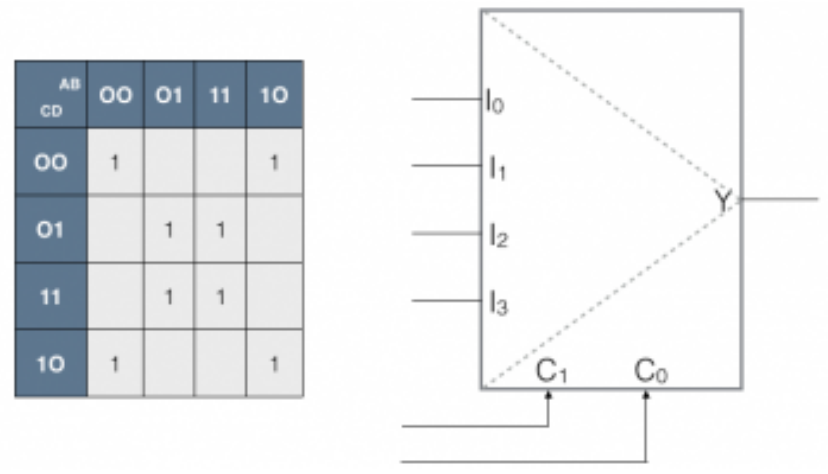
Kaitlyn Stumpf
February 21st, 2016
CSC 270, Spring
Dominique Thiebaut

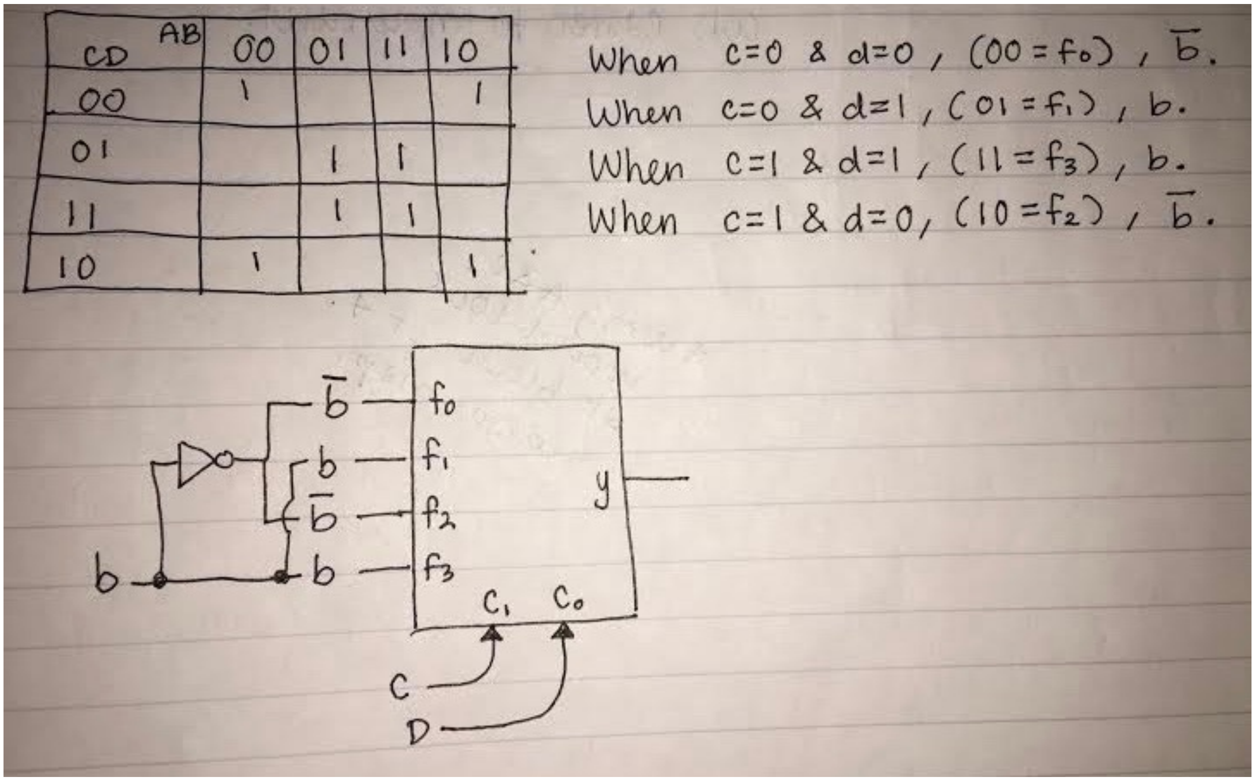
Homework #4
Problem 1

- *Using as few additional gates as possible, implement the function f(A, B, C, D) whose Karnaugh map is shown below, on the left, using the multiplexer shown on the right.*



CD \ AB	00	01	11	10
00	1			1
01		1	1	
11		1	1	
10	1			1

My work for part one of problem 1 is shown below. For each CD row I connected the C & D binary value to its matching f (i.e. 11 maps to f_3), and set the input to be the defining feature of that row of the Kernaugh map.

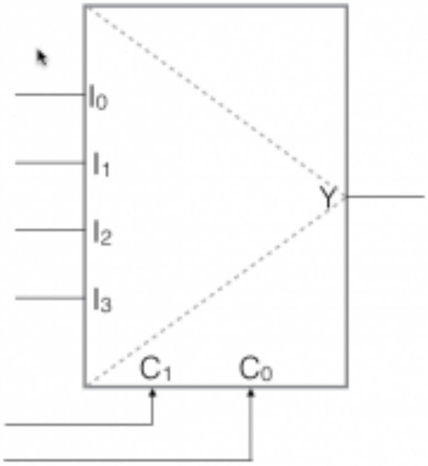


CD \ AB	00	01	11	10
00	1			1
01		1	1	
11		1	1	
10	1			1

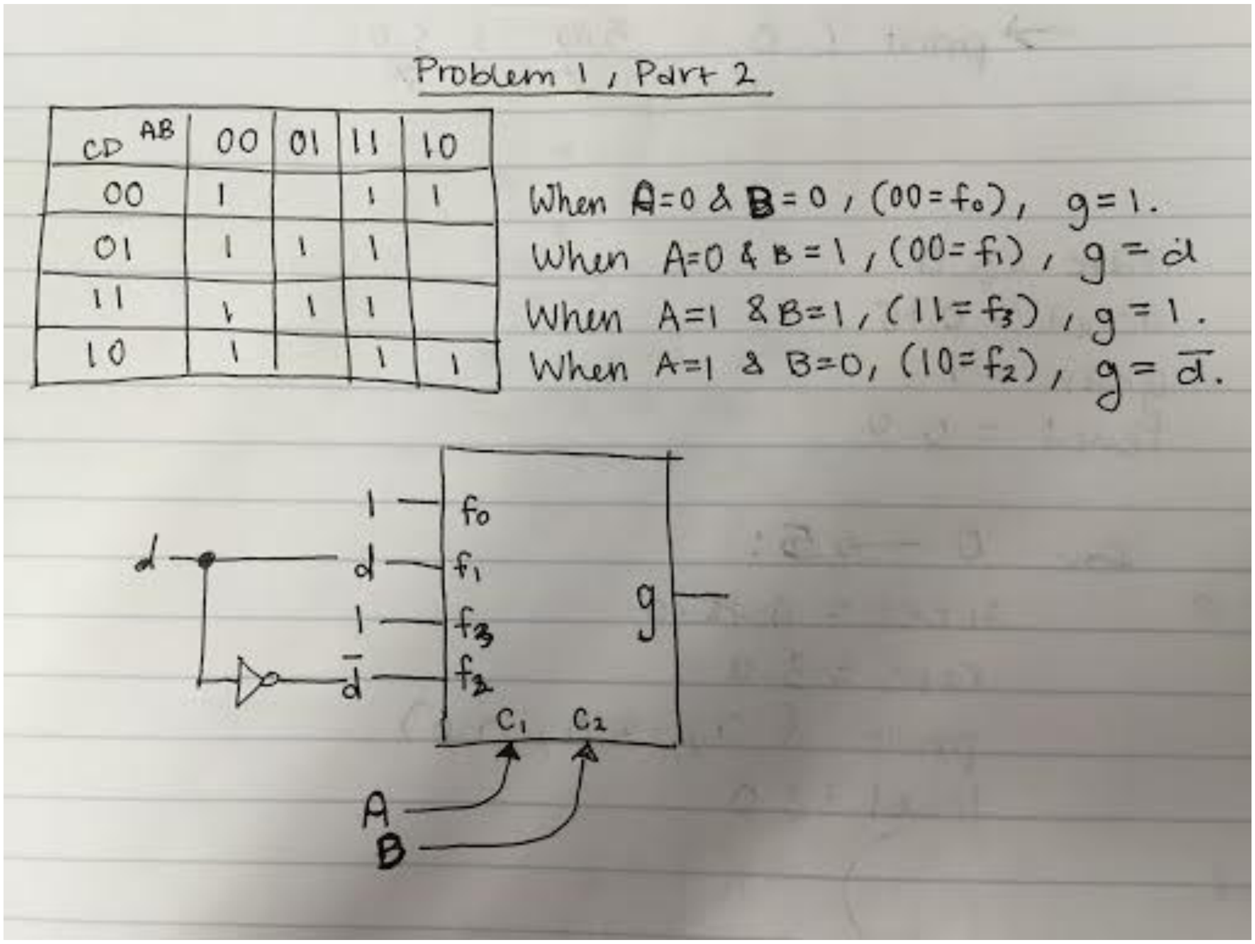
When c=0 & d=0, (00 = f_0), $\overline{b}$.
When c=0 & d=1, (01 = f_1), b.
When c=1 & d=1, (11 = f_3), b.
When c=1 & d=0, (10 = f_2), $\overline{b}$.

- *Using as few additional gates as possible, implement the function g(A, B, C, D) whose Karnaugh map is shown below, on the left, using the multiplexer shown on the right.*

CD \ AB	00	01	11	10
00	1		1	1
01	1	1	1	
11	1	1	1	
10	1		1	1



My work for part one of problem 1 is shown below. For each CD row I connected the C & D binary value to its matching f (i.e. 11 maps to f_3), and set the input to be the defining feature of that row of the Kernaugh map.



Problem 1, Part 2

CD \ AB	00	01	11	10
00	1		1	1
01	1	1	1	
11	1	1	1	
10	1		1	1

When A=0 & B=0, (00=f_0), g=1.
When A=0 & B=1, (00=f_1), g=d
When A=1 & B=1, (11=f_3), g=1.
When A=1 & B=0, (10=f_2), g=$\bar{d}$.

d
1
d
1
$\bar{d}$
f_0
f_1
f_3
f_2
g
C_1
C_2
A
B

Problem 2

Implement your two answers for Problem 1 in Python, and shown that your design is correct (although it might not be as minimal as required).

Your Python code should have the following features:

- *It must contain a header with your name(s), and a description of what it does*
- *It must contain a mux() function with 6 inputs (A, B, C, D, C1, C0), and returning one value (Y).*
- *It must generate the truth table for f and for g.*

Include the code and a copy of the output in your pdf. Please use a non-proportional font when displaying code and output: it makes it much easier for the reader to read and understand code quickly.

CODE:

```
# Kaitlyn Stumpf
# Feb 23, 2016
# CSC270, Thiebaut
# Hw4, Python/Java Ques.

# This program contains a mux() func w/ 6 inputs (F0, F1, F2, F3, C1, C0)
# that returns one value (Y).
# It prints out truth tables for my two answers for Prob 1,
# by calling the mux() function w/ the values specific to the problem.

def mux(F0, F1, F2, F3, C1, C0):
      '''
      Gets F0, F1, F2, F3 input values as well as C1 and C0.
      Combines C1 & C0 into a binary string & then into a base ten value.
      Uses base ten value to pick which of the multiplexer inputs, f0, f1,
f2, or f3,
      is needed, and returns this vale.

      '''
      muxInputs = [F0, F1, F2, F3]
      Y = 0
      # Assuming inputs to C1 & C0 are ints equal to either 0 or 1, turns
into binary string.
      binaryStr = str(C1) + str(C0)
      # Then converts binary string into base ten int.
      baseTenVal = int(binaryStr, 2)

      return muxInputs[baseTenVal]


def truthTable():
      ''' truthTable() references mux() in order to print a truth table
          for the multiplexers f & g.
      '''

      print
      print( " For the following multiplexer: ")
      print( " F0 = B', F1 = B, F2 = B', F3 = B ")
      print( " C1 = C, C0 = D ")
      print( " Given this info, I generated the following truth table.")
      print
      print( "  A   B   C   D  |  f  " )
      print( "-----------------+-----" )
      for A in [0, 1]:
            for B in [0, 1]:
                  BINV = 0
                  if B == 0:
                        BINV = 1
                  for C in [0, 1]:
                        for D in [0, 1]:
                              f = mux(BINV, B, BINV, B, C, D)
                              print("  {0}   {1}   {2}   {3}  |  {4}  
".format(A, B, C, D, f))


      print
      print
        print( " For the following multiplexer: ")
        print( " F0 = 1, F1 = D, F2 = D’, F3 = 1 ")
        print( " C1 = A, C0 = B ")
        print( " Given this info, I generated the following truth table.")
        print
        print( "  A   B   C   D  |  g  " )
        print( "-----------------+-----" )
        for A in [0, 1]:
                for B in [0, 1]:
                        for C in [0, 1]:
                                for D in [0, 1]:
                              DINV = 0
                              if D == 0:
                                    DINV = 1
                                        g = mux(1, D, DINV, 1, A, B)
                                        print("  {0}   {1}   {2}   {3}  |  
{4}  ".format(A, B, C, D, g))

def main():
      ''' Calls truthTable in order to print the truth table.'''
      truthTable()

main()
```

OUTPUT:

```
Kaitlyns-MacBook-Pro:hw4 kaitlynstumpf$ python stumpf_hw4.py

 For the following multiplexer:
 F0 = B', F1 = B, F2 = B', F3 = B
 C1 = C, C0 = D
 Given this info, I generated the following truth table.

  A   B   C   D  |  f
-----------------+-----
  0   0   0   0  |  1
  0   0   0   1  |  0
  0   0   1   0  |  1
  0   0   1   1  |  0
  0   1   0   0  |  0
  0   1   0   1  |  1
  0   1   1   0  |  0
  0   1   1   1  |  1
  1   0   0   0  |  1
  1   0   0   1  |  0
  1   0   1   0  |  1
  1   0   1   1  |  0
  1   1   0   0  |  0
  1   1   0   1  |  1
  1   1   1   0  |  0
  1   1   1   1  |  1
```

For the following multiplexer:
F0 = 1, F1 = D, F2 = D’, F3 = 1
C1 = A, C0 = B
Given this info, I generated the following truth table.

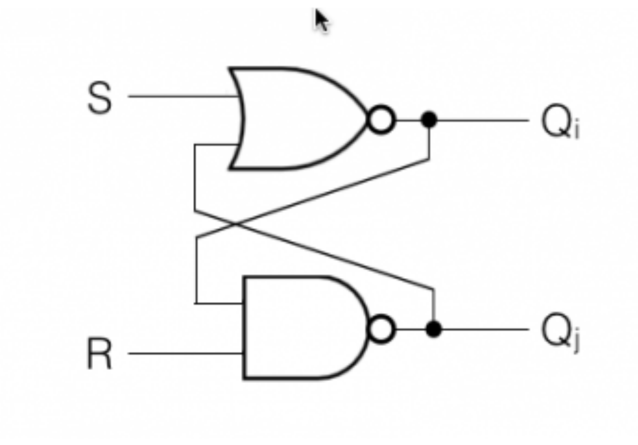
```
  A   B   C   D  |  g
-----------------+-----
  0   0   0   0  |  1
  0   0   0   1  |  1
  0   0   1   0  |  1
  0   0   1   1  |  1
  0   1   0   0  |  0
  0   1   0   1  |  1
  0   1   1   0  |  0
  0   1   1   1  |  1
  1   0   0   0  |  1
  1   0   0   1  |  0
  1   0   1   0  |  1
  1   0   1   1  |  0
  1   1   0   0  |  1
  1   1   0   1  |  1
  1   1   1   0  |  1
  1   1   1   1  |  1
```

Problem 3

Is the circuit below a latch? Why or why not?

If you find out that it is a latch, you should be able to indicate:

- *What are the inputs that are "passing" and putting the latch in a stable state.*
- *What input (S or R) must be activated (and how) to force a 1 on Qi*
- *What input (S or R) must be activated (and how) to force a 0 on Qi*



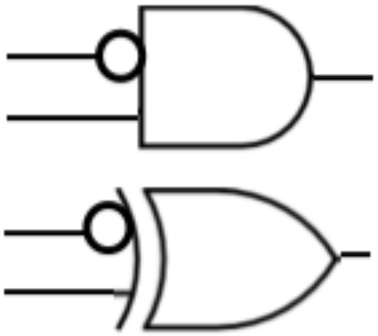
To solve this problem, I first tested if the circuit above was a latch. I did this test by using a state table to test if changes in S and R caused Q_i and Q_j to change in a predictable way. If they changed in a predictable way, (i.e. whenever S and R switched values, Q_i and Q_j switched values as well), this would indicate that the values of S and R were being stored and remembered. My state table is found below.

S	R	Q_i	Q_j	Is Circuit Consistent?
0	1	1	0	Yes
0	1	1	0	Yes
1	0	0	1	Yes
1	0	0	1	Yes
1	0	0	1	Yes
0	1	0	0	No

I have used the state table above to prove by contradiction that this circuit is inconsistent, and therefore not a latch. It does not always properly switch off the values saved in Q_i and Q_j when S and R switch values. When the latch goes from S = 1, R = 0 to S = 0, R = 1, rather than Q_i and Q_j switching values (and thus properly remembering which value was passed), it accidentally sets both Q_i and Q_j to zero.

Therefore, the above circuit is not a latch.

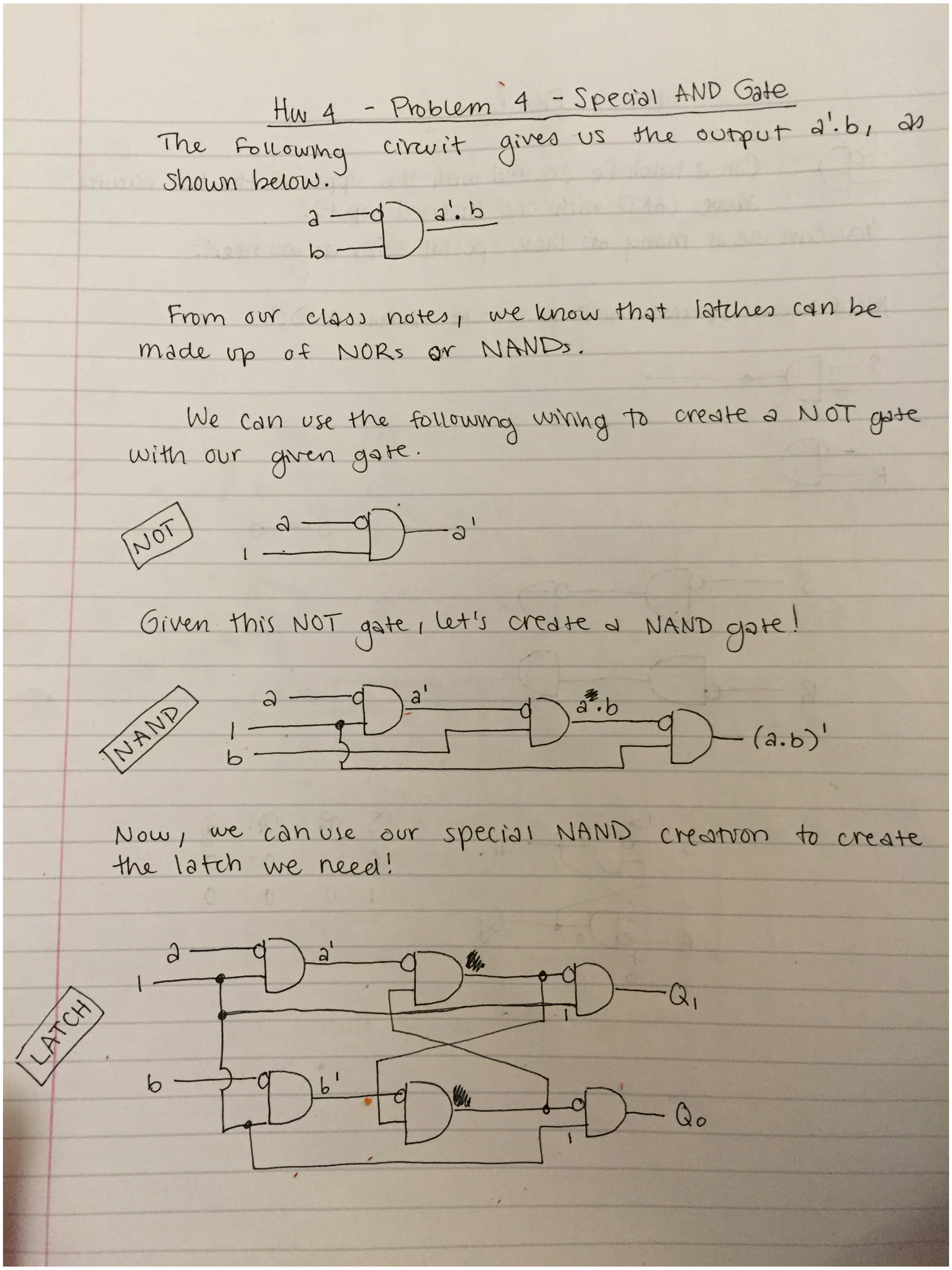
Problem 4



- *Can a latch be created with the upper of the two circuits above (AND with one inverted input)? You can use as many of these special ANDs as you need.*
- *Same question about the lower circuit (XOR with one inverted input)? You can use as many of these special XORs as you need.*

- *Please explain carefully why it is possible to create a latch, or why it is not possible to create a latch with each one the circuits.*

I was confused about how to go about this question, and so I asked Karen Diaz for help! She guided me in how to think about the first part.



Hw 4 - Problem 4 - Special AND Gate

The following circuit gives us the output $a' \cdot b$, as shown below.

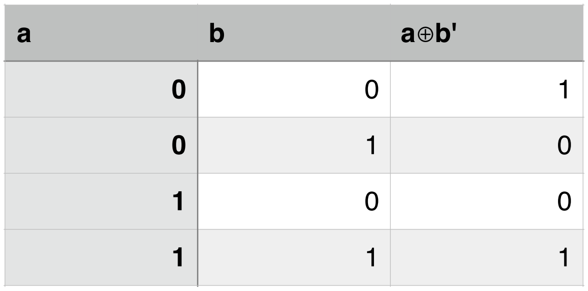
From our class notes, we know that latches can be made up of NORs or NANDs.

We can use the following wiring to create a NOT gate with our given gate.

Given this NOT gate, let's create a NAND gate!

Now, we can use our special NAND creation to create the latch we need!

The special XOR gate with one inverted input, however, cannot be a latch. This is because, as proven in homework 2, the special XOR gate with one inverted input is not a universal gate. As shown by the truth table below, there is no exclusivity in the output of the special gate. This means that we cannot isolate a single a or b value and use this to build a universal gate.



a	b	a⊕b'
0	0	1
0	1	0
1	0	0
1	1	1

Because we cannot use this special gate as a universal gate, it follows that it would not be possible to build either a NAND or a NOR gate using the special gate. Therefore, it would not be possible to build a latch using the special gate.