**IFT 510: Project 2**

Mary Tess Joseph

**IFT 510 Principles of Computer Information and Technology**

Edmund Lorenzen

28 April 2021

**1.**The Batcher network is built by connecting larger bitonic sorters in order, starting with a 2 \* 2

bitonic sorter. As a result, connecting to the input long size n \* n bitonic sorter when rearranging the

array correctly is necessary. The following functions are used to programmatically enforce these.  
sorter\_base (): 2 \* 2 bitonic sorter

sorter (): n \* n bitonic sorter

batcher (): batcher network.  
The Banyan network's basic structure is a sorter with two inputs that sorts by focusing on the defined

element of the bit string. When changing which bit to concentrate on and sort, this sorter is linked. It's

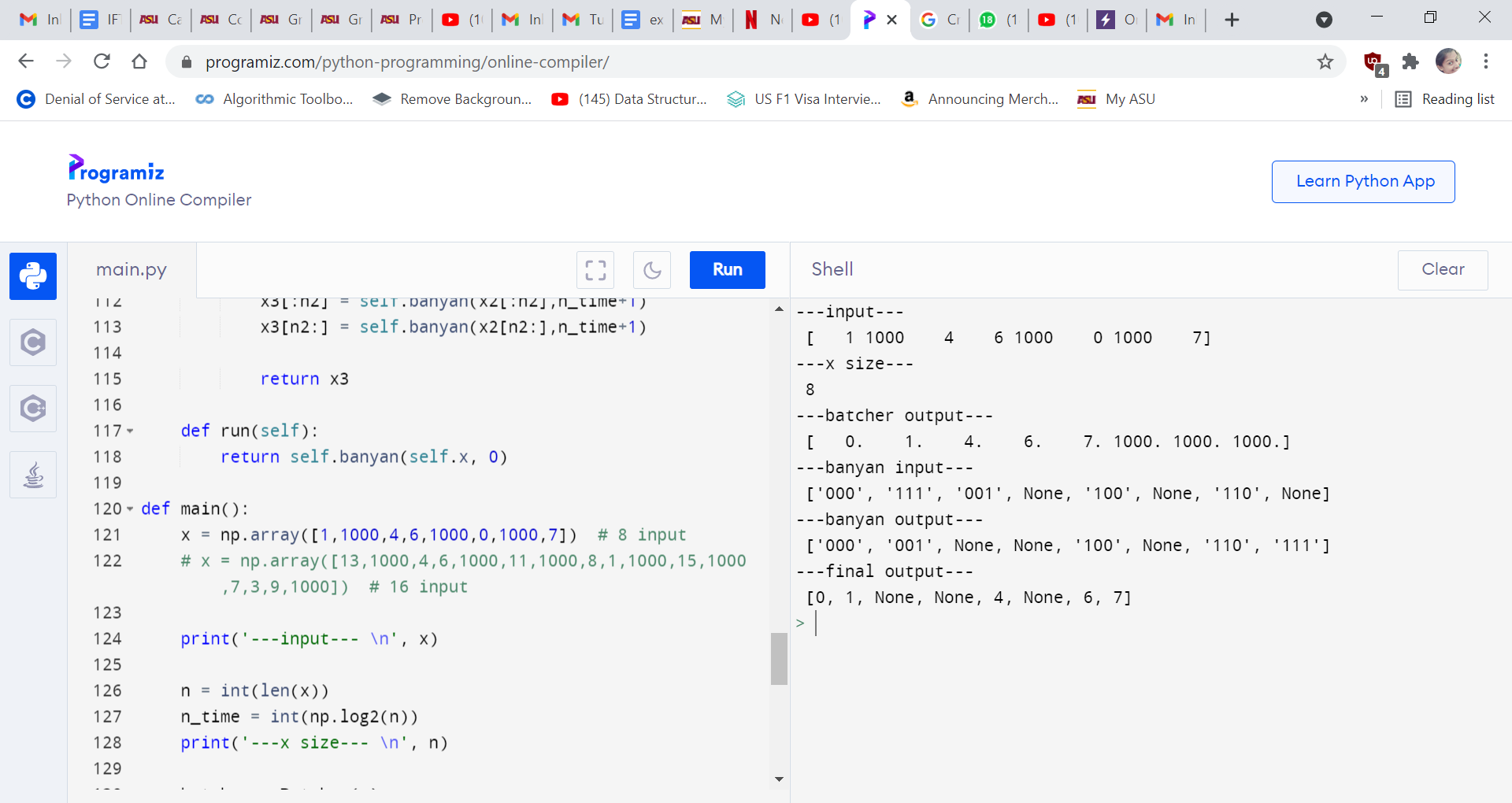
a Banyan system. The link method is the same as the n \* n bitonic sorter, so it can be defined in the same

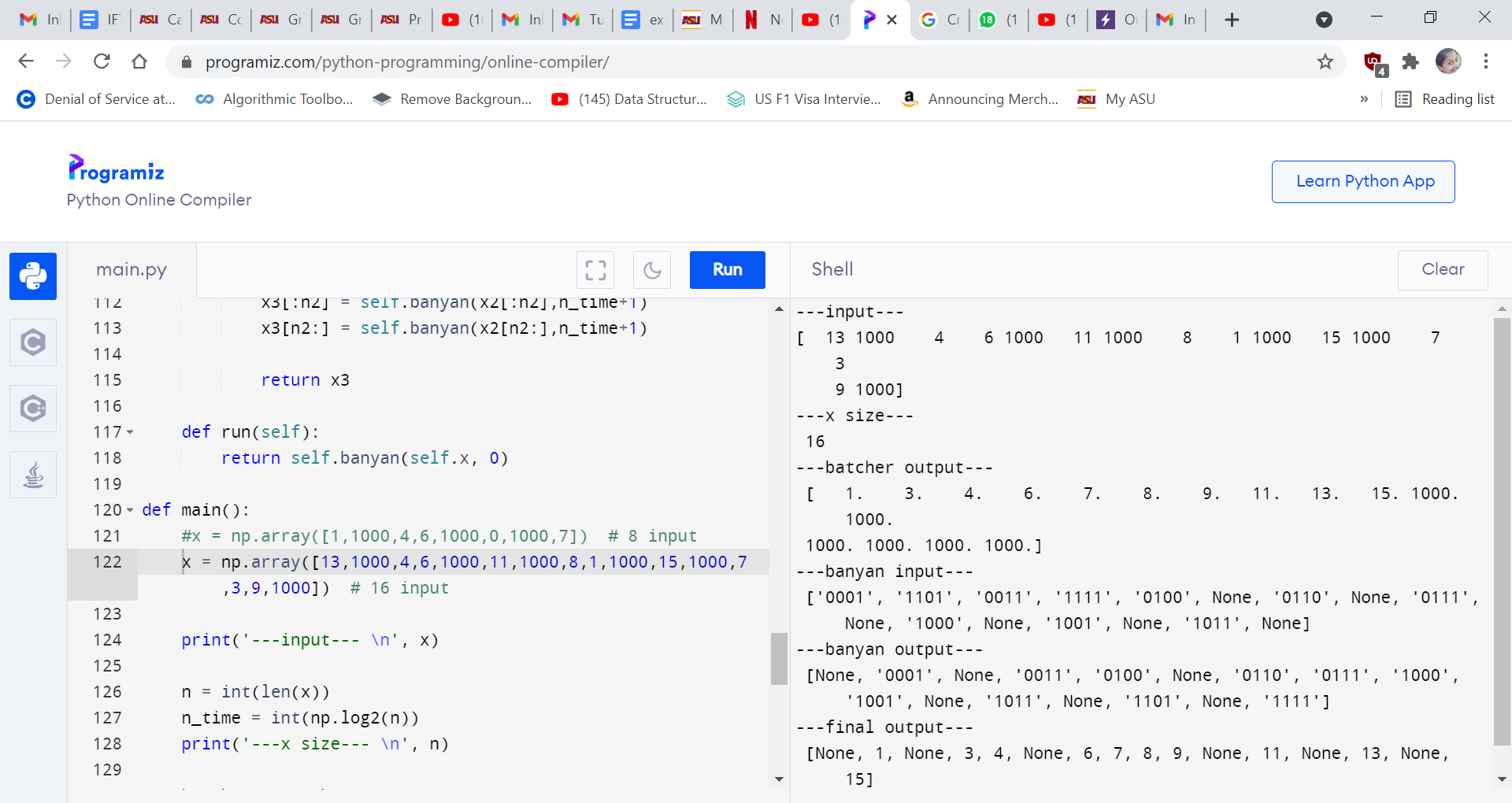
way recursively. After that, you should decide how much of each stage is of interest to you. These are

mentioned in the following order in the program. The role of is in charge of putting it into action.  
banyan\_base (): 2 \* 2 sorter

banyan (): banyan network.

**Input – Output**

8 input 8 output  


16 input 16 output  


**2.**  
import numpy as np

class Batcher:

def \_\_init\_\_(self, x):

self.x = x

self.n = int(len(x))

def sorter\_base(self, x):

if x[0] >= x[1]:

return x[::-1]

else:

return x

def sorter(self, x):

n = int(len(x))

n2 = int(n/2)

if n == 2:

return self.sorter\_base(x)

else:

x1 = np.zeros(n)

for i in range(n2):

x1[2\*i:2\*i+2] = self.sorter\_base(x[2\*i:2\*i+2])

x2 = np.zeros(n)

for i in range(n):

if i<n2 and i%2 == 1:

x2[i] = x1[n2+i-1]

elif i>=n2 and i%2 == 0:

x2[i] = x1[i-n2+1]

else:

x2[i] = x1[i]

x3 = np.zeros(n)

x3[:n2] = self.sorter(x2[:n2])

x3[n2:] = self.sorter(x2[n2:])

return x3

# n batcher network

def run(self):

data = self.x

k = 2

# bitnic sorter

while k <= self.n:

p = int(self.n/k)

data1 = np.zeros(self.n)

for i in range(p):

for j in range(k):

if j<k/2:

data1[2\*j+i\*k] = data[j+i\*k]

else:

data1[2\*j-k+1+i\*k] = data[j+i\*k]

data2 = np.zeros(self.n)

for i in range(p):

s = self.sorter(data1[k\*i:k\*i+k])

if i%2 == 1:

s = s[::-1]

data2[k\*i:k\*i+k] = s

k = k\*2

data = data2

return data

class Banyan:

def \_\_init\_\_(self, x):

self.x = x

def banyan\_base(self, x, n\_time):

if x[0] is not None:

if x[0][n\_time] == '1':

return x[::-1]

elif x[1] is not None:

if x[1][n\_time] == '0':

return x[::-1]

return x

def banyan(self, x, n\_time):

n = int(len(x))

n2 = int(n/2)

if n == 2:

return self.banyan\_base(x,n\_time)

else:

x1 = [''] \* n

for i in range(n2):

x1[2\*i:2\*i+2] = self.banyan\_base(x[2\*i:2\*i+2],n\_time)

x2 = [''] \* n

for i in range(n):

if i<n2 and i%2 == 1:

x2[i] = x1[n2+i-1]

elif i>=n2 and i%2 == 0:

x2[i] = x1[i-n2+1]

else:

x2[i] = x1[i]

x3 = [''] \* n

x3[:n2] = self.banyan(x2[:n2],n\_time+1)

x3[n2:] = self.banyan(x2[n2:],n\_time+1)

return x3

def run(self):

return self.banyan(self.x, 0)

def main():

#x = np.array([1,0,3,2,5,4,7,6]) # 8 input

x = np.array([1,0,3,2,5,4,7,6,9,8,11,10,13,12,15,14]) # 16 input

print('---input--- \n', x)

n = int(len(x))

n\_time = int(np.log2(n))

print('---x size--- \n', n)

batcher = Batcher(x)

x1 = batcher.run()

print('---batcher output--- \n', x1)

x2 = np.zeros(n)

for i in range(n):

if i<n/2:

x2[2\*i] = x1[i]

else:

x2[2\*i-n+1] = x1[i]

x3 = [None]\*n

for i in range(n):

if x2[i] < n:

x3[i] = format(int(x2[i]), '0{}b'.format(n\_time))

print('---banyan input--- \n', x3)

banyan = Banyan(x3)

x4 = banyan.run()

print('---banyan output--- \n', x4)

x5 = [None] \* n

for i in range(n):

if x4[i] != None:

x5[i] = int(x4[i], 2)

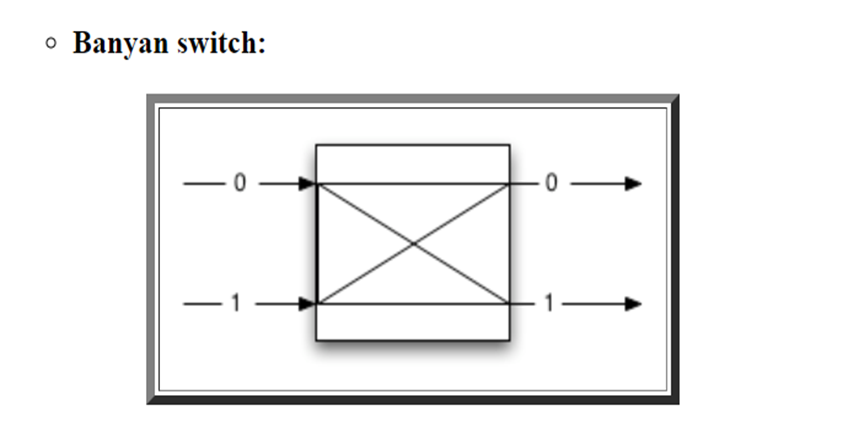
print('---final output--- \n', x5)

if \_\_name\_\_ == '\_\_main\_\_':

main()

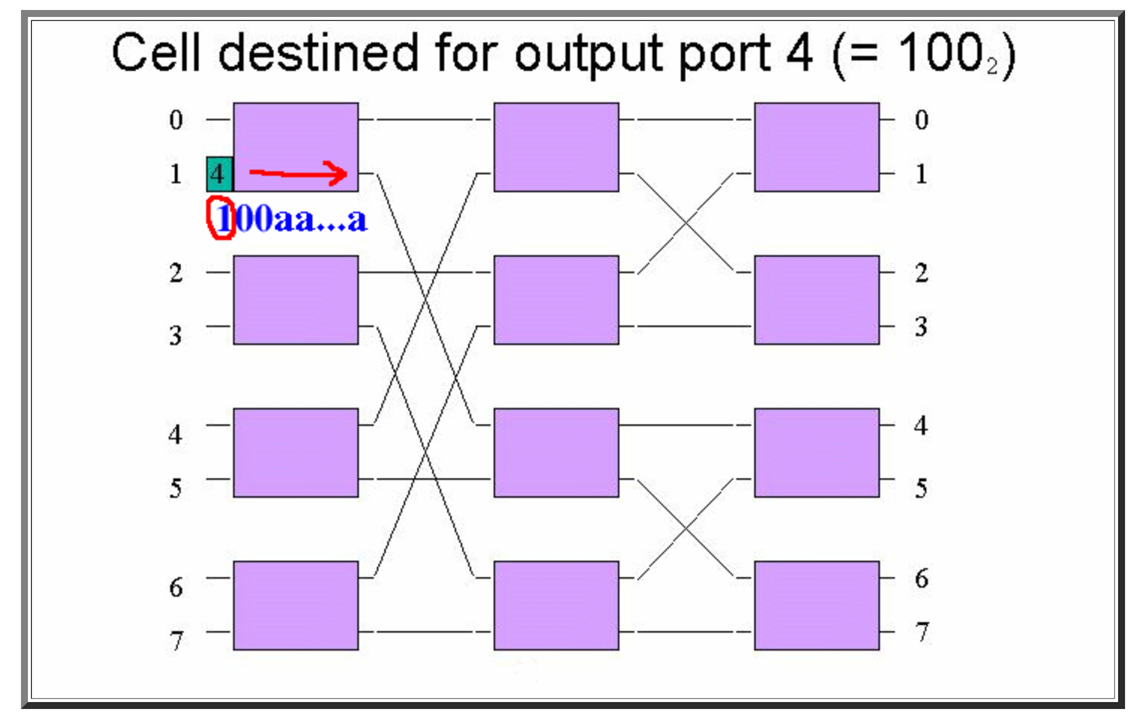
**3.  
The Banyan Switch**

* A 2x2 Banyan switch has two inputs and two outputs

-numbered 0 and 1.  
  
****

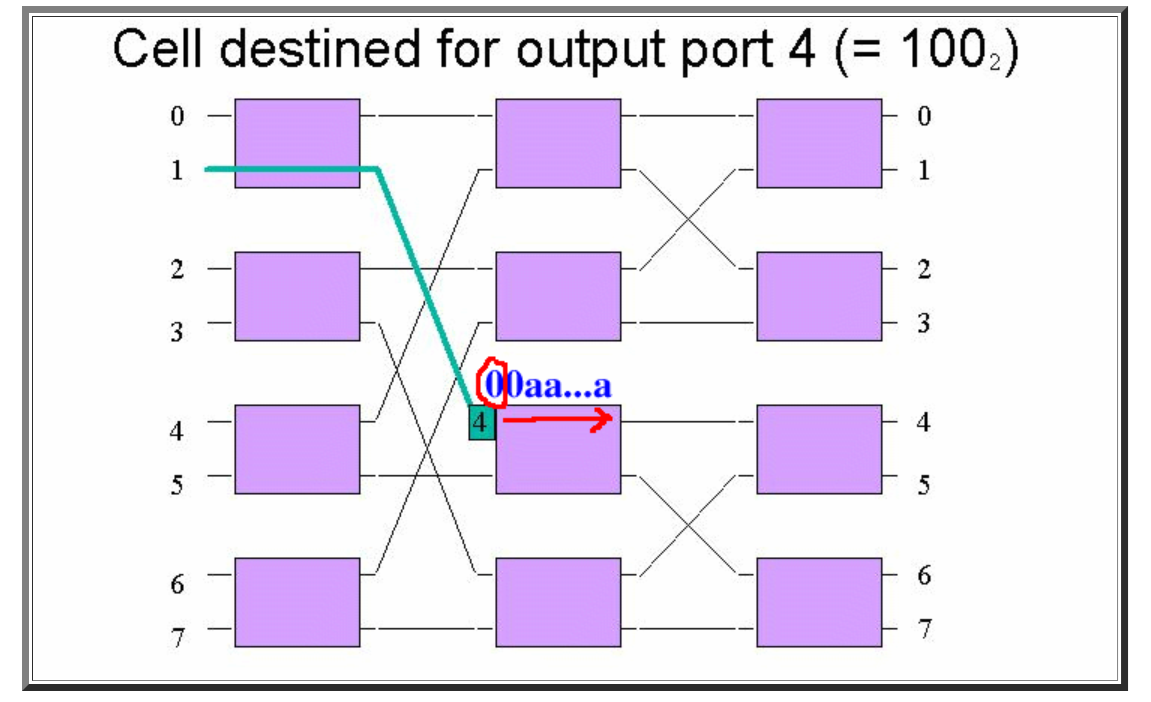
The Delta Interconnection Network is a digit controlled multi-stage interconnection network

made up with Banyan Switches. Each rectangle represents a 2x2 Banyan Switch.  
(Self) Routing in Delta Network :

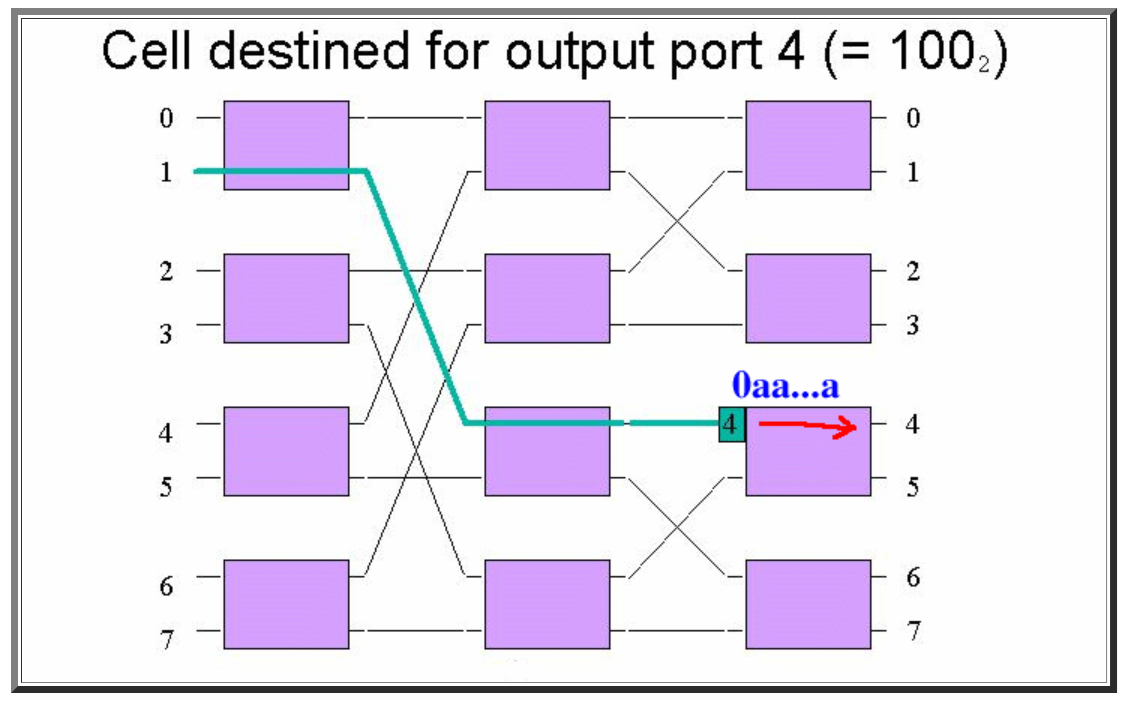
Suppose CPU 1 wants to send an address value to Memory Module 4.  
Step 1 of the transmission:  
  
Because the routing (first) bit is 1 , the entire message (except the routing bit) is forwarded onto the

lower connection.

Step 2 of the transmission:

  
Because the routing (first) bit is 0 , the entire message (except the routing bit) is forwarded onto

the upper connection.

Step 3 of the transmission:  
  
Because the routing (first) bit is 0 , the entire message (except the routing bit) is forwarded onto

the upper connection.

The address value is then "delivered" (connected) to the memory module 4.

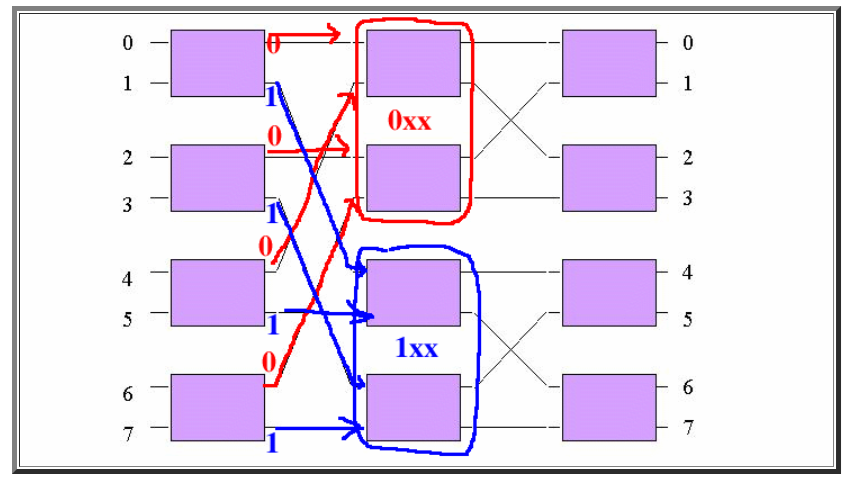
In each stage, the Delta Switch can discover the value of each routing bit.

Consider what happens during the first stage of the Delta Switch:

In each stage, the Delta Switch can discover the value of each routing bit.

Consider what happens during the first stage of the Delta Switch:

In each stage, the Delta Switch can discover the value of each routing bit.

Consider what happens during the first stage of the Delta Switch:  


If the routing (first) bit is equal to 0 (zero), then the message is routed towards the TWO UPPER

Banyan switches in the SECOND stage.

If the routing (first) bit is equal to 1 (one), then the message is routed towards the TWO LOWER

Banyan switches in the SECOND stage.

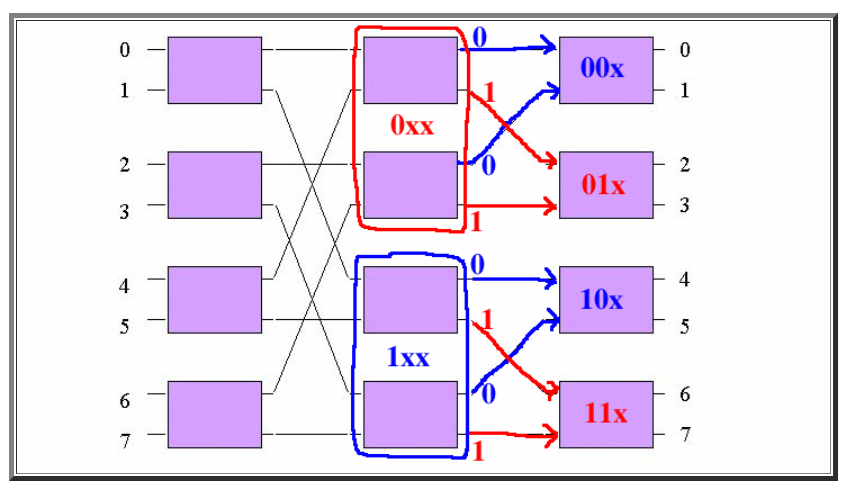
Therefore:

Messages arriving into the TWO UPPER Banyan switches in the SECOND stage always have a

destination address (ID of memory module) that is equal to 0xx.

Messages arriving into the TWO LOWER Banyan switches in the SECOND stage always have a

destination address (ID of memory module) that is equal to 1xx.

The destination address (ID of memory module) is then further identified by the second stage:  


If the routing bit (second bit in address) is equal to 0 (zero), then the message is routed towards the

TWO UPPER Banyan switches in the THIRD stage.

If the routing bit (second bit in address) is equal to 1 (one), then the message is routed towards the

TWO LOWER Banyan switches in the THIRD stage.

Therefore:

Messages arriving into the MOST UPPER Banyan switch in the THIRD stage always have a destination

address (ID of memory module) that is equal to 00x.

Messages arriving into the SECOND UPPER Banyan switch in the THIRD stage always have a

destination address (ID of memory module) that is equal to 01x.

Messages arriving into the THIRD UPPER Banyan switch in the THIRD stage always have a destination

address (ID of memory module) that is equal to 10x.

Messages arriving into the FOURTH UPPER (most lower) Banyan switch in the THIRD stage always

have a destination address (ID of memory module) that is equal to 11x.

**4.  
Batcher-Banyan network**

In a network where there are several inputs and outputs, when some of the inputs specify the information that you want to send to a specific terminal on the output side, it is sorted appropriately and output.

It consists of Batcher network and Banyan network.

Here, 2 ^ n inputs and n outputs each (n is any positive integer).

The output destination specified by the input is given as an integer from 1 to 2 ^ n.

Output destinations do not overlap for information input at the same time.

All input terminals do not have to have input.  
**Batcher Network**

A Batcher network is a network that sorts existing elements for an appropriate array and returns an array packed on one side. The basic structure of this network is a 2 \* 2 bitonic sorter that sorts 2 inputs and combines them. This creates an n \* n bitonic sorter, and then combines it with an n \* n bitonic sorter to create a Batcher network.

In general, the n \* n bitonic sorter first applies a 2 \* 2 bitonic sorter to n inputs, sorts the output appropriately, and then two (n / 2) \* (n / 2) bitonic sorters. (N / 2) \* (n / 2) If bitonic sorter is created, n \* n bitonic sorter can be created, so the structure can be described recursively.

The Batcher network starts with 2 \* 2 bitonic sorter and is created by connecting larger bitonic sorters in order. Therefore, it is sufficient to connect to the input long size n \* n bitonic sorter while rearranging the array appropriately. These are implemented programmatically by the following functions.

sorter\_base (): 2 \* 2 bitonic sorter

sorter (): n \* n bitonic sorter

batcher (): batcher network

**Banyan Network**

Banyan network is a network that arranges the output of Batcher network to an appropriate port considering the part without input elements. Since Banyan network processes bit strings, first output 2 of Batcher network. Convert to base number representation. At this time, replace the part without input element with None.

The basic structure of the Banyan network is a sorter that sorts by focusing on the specified element of the bit string with 2 inputs. This sorter is connected while changing which bit to focus on and sort. It is a Banyan network. Since the connection method is the same as n \* n bitonic sorter, it can be described recursively in the same way. After that, you can specify the bit of interest at each stage. These are as follows in the program. It is implemented by the function of.

banyan\_base (): 2 \* 2 sorter

banyan (): banyan network