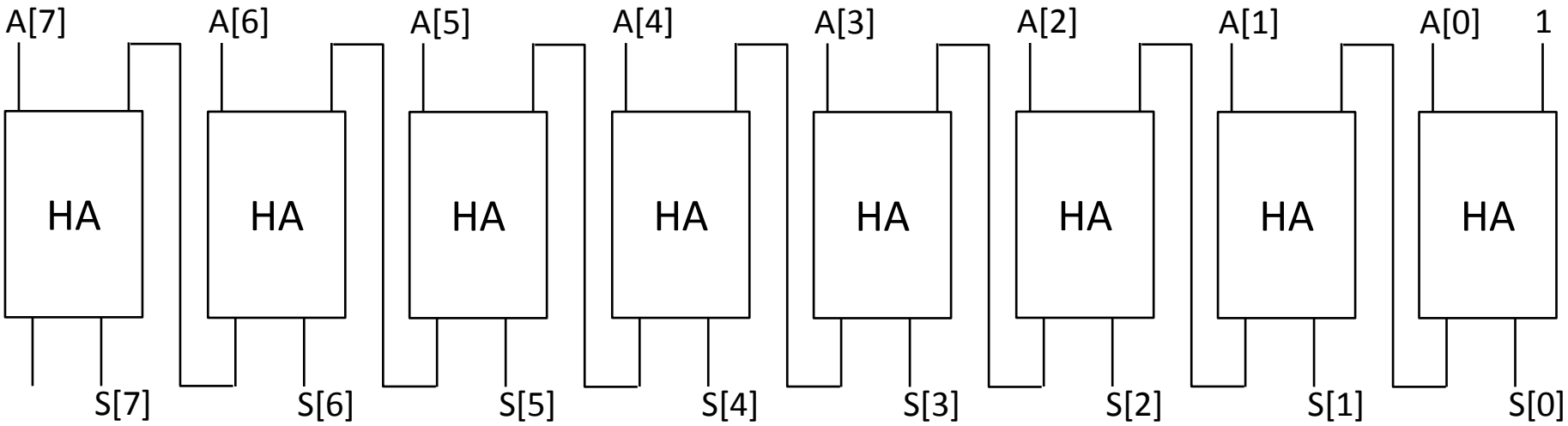
Incrementor using “half adder”

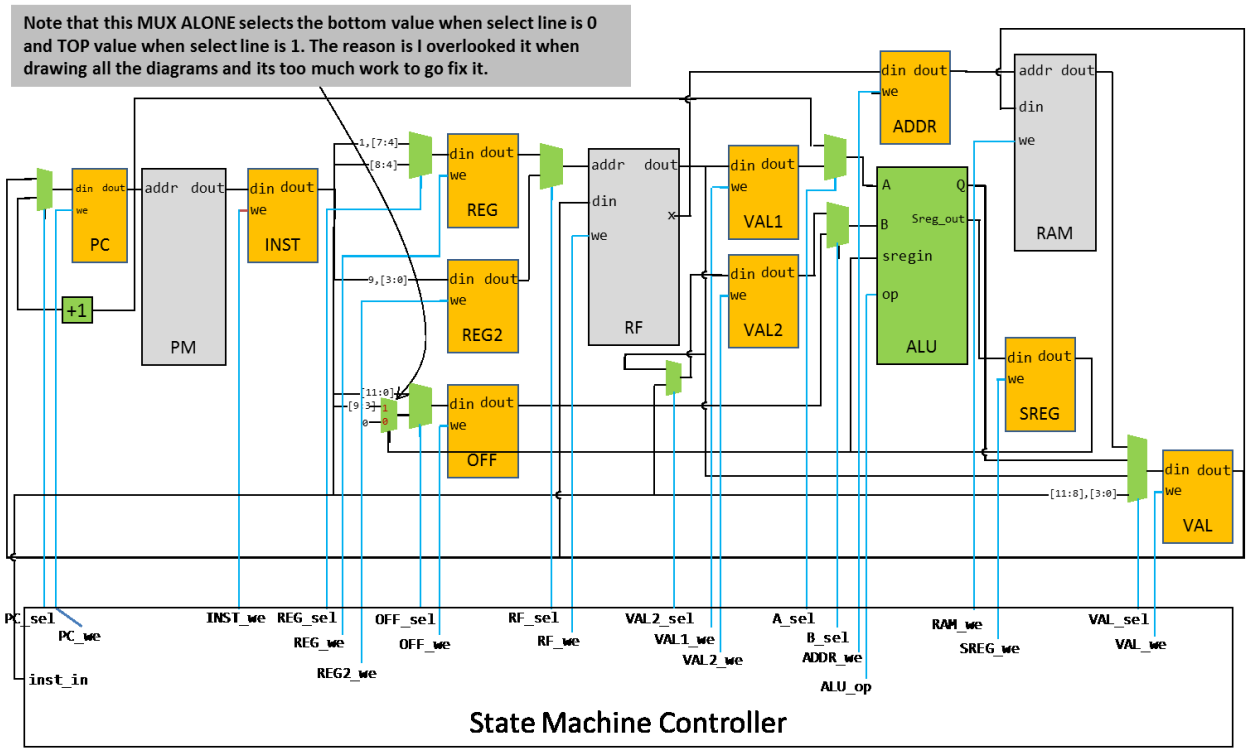
A	Cin	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S = A XOR Cin
Cout = A AND Cin



Incrementor





- Memories (PM, RF, RAM) – Friday
- Registers – Friday
- **MUX (done)**
- **ALU (done)**
- **Incrementor (done)**

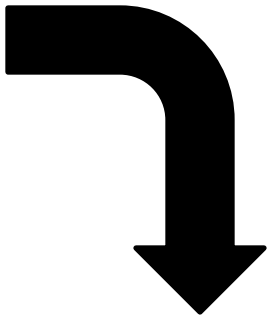
Decoder

- Convert n-bit signal into a representation where exactly one-bit is one.
- N-bit input, 2^N bit output

A[2]	A[1]	A[0]	d0	d1	d2	d3	d4	d5	d6	d7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

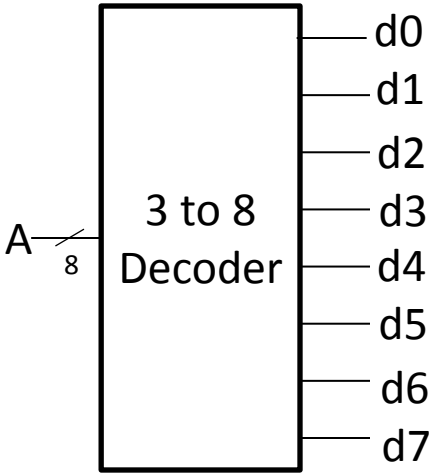
Decoder

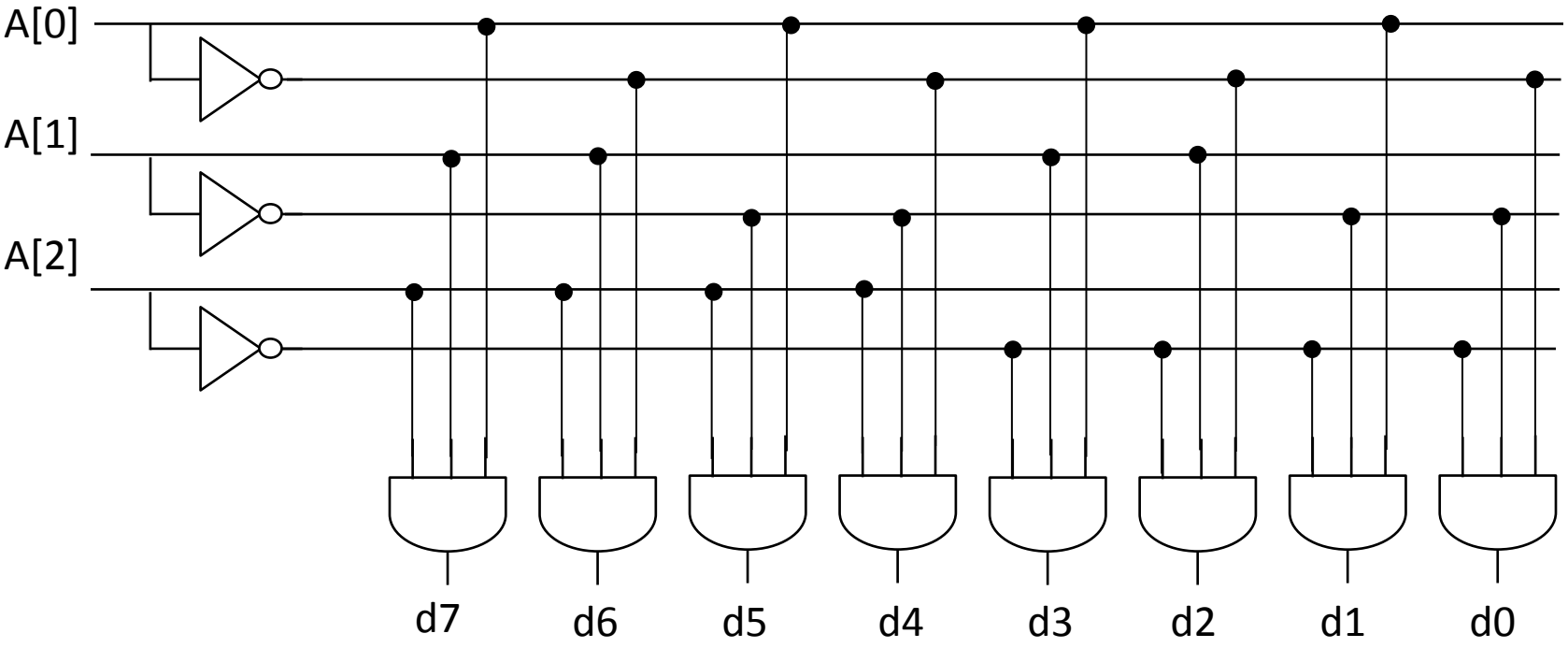
A[2]	A[1]	A[0]	d0	d1	d2	d3	d4	d5	d6	d7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



A[2]	A[1]	A[0]	Output	
0	0	0	d0=1	*d0=a[2]'a[1]'a[0]'*
0	0	1	d1=1	*d1=a[2]'a[1]'a[0]*
0	1	0	d2=1	*d2=a[2]'a[1]a[0]'*
0	1	1	d3=1	*d3=a[2]'a[1]a[0]*
1	0	0	d4=1	*d4=a[2]a[1]'a[0]'*
1	0	1	d5=1	*d5=a[2]a[1]'a[0]*
1	1	0	d6=1	*d6=a[2]a[1]a[0]'*
1	1	1	d7=1	*d7=a[2]a[1]a[0]*

A[2]	A[1]	A[0]	Output	
0	0	0	d0=1	*d0=a[2]'a[1]'a[0]'*
0	0	1	d1=1	*d1=a[2]'a[1]'a[0]*
0	1	0	d2=1	*d2=a[2]'a[1]a[0]'*
0	1	1	d3=1	*d3=a[2]'a[1]a[0]*
1	0	0	d4=1	*d4=a[2]a[1]'a[0]'*
1	0	1	d5=1	*d5=a[2]a[1]'a[0]*
1	1	0	d6=1	*d6=a[2]a[1]a[0]'*
1	1	1	d7=1	*d7=a[2]a[1]a[0]*



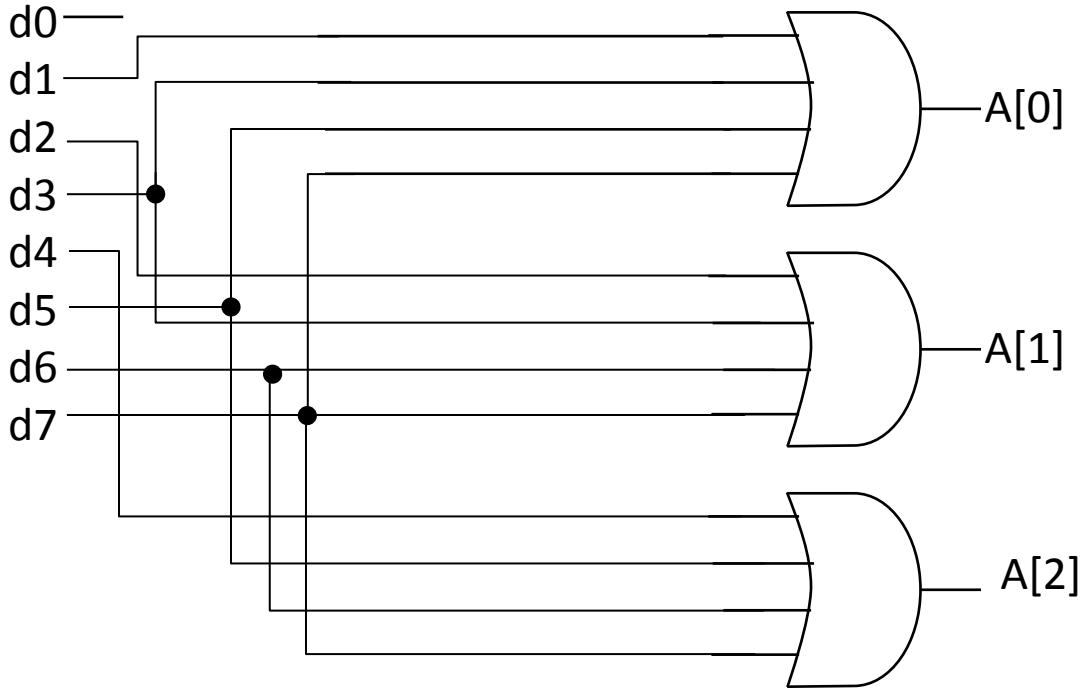


Encoder

- Takes a one-hot representation and convert to unsigned
- 2^N bit input, N-bit output

d7	d6	d5	d4	d3	d2	d1	d0	A[2]	A[1]	A[0]
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	**1**
0	0	0	0	0	1	0	0	0	**1**	0
0	0	0	0	1	0	0	0	0	**1**	**1**
0	0	0	1	0	0	0	0	**1**	0	0
0	0	1	0	0	0	0	0	**1**	0	**1**
0	1	0	0	0	0	0	0	**1**	**1**	0
1	0	0	0	0	0	0	0	**1**	**1**	**1**

$$A[0] = d1.d3.d5.d7$$
$$A[1] = d2.d3.d6.d7$$
$$A[0] = d4.d5.d6.d7$$



We will use encoders and decoders to build memories

Today

- Today
  - Datapath modules
  - Complete processor using gates