COMPUTER SYSTEMS ARCHITECTURE SEMINAR 0x02

SEMINAR SUPPORT NOTES

Cristian Rusu, Bogdan Macovei

TRUTH TABLES, EX 1

$X = A + BC$

A	B	C	BC	X

TRUTH TABLES, EX 1

X = A + BC

A	B	C	BC	X
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

TRUTH TABLES, EX 1

X = A + BC

A	B	C	BC	X
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	

TRUTH TABLES, EX 1

X = A + BC

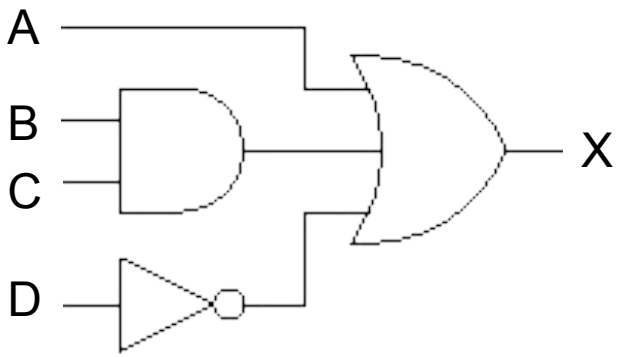
A	B	C	BC	X
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

DRAW THE CIRCUIT, EX 2

$X = A + BC + \overline{D}$

DRAW THE CIRCUIT, EX 2

$$X = A + BC + \overline{D}$$



DE MORGAN, EX 3

$$X = \overline{(\overline{A\bar{B}})(\bar{B}+C)}$$

$=$

$=$

$=$

$=$

$=$

DE MORGAN, EX 3

$$X = \overline{\overline{(A\bar{B})}(\bar{B}+C)}$$

$$= \overline{\overline{(A\bar{B})}} + \overline{(\bar{B}+C)}$$

$$=$$

$$=$$

$$=$$

$$=$$

DE MORGAN, EX 3

$$X = \overline{\overline{(A\bar{B})}(\bar{B}+C)}$$

$$= \overline{\overline{(A\bar{B})}} + \overline{(\bar{B}+C)}$$

$$= A\bar{B} + \overline{(\bar{B}+C)}$$

$$=$$

$$=$$

$$=$$

DE MORGAN, EX 3

$$
\begin{aligned}
X &= \overline{(\overline{A\bar{B}})(\bar{B}+C)} \\
&= \overline{(\overline{A\bar{B}})} + \overline{(\bar{B}+C)} \\
&= A\bar{B} + \overline{(\bar{B}+C)} \\
&= A\bar{B} + (\bar{\bar{B}}\bar{C}) \\
&= \\
&=
\end{aligned}
$$

DE MORGAN, EX 3

$$
\begin{aligned}
X &= \overline{\overline{(A\bar{B})}(\bar{B}+C)} \\
&= \overline{\overline{(A\bar{B})}} + \overline{(\bar{B}+C)} \\
&= A\bar{B} + \overline{(\bar{B}+C)} \\
&= A\bar{B} + (\bar{\bar{B}}\bar{C}) \\
&= A\bar{B} + (B\bar{C}) \\
&=
\end{aligned}
$$

DE MORGAN, EX 3

$$
\begin{aligned}
X &= \overline{\overline{(A\bar{B})}(\bar{B}+C)} \\
&= \overline{\overline{(A\bar{B})}} + \overline{(\bar{B}+C)} \\
&= A\bar{B} + \overline{(\bar{B}+C)} \\
&= A\bar{B} + (\bar{\bar{B}}\bar{C}) \\
&= A\bar{B} + (B\bar{C}) \\
&= A\bar{B} + B\bar{C}
\end{aligned}
$$

DE MORGAN, EX 3

$$X = \overline{(\bar{\mathrm{A}} + \mathrm{C})(\overline{\mathrm{AB}})}$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

DE MORGAN, EX 3

$$X = \overline{(\bar{\mathrm{A}} + \mathrm{C})(\overline{\mathrm{AB}})}$$

$$= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + \overline{(\overline{\mathrm{AB}})}$$

$$=$$

$$=$$

$$=$$

$$=$$

DE MORGAN, EX 3

$$
\begin{aligned}
X &= \overline{(\bar{\mathrm{A}} + \mathrm{C})(\overline{\mathrm{AB}})} \\
&= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + \overline{(\overline{\mathrm{AB}})} \\
&= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + (\mathrm{AB}) \\
&= \\
&= \\
&=
\end{aligned}
$$

DE MORGAN, EX 3

$$
\begin{aligned}
X &= \overline{(\bar{\mathrm{A}} + \mathrm{C})(\overline{\mathrm{AB}})} \\
&= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + \overline{(\overline{\mathrm{AB}})} \\
&= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + (\mathrm{AB}) \\
&= (\overline{\bar{\mathrm{A}}}\,\overline{\mathrm{C}}) + (\mathrm{AB}) \\
&= \\
&=
\end{aligned}
$$

DE MORGAN, EX 3

$$
\begin{aligned}
X &= \overline{(\bar{\mathrm{A}} + \mathrm{C})(\overline{\mathrm{AB}})} \\
&= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + \overline{(\overline{\mathrm{AB}})} \\
&= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + (\mathrm{AB}) \\
&= (\overline{\bar{\mathrm{A}}}\,\overline{\mathrm{C}}) + (\mathrm{AB}) \\
&= (\mathrm{A}\overline{\mathrm{C}}) + (\mathrm{AB}) \\
&=
\end{aligned}
$$

DE MORGAN, EX 3

$$
\begin{aligned}
X &= \overline{(\bar{\mathrm{A}} + \mathrm{C})(\overline{\mathrm{AB}})} \\
&= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + \overline{(\overline{\mathrm{AB}})} \\
&= \overline{(\bar{\mathrm{A}} + \mathrm{C})} + (\mathrm{AB}) \\
&= (\overline{\bar{\mathrm{A}}}\,\overline{\mathrm{C}}) + (\mathrm{AB}) \\
&= (\mathrm{A}\overline{\mathrm{C}}) + (\mathrm{AB}) \\
&= \mathrm{A}(\mathrm{B} + \overline{\mathrm{C}})
\end{aligned}
$$

DE MORGAN, EX 3

- !(!A+!B) = AB
- !(!A!B) = A+B
- !(A+B+C) = !A!B!C
- !(ABC) = !A+!B+!C
- !(A+B)!A!B = !A!B
- !(AB)(!A+!B) = !A+!B
- !(A+B)(!A+!B) = !A!B
- !A!B!(AB) = !A!B
- C+!(CB) = 1
- !(AB)(!A+B)(!B+!B) = !A!B

SIMPLIFICATIONS, EX 4

$$(A + C)(AD + A\bar{D}) + AC + C$$

SIMPLIFICATIONS, EX 4

$$(A + C)(AD + A\bar{D}) + AC + C$$

$$(A + C)A(D + \bar{D}) + AC + C \quad \text{// factor out}$$

SIMPLIFICATIONS, EX 4

$(A + C)(AD + A\bar{D}) + AC + C$

$(A + C)A(D + \bar{D}) + AC + C$ // factor out

$(A + C)A + AC + C$ // a variable OR its complement

SIMPLIFICATIONS, EX 4

$(A + C)(AD + A\bar{D}) + AC + C$

$(A + C)A(D + \bar{D}) + AC + C$ // factor out

$(A + C)A + AC + C$ // a variable OR its complement

$A((A + C) + C) + C$ // factor out

SIMPLIFICATIONS, EX 4

$(A + C)(AD + A\bar{D}) + AC + C$

$(A + C)A(D + \bar{D}) + AC + C$ // factor out

$(A + C)A + AC + C$ // a variable OR its complement

$A((A + C) + C) + C$ // factor out

$A(A + C) + C$ // associativity, idempotence

SIMPLIFICATIONS, EX 4

$(A + C)(AD + A\bar{D}) + AC + C$

$(A + C)A(D + \bar{D}) + AC + C$ // factor out

$(A + C)A + AC + C$ // a variable OR its complement

$A((A + C) + C) + C$ // factor out

$A(A + C) + C$ // associativity, idempotence

$AA + AC + C$ // distribute

SIMPLIFICATIONS, EX 4

$(A + C)(AD + A\bar{D}) + AC + C$

$(A + C)A(D + \bar{D}) + AC + C$ // factor out

$(A + C)A + AC + C$ // a variable OR its complement

$A((A + C) + C) + C$ // factor out

$A(A + C) + C$ // associativity, idempotence

$AA + AC + C$ // distribute

$A + (A + 1)C$ // idempotence, identity, factoring

SIMPLIFICATIONS, EX 4

$(A + C)(AD + A\bar{D}) + AC + C$

$(A + C)A(D + \bar{D}) + AC + C$ // factor out

$(A + C)A + AC + C$ // a variable OR its complement

$A((A + C) + C) + C$ // factor out

$A(A + C) + C$ // associativity, idempotence

$AA + AC + C$ // distribute

$A + (A + 1)C$ // idempotence, identity, factoring

$A + C$ // apply identity laws twice

SIMPLIFICATIONS, EX 4

$$\bar{\mathrm{A}}(\mathrm{A} + \mathrm{B}) + (\mathrm{B} + \mathrm{AA})(\mathrm{A} + \bar{\mathrm{B}})$$

SIMPLIFICATIONS, EX 4

$$\bar{A}(A + B) + (B + AA)(A + \bar{B})$$

$$\bar{A}(A + B) + (B + A)(A + \bar{B}) \text{ // AA is A}$$

SIMPLIFICATIONS, EX 4

$$\bar{A}(A + B) + (B + AA)(A + \bar{B})$$

$$\bar{A}(A + B) + (B + A)(A + \bar{B}) \text{ // AA is A}$$

$$(A + B)(\bar{A} + A + \bar{B}) \text{ // factor out A + B}$$

SIMPLIFICATIONS, EX 4

$\bar{A}(A + B) + (B + AA)(A + \bar{B})$

$\bar{A}(A + B) + (B + A)(A + \bar{B})$ // AA is A

$(A + B)(\bar{A} + A + \bar{B})$ // factor out A + B

$(A + B)(1 + \bar{B})$ // a variable OR its

SIMPLIFICATIONS, EX 4

$\bar{A}(A + B) + (B + AA)(A + \bar{B})$

$\bar{A}(A + B) + (B + A)(A + \bar{B})$ // AA is A

$(A + B)(\bar{A} + A + \bar{B})$ // factor out A + B

$(A + B)(1 + \bar{B})$ // a variable OR its complement

$A + B$ // 1 OR anything is 1

SIMPLIFICATIONS, EX 4

a) A+0 = A

b) !Ax0 = 0

c) A+!A = 1

d) A+A = A

e) A+AB = A

f) A+!AB = A+B

g) A(!A+B) = AB

h) AB+!AB = B

i) (!A!B+!AB) = !A

j) A(A+B+C+...) = A

k) subitems

   a) A+B
   b) 1
   c) 1

l) A+A!A = A

m) AB+A!B = A

n) !A+B!A = !A

o) (D+!A+B+!C)B = B

p) (A+!B)(A+B) = A

q) C(C+CD) = C

r) A(A+AB) = A

s) !(!A+!A) = A

t) !(A+!A) = 0

u) D+(D!CBA) = D

v) !D!(DBCA) = !D

w) AC+!AB+BC = AC+!AB

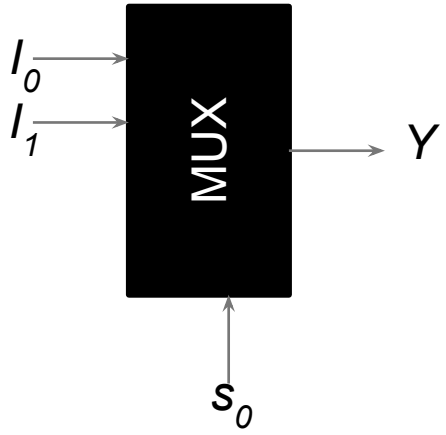
x) (A+C)(!A+B)(B+C) = AB+!AC

y) !A+!B+AB!C = !A+!B+!C

$(A+B)^2+(A+B)^3+A+3!A+A^3 = 1$

MUX, EX 5 A

- **MUX, two inputs, one selection signal s, and one output**

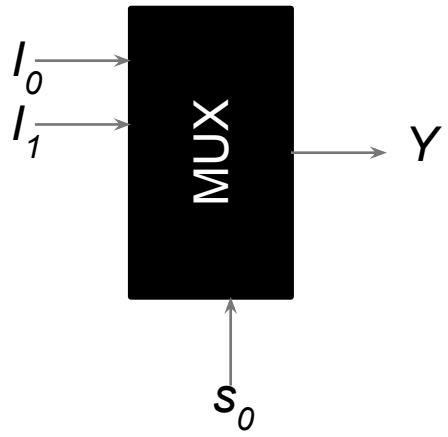


I_0	I_1	s_0	Y
*	*	0	I_0
*	*	1	I_1

- **what is the output-input relation?**
  - Y = ?

MUX, EX 5 A

- **MUX, two inputs, one selection signal s, and one output**

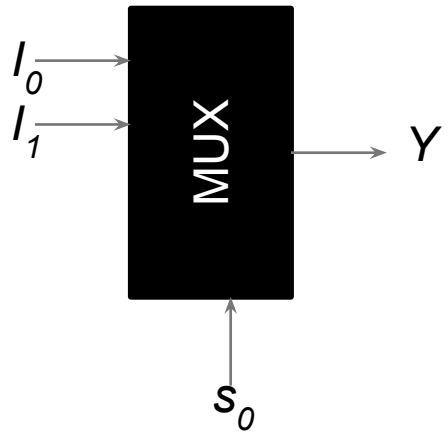


I_0	I_1	s_0	Y
*	*	0	I_0
*	*	1	I_1

- **what is the input-output relationship?**
  - $Y = I_0\bar{s}_0 + I_1 s_0$

MUX, EX 5 B

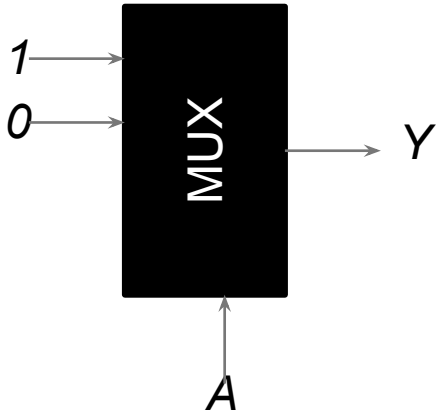
- **MUX, two inputs, one selection signal s, and one output**



- **a MUX is a universal circuit, i.e. it can implement NOT, OR, and AND gates**
  - implement a NOT gate using a MUX

MUX, EX 5 B

- **MUX, two inputs, one selection signal s, and one output**

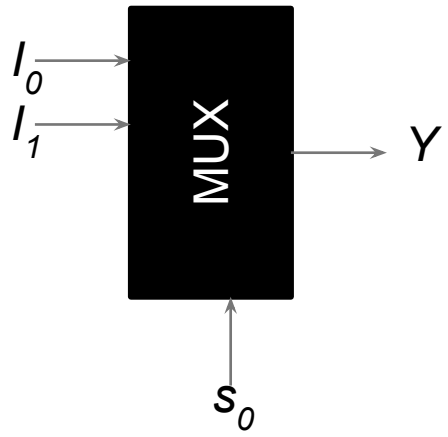


I_0	I_1	s_0 (A)	Y
1	0	0	I_0
1	0	1	I_1

- **a MUX is a universal circuit, i.e. it can implement NOT, OR, and AND gates**
  - implement a NOT gate using a MUX: Y = NOT A
  - $Y = I_0\bar{s}_0 + I_1 s_0 = 1\bar{A} + 0A = \bar{A}$

MUX, EX 5 B

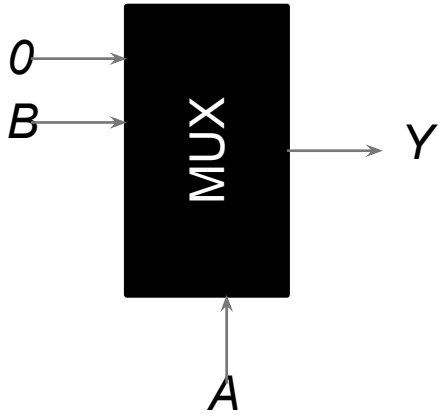
- **MUX, two inputs, one selection signal s, and one output**



- **a MUX is a universal circuit, i.e. it can implement NOT, OR, and AND gates**
  - implement an AND gate using a MUX

MUX, EX 5 B

- **MUX, two inputs, one selection signal s, and one output**

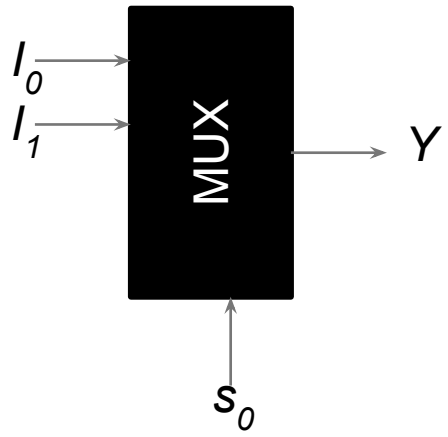


I_0	I_1 (B)	s_0 (A)	Y
0	B	0	I_0 (0)
0	B	1	I_1 (B)

- **a MUX is a universal circuit, i.e. it can implement NOT, OR, and AND gates**
  - implement an AND gate using a MUX: Y = A AND B
  - $Y = I_0\bar{s}_0 + I_1 s_0 = 0\bar{A} + BA = AB$

MUX, EX 5 B

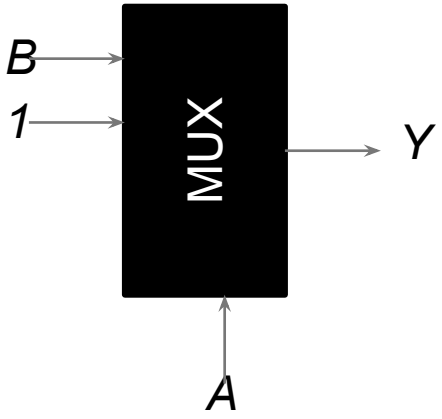
- **MUX, two inputs, one selection signal s, and one output**



- **a MUX is a universal circuit, i.e. it can implement NOT, OR, and AND gates**
  - implement an OR gate using a MUX

MUX, EX 5 B

- **MUX, two inputs, one selection signal s, and one output**

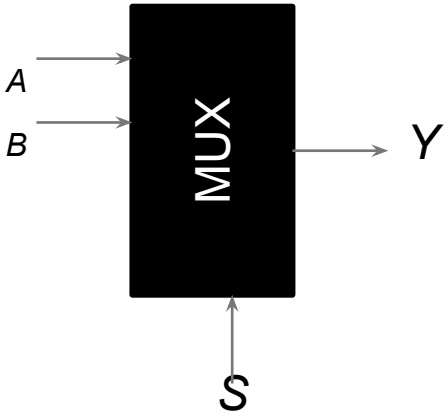


I_0 (B)	I_1	s_0 (A)	Y
B	1	0	I_0 (B)
B	1	1	I_1 (1)

- **a MUX is a universal circuit, i.e. it can implement NOT, OR, and AND gates**
  - implement an OR gate using a MUX: Y = A OR B
  - $Y = I_0\bar{s}_0 + I_1 s_0 = B\bar{A} + 1A = A + B\bar{A} = A + B$ [see Ex. 4(f)]

MUX, EX 5 C

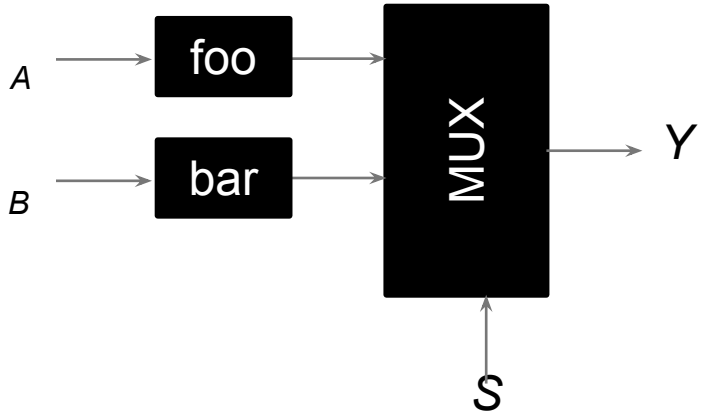
- Y = S ? foo(A) : bar(B)



- how do we implement the “if” above?

MUX, EX 5 C

- Y = S ? foo(A) : bar(B)

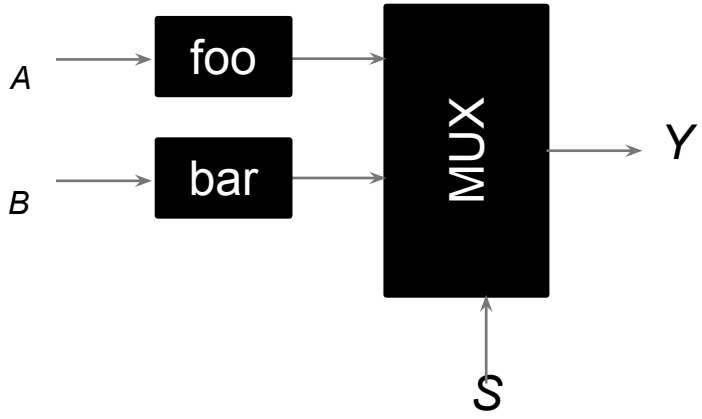


I_0 foo(X)	I_1 bar(Y)	S	Y
*	*	0	I_1
*	*	1	I_0

- what is the difference compared with a programming language?

MUX, EX 5 C

- **Y = S ? foo(A) : bar(B)**

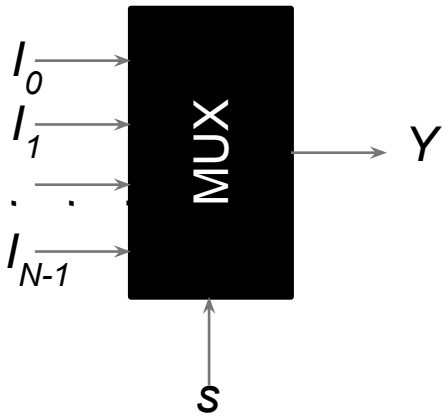


I_0 foo(X)	I_1 bar(Y)	S	Y
*	*	0	I_1
*	*	1	I_0

- **what is the difference compared with a programming language?**
  - regardless of the value of S, both foo(A) and bar(B) are executed
  - but at the output we see only one of the functions (the one selected by S)

MUX, EX 5 D

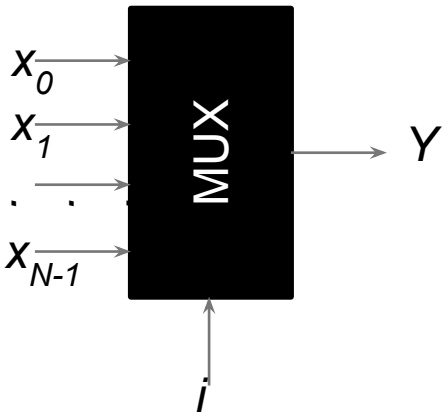
- we want to access an element x_i of a vector



- what can we place at the inputs?
- what is the signal s?

MUX, EX 5 D

- we want to access x_i

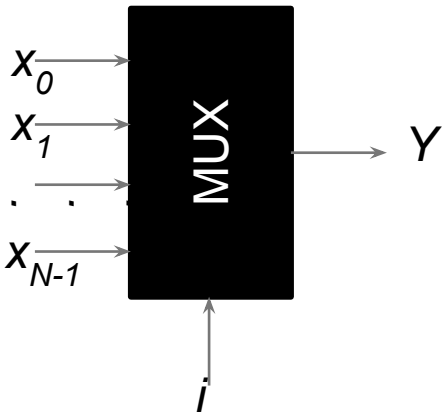


- what can we place at the inputs? we place the vector x
- what is the signal s? we use the index i

- what is the input size?
- what is the size of s?

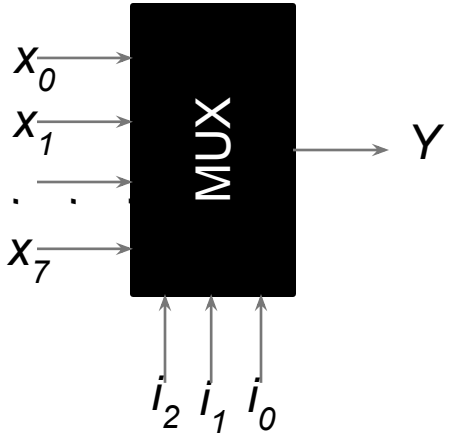
MUX, EX 5 D

- we want to access x_i



s_0 (i)	Y
000	I_0 (x_0)
001	I_1 (x_1)
010	I_2 (x_2)
011	I_3 (x_3)
100	I_4 (x_4)
101	I_5 (x_5)
110	I_6 (x_6)
111	I_7 (x_7)

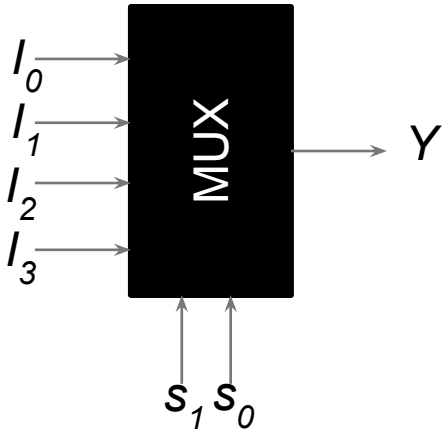
- what can we place at the inputs? we place the vector x
- what is the signal s? we use the index i



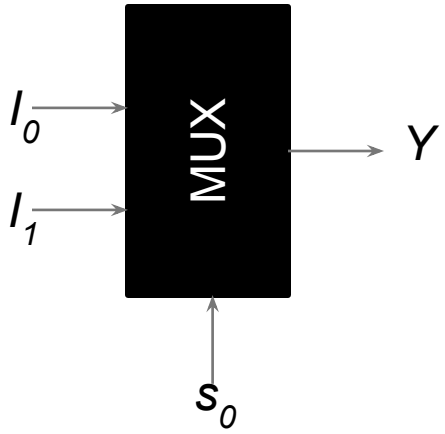
- what is the input size? N
- what is the size of s? ceil(log2 N)

MUX, EX 5 E

- **a MUX with 4 inputs**
  - we automatically know that the signal s has two bits

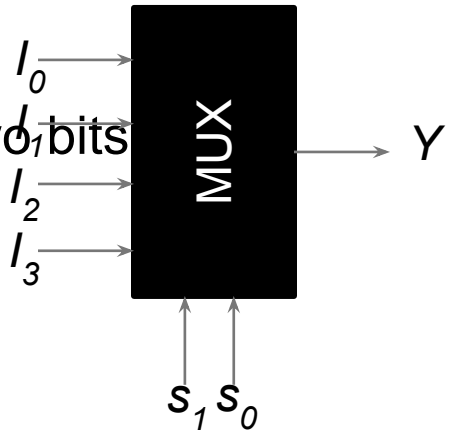


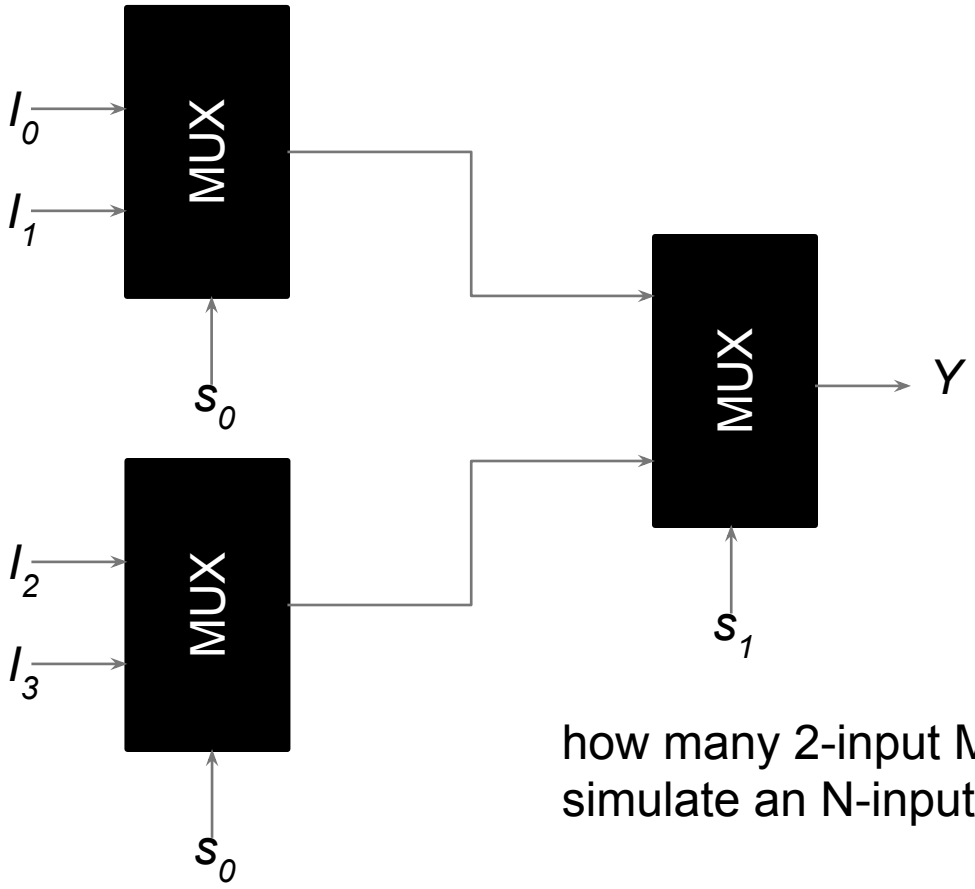
- **we have a 2-input MUX available**



MUX, EX 5 E

- **a MUX with 4 inputs**
  - we automatically know that the signal s has two bits

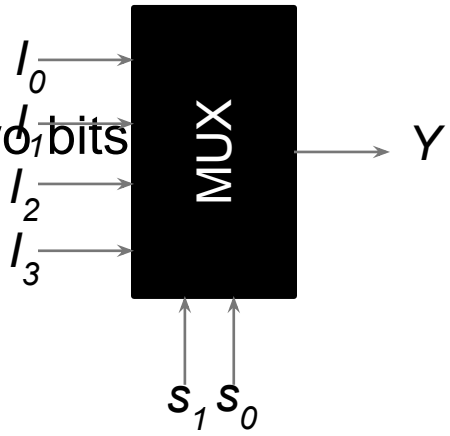


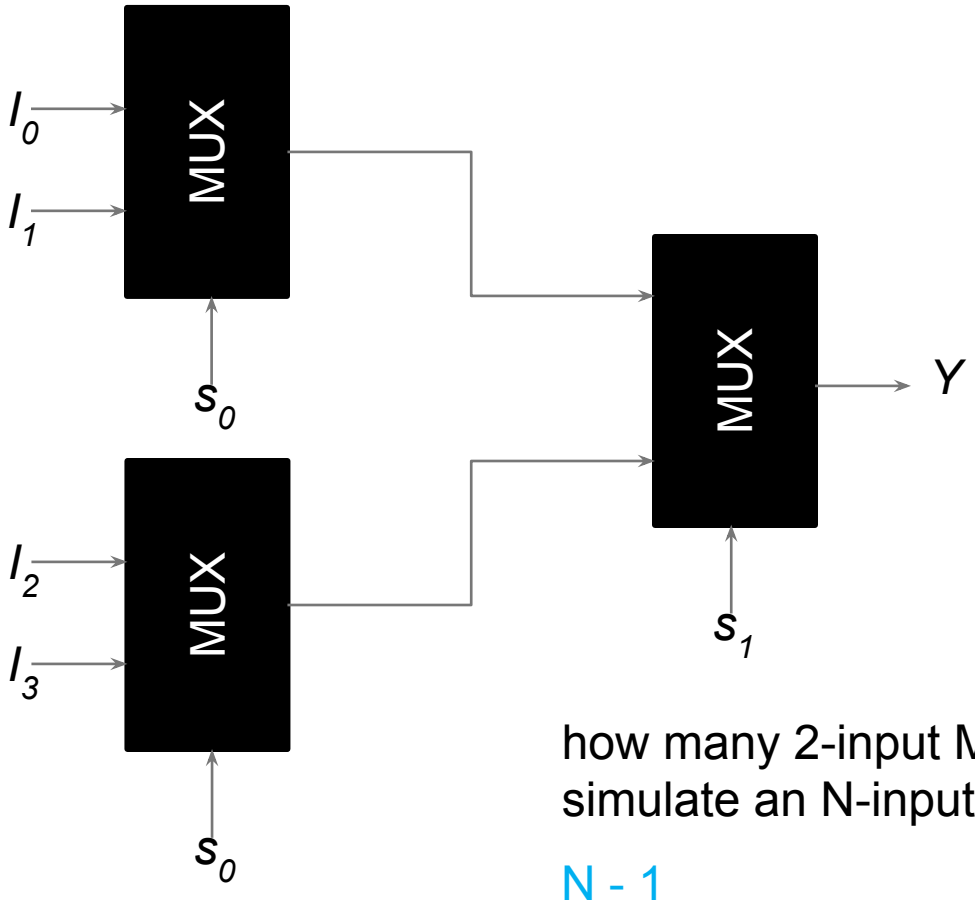


how many 2-input MUXes do we need to simulate an N-input MUX?

MUX, EX 5 E

- **a MUX with 4 inputs**
  - we automatically know that the signal s has two bits





how many 2-input MUXes do we need to simulate an N-input MUX?

N - 1

SHIFTS, EX 7 A

- **our number x is**



- **logical shift right by 2 >>**



equivalent to division by 2^2

- **arithmetic shift right by 2 $>>_a$**



equivalent to division by 2^2

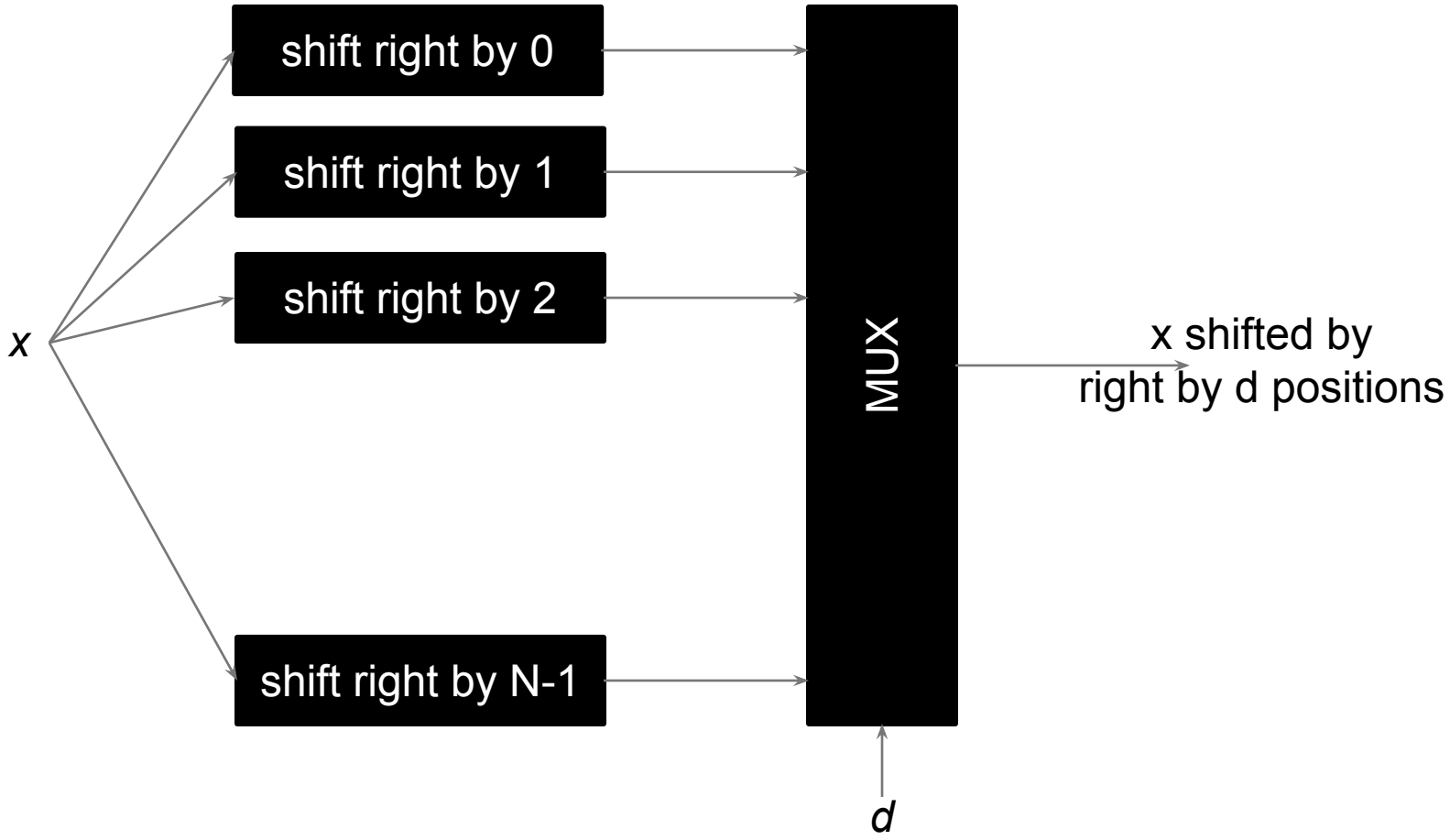
- **circular shift right by 2 $>>_c$**



All shifts can be defined for any numeric base, not only base 2. For example, in base 10 we have $754 <<_c 2 = 475$ for a circular left shift, and $754 << 2 = 400$ for a logical left shift (on 3 digits)

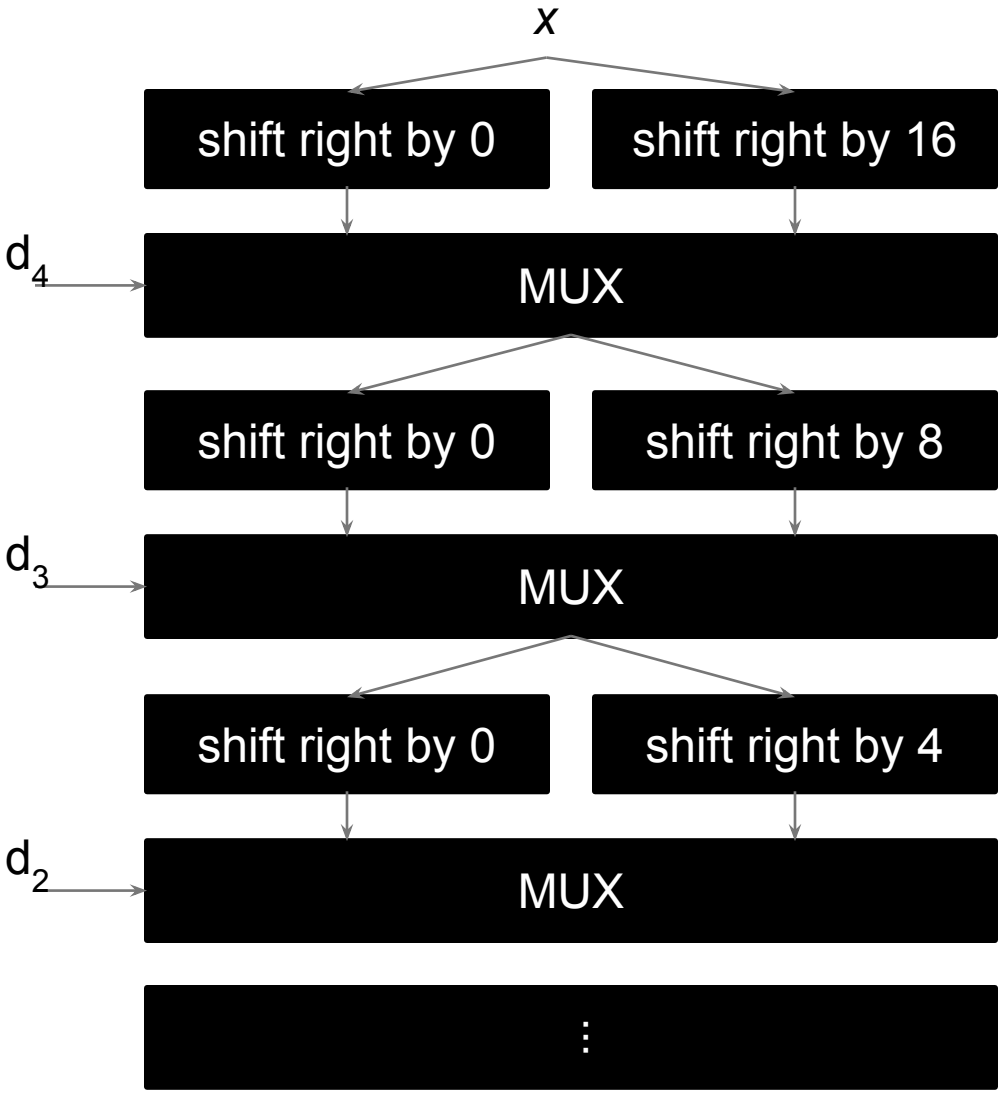
SHIFTS, EX 7 B

- **shift a number x to the right by d positions**



SHIFTS, EX 7 C

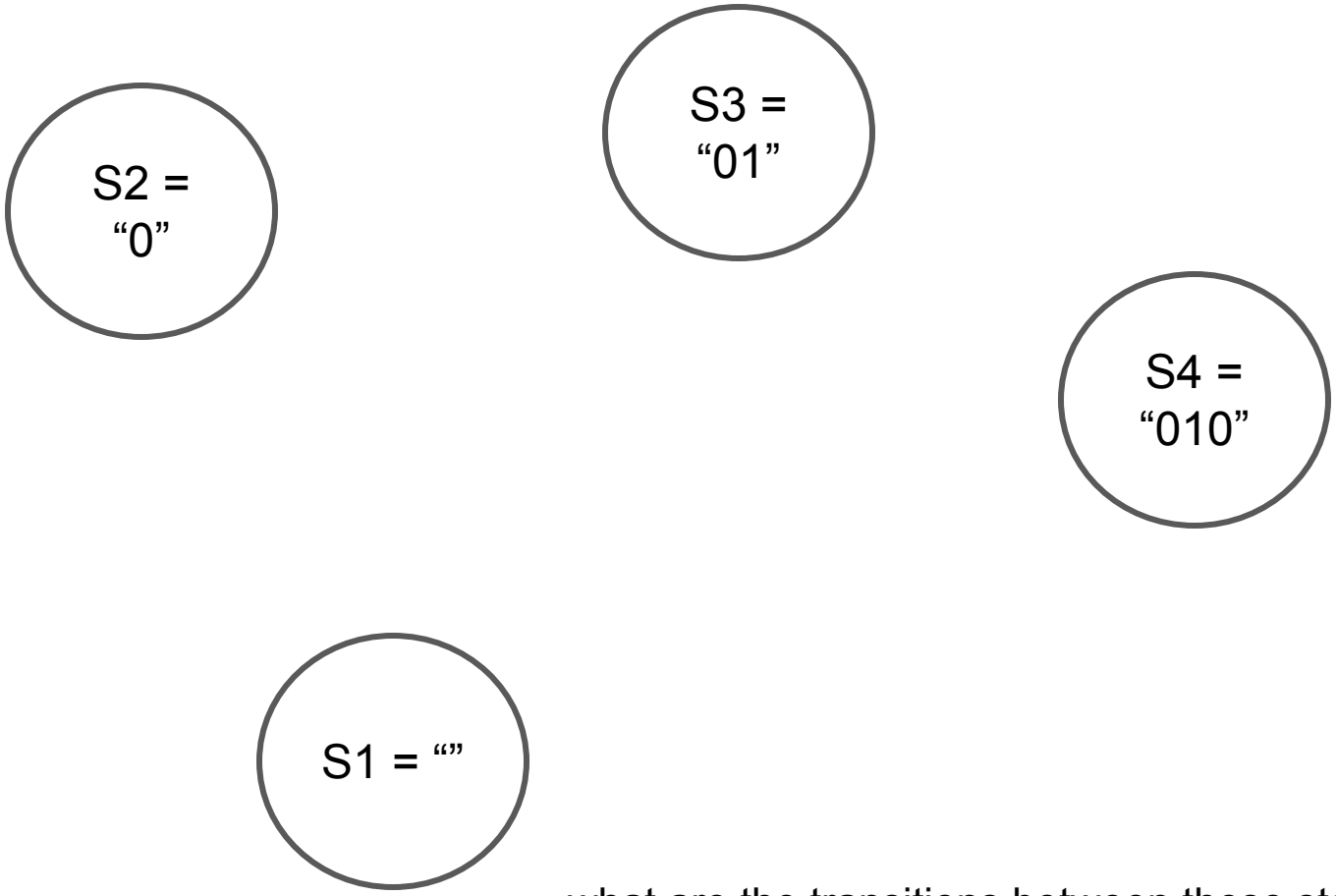
- suppose x is represented on 32 bits
- the shift amount d is represented on 5 bits: $d_4 d_3 d_2 d_1 d_0$



we need $\log_2 d$ MUXes

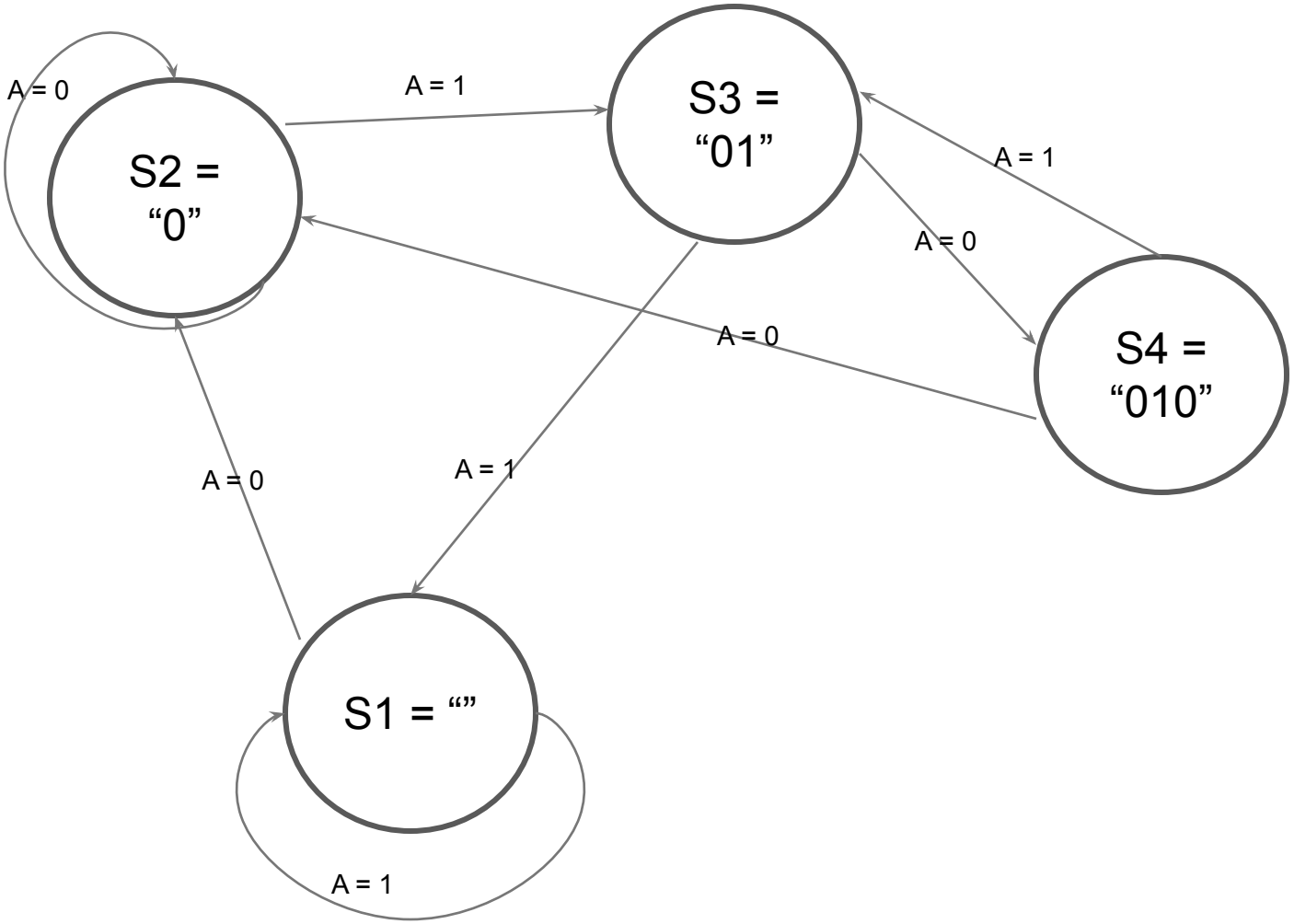
SEQUENTIAL, EX 10

- **we define 4 states**

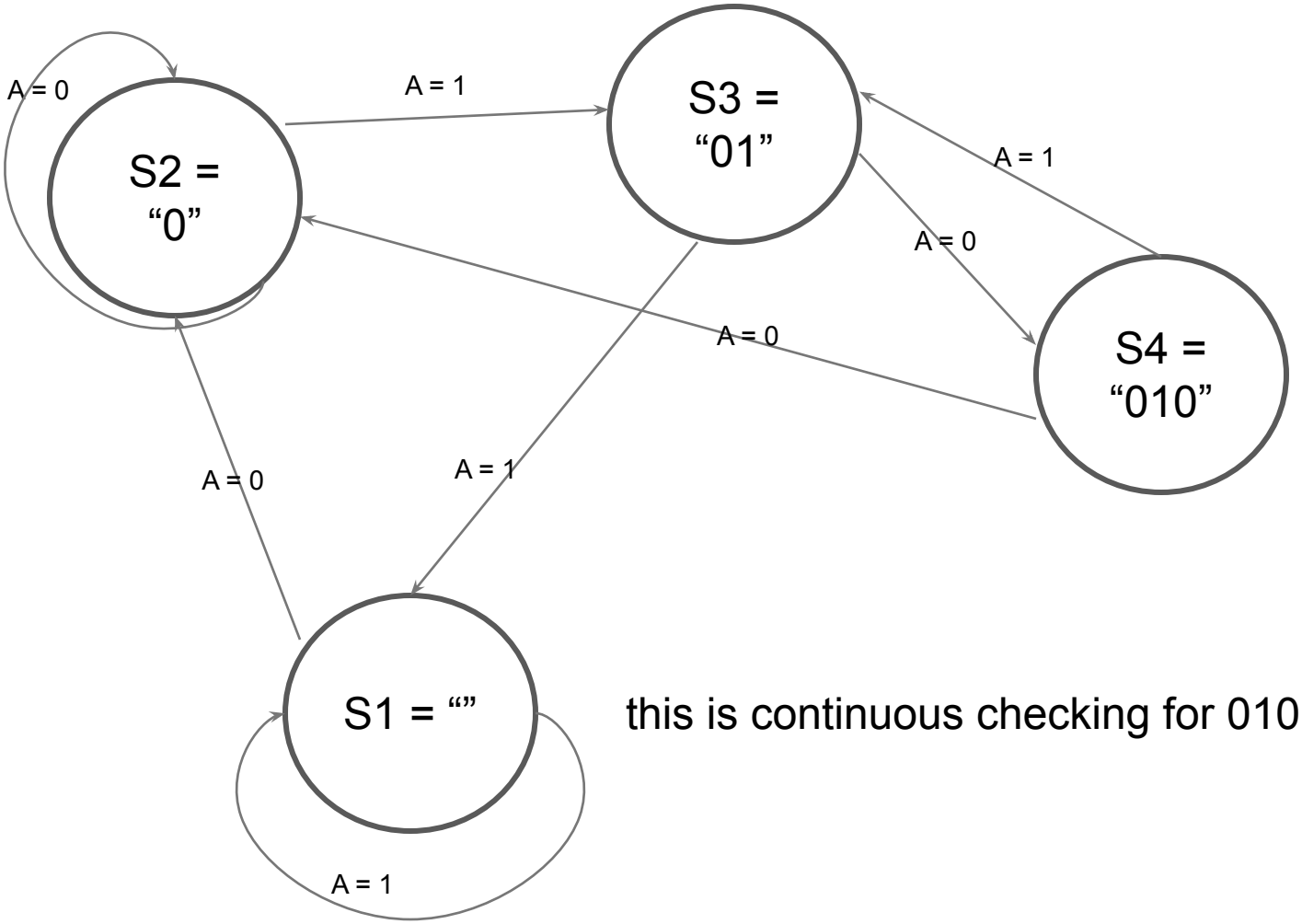


- what are the transitions between these states?

SEQUENTIAL, EX 10

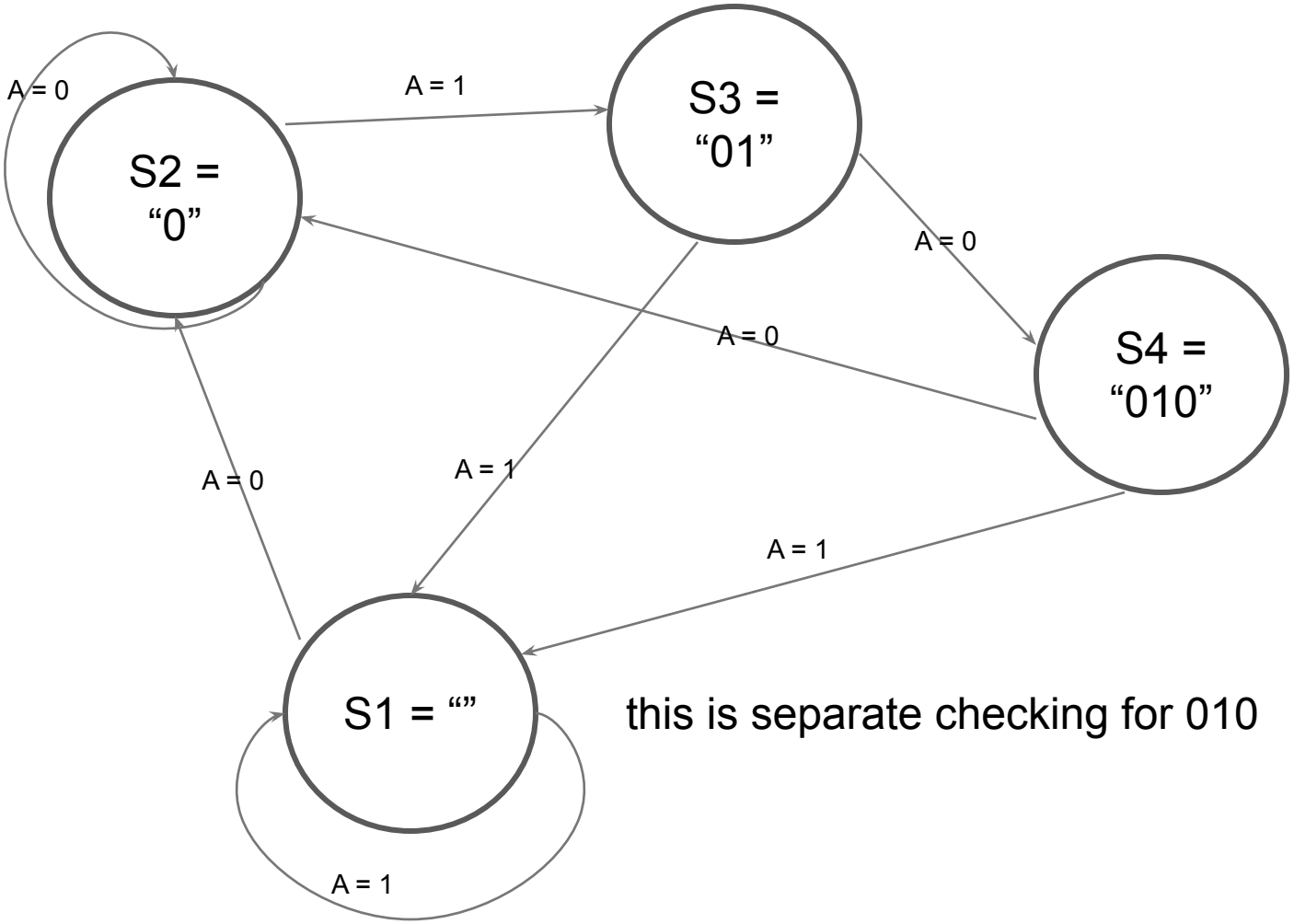


SEQUENTIAL, EX 10



this is continuous checking for 010

SEQUENTIAL, EX 10

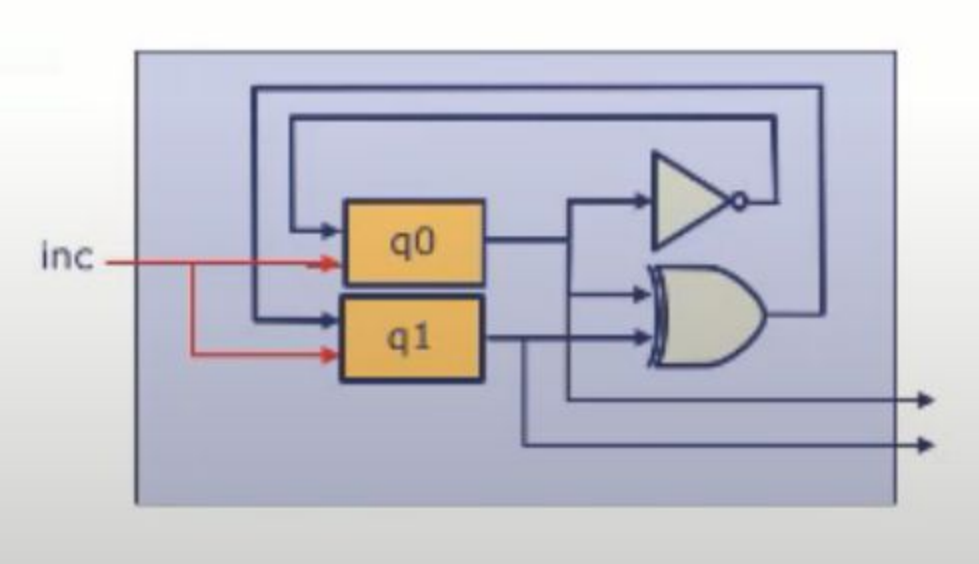


this is separate checking for 010

2-BIT COUNTER, EX 12

- $q_0^{(t+1)} = !\mathrm{INC} \times q_0^{(t)} + \mathrm{INC} \times !q_0^{(t)}$, $q_1^{(t+1)} = !\mathrm{INC} \times q_1^{(t)} + \mathrm{INC} \times (q_1^{(t)} \otimes q_0^{(t)})$

$q_1^{(t)}$	$q_0^{(t)}$	$q_1^{(t+1)}$	$q_0^{(t+1)}$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



- **here INC acts as the Enable signal**

2-BIT COUNTER, EX 12

- **Gray-code counter**

$q_2^{(t)}$	$q_1^{(t)}$	$q_0^{(t)}$	$q_2^{(t+1)}$	$q_1^{(t+1)}$	$q_0^{(t+1)}$
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

- $q_0^{(t+1)} = !q_2^{(t)}!q_1^{(t)}!q_0^{(t)} + !q_2^{(t)}!q_1^{(t)}q_0^{(t)} + q_2^{(t)}q_1^{(t)}!q_0^{(t)} + q_2^{(t)}q_1^{(t)}q_0^{(t)}$
  $= q_2^{(t)}q_1^{(t)} + !q_2^{(t)}!q_1^{(t)}$
- $q_1^{(t+1)} = \ldots$
- $q_2^{(t+1)} = \ldots$

GCD, EX 13

- **implement Euclid's algorithm for GCD using a sequential circuit.**
- **example: given a = 15, b = 6**
  - step 1: a = 9, b = 6 (a := a – b)
  - step 2: a = 3, b = 6 (a := a – b)
  - step 3: a = 6, b = 3 (a,b := b,a)
  - step 4: a = 3, b = 3 (a := a – b)
  - step 5: a = 0, b = 3
  - the answer is 3

- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

GCD, EX 13

- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

  - we have the variables a(t) and b(t)
  - according to the algorithm above, we have the state-evolution equations
    - we do something only if a(t) != 0

GCD, EX 13

- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

  - we have the variables a(t) and b(t)
  - according to the algorithm above, we have the state-evolution equations
    - we do something only if a(t) != 0
    - p = ($a^{(t)} > b^{(t)}$)
    - $a^{(t+1)}$ = ?
    - $b^{(t+1)}$ = ?

GCD, EX 13

- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

  - we have the variables a(t) and b(t)
  - according to the algorithm above, we have the state-evolution equations
    - we do something only if a(t) != 0
    - p = $(a^{(t)} > b^{(t)})$
    - $a^{(t+1)} = p \times (a^{(t)} - b^{(t)}) + !p \times b^{(t)}$
    - $b^{(t+1)} = p \times b^{(t)} + !p \times a^{(t)}$

GCD, EX 13

- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

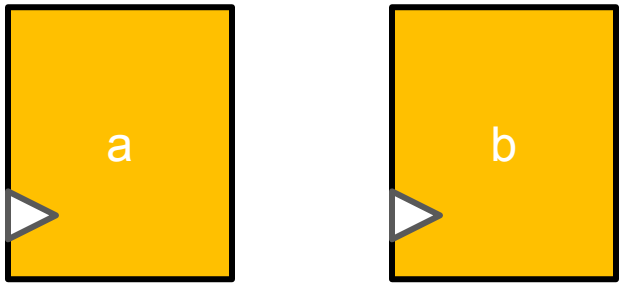
what kind of circuits are a and b?

GCD, EX 13

- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

when are these registers active?



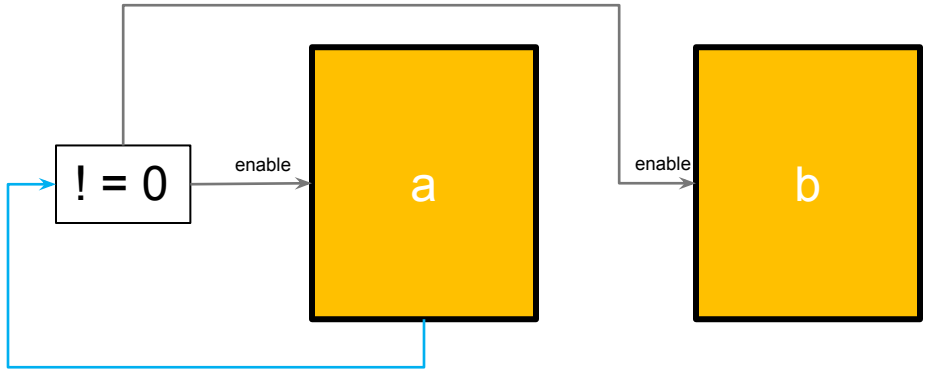
GCD, EX 13

- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

$p = (a^{(t)} > b^{(t)})$
$a^{(t+1)} = p \times (a^{(t)} - b^{(t)}) + !p \times b^{(t)}$
$b^{(t+1)} = p \times b^{(t)} + !p \times a^{(t)}$

how do a and b change?



GCD, EX 13

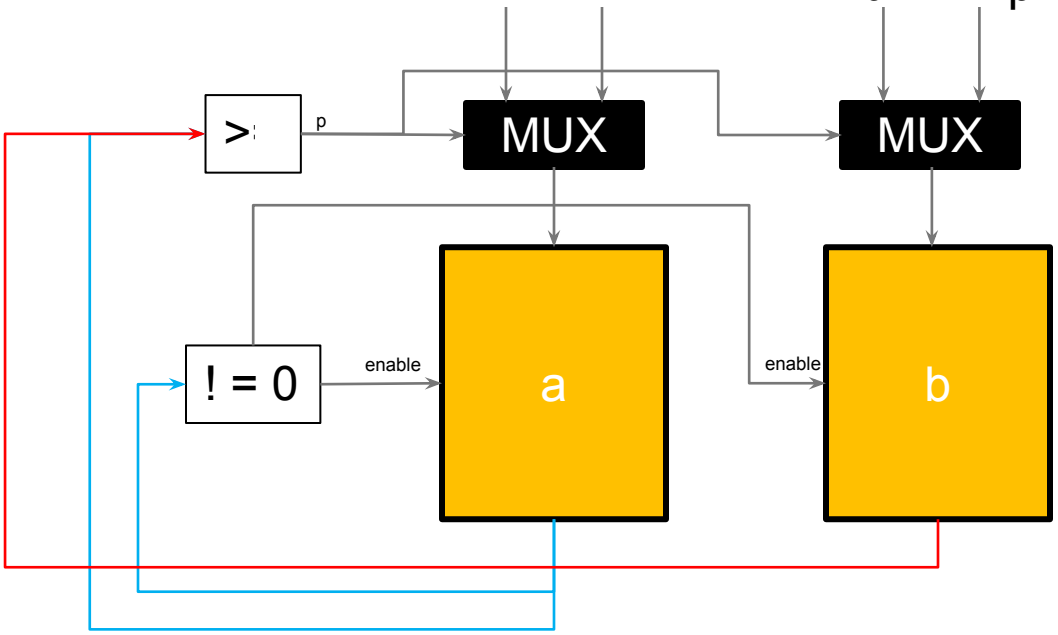
- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

$p = (a^{(t)} > b^{(t)})$

$a^{(t+1)} = p \times (a^{(t)} - b^{(t)}) + !p \times b^{(t)}$

$b^{(t+1)} = p \times b^{(t)} + !p \times a^{(t)}$



what are the inputs of MUX 1?

GCD, EX 13

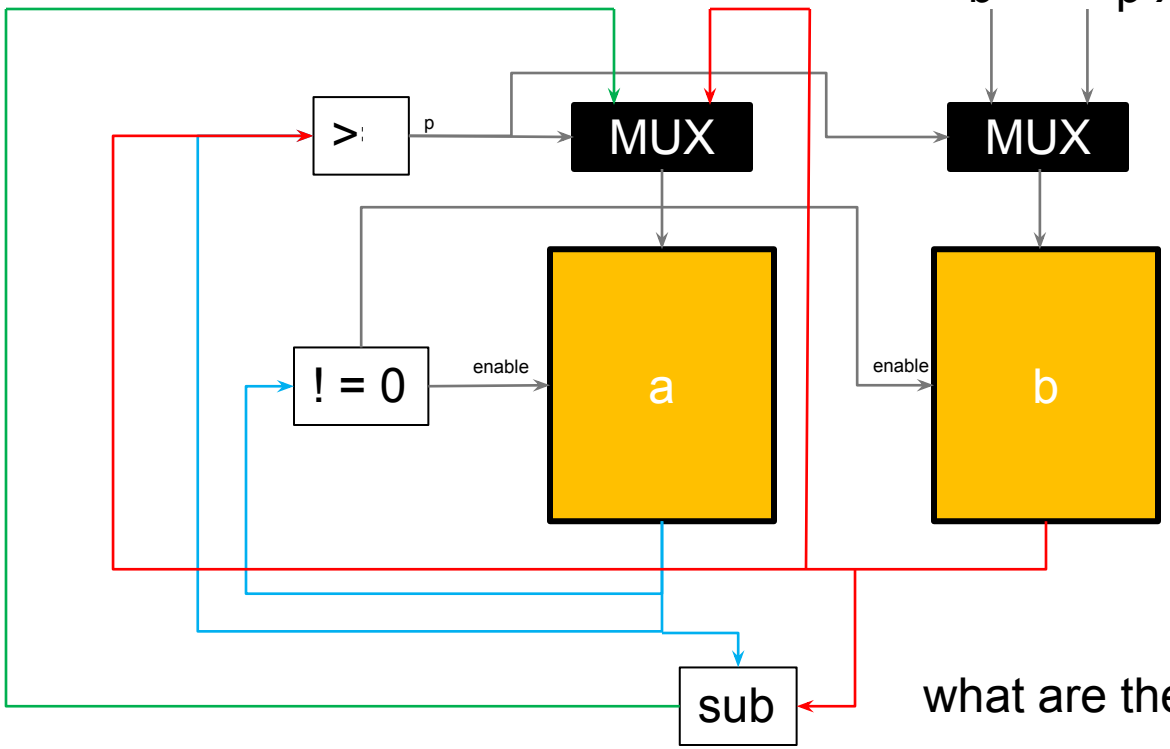
- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

$p = (a^{(t)} > b^{(t)})$

$a^{(t+1)} = p \times (a^{(t)} - b^{(t)}) + !p \times b^{(t)}$

$b^{(t+1)} = p \times b^{(t)} + !p \times a^{(t)}$



what are the inputs of MUX 2?

GCD, EX 13

- **the Python code:**

```
def gcd(a, b):
    if a == b: return b
    elif a > b: return gcd(a-b, b)
    else: return gcd(b, a)
```

$p = (a^{(t)} > b^{(t)})$

$a^{(t+1)} = p \times (a^{(t)} - b^{(t)}) + !p \times b^{(t)}$

$b^{(t+1)} = p \times b^{(t)} + !p \times a^{(t)}$

