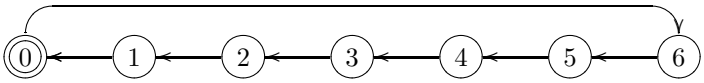
British Informatics Olympiad Final

12–14 April, 2002

Sponsored by Lionhead Studios

Bubble Memory

Modern *semiconductor memory* consists of many tiny electronic circuits, each of which can be on or off (i.e. storing a 0 or a 1). An alternative type of memory is *bubble memory*, which consists of many tiny bubbles that can be moved around; each bubble also stores a 0 or a 1 (by means of an electrical charge). Bubble memory has several advantages over modern memory; for example, bubbles can maintain their on/off value without power (i.e. it is non-volatile). Semiconductor memory has its own advantages, such as high speed, which is why it is the currently used technology.



Bubble memory is read and written by moving the bubbles around the chip until the required bubble is in the read/write position. In the above example there is a single path that the bubbles can follow; the read/write position is marked with a double circle. Three operations can take place on the example system: the marked element can be read, the marked element can be written or *all* the bubbles can be simultaneously moved along the path. [At any one time each position holds a single bubble.]

For example, to duplicate the contents of bubble 3 in bubble 2, eleven operations are required: three rotations, one read, six more rotations then one write. After these operations, bubble 2 is in position zero (marked with a double circle), bubble 3 is to its immediate right, etc...

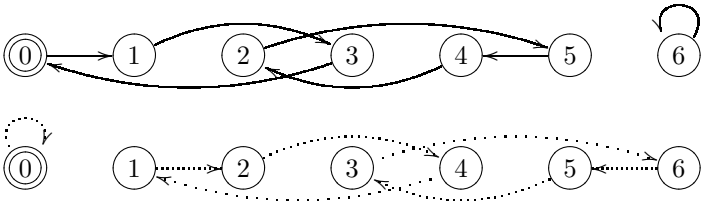
Question 1.1

The number of operations required to read b bubbles depends on the required bubbles, their order, current location and the total amount of memory. If the memory consists of 100 bubbles, what is the largest number of operations required to read 5 different bubbles (in the given order)? What is the smallest?

Question 1.2

Outline an algorithm that, given the bubble currently in the read / write position and the required bubble, calculates and applies the necessary operations.

We can decrease the number of operations required to access bubbles by adding additional paths for the bubbles to take. In the following example, two paths are shown for the data. [They are shown on separate diagrams for clarity but they are linking the same locations on the chip.] This bubble memory has four different types of operation: reading the marked element, writing the mark element, rotating the bubbles along the first path and rotating the bubbles along the second path.



Recall from round one the in-riffle and out-riffle; in these shuffles the pack is split into two halves which are then interleaved. The solid line path is equivalent to an in-riffle on the bubbles; similarly the dotted line path is equivalent to an out-riffle. [We will denote an operation of the solid path as a **1** and the dotted path as a **0**.] The single path in the first diagram is equivalent to the break (denoted **b**).

Question 1.3
In the above system, what is the effect of **101** on the bubbles?

Question 1.4
(a) In the above system, what is the largest number of moves that may be required to get a bubble into the read / write position?
(b) What is the largest number of moves if there are 19 bubbles (linked with an in-riffle path and an out-riffle path)?

For a combination of paths to be useful it is necessary to be able to move bubbles from any location to the read / write position. We could pair the break path with any other path and still have a useful combination. There are some paths which, if combined with the in-riffle path, do not create useful combinations; for example, leaving location 6 isolated from the read / write position.

Question 1.5
In a memory with 7 locations, how many different paths can the in-riffle be combined with to create useful combinations?

In the single path system it was only necessary to know which bubble was in the read / write position to be able to deduce where the other bubbles were. If we allowed arbitrary combinations of riffles in the two path system this would not be the case.

Question 1.6
(a) If arbitrary combinations of riffles were permitted, what additional information would you require to deduce the positions of the other bubbles? Detailed calculations are not required.
(b) Why is this undesirable?

It is possible to show that, if there are $2^n - 1$ bubbles, applying S_1 then S_2 ... then S_n (where each S_i is either a **1** or **0**) will move the bubble in position p on the chip, to position

$$(p + 2^{n-1}.S_1 + \cdots + 2^0.S_n) \textbf{ mod } (2^n - 1)$$

[a **mod** b is equivalent to the remainder when a is divided by b.] In other words, by applying n riffle operations we can get the required bubble in the read / write position and ensure the bubbles are kept in order.

Question 1.7
Outline an algorithm for a memory with $2^n - 1$ bubbles that, given the bubble currently in the read / write position and the required bubble, applies the necessary (riffle) path operations to get the required bubble and keep the data ordered.

When accessing memory there are two different types of request. The first is *random access* where the requested memory is not necessarily related to previously requested memory (and could be anywhere). The second is *sequential access* where are sequence of adjacent memory locations (in this case bubbles) are requested.

Question 1.8
How does your algorithm in 1.7 compare to the algorithm in 1.2 for random memory access? How about sequential access? Justify briefly, for small and large memory sizes.

One possible way of increasing the performance for sequential access would be to combine the two systems, creating an amalgamated system containing three paths. There are however manufacturing overheads when increasing the number of paths, such as requiring better equipment, higher chip densities and getting higher failure rates.

Question 1.9
Find a relationship between the three paths (**b**, **1** and **0**) and hence show how a *two* path system can be built which is suitable for random and sequential access.

Question 1.10

Suppose the memory size is $2^{16} - 1$ bubbles. What is the largest number of operations that your new system might require when retrieving a bubble?

Question 1.11

Are bubbles ever available, at the read / write position, before a sequence of operations is complete? If bubbles were accessible, would there be any advantage in using them?

Question 1.12

The shuffle based bubble memories that we have discussed rely on the memory having $2^n - 1$ locations. It is more usual (and useful) to have memories with 2^n locations. How would you adapt the previous methods to 2^n locations?

One advantage of the single path bubble memory, compared to the multiple path systems we have considered, is that it can be built flat without any of the paths crossing over each other. Another 'multiple path' approach, which keeps the same property, is to have several single paths, each of which stores different sections of memory. Such a system also requires an additional mechanism for getting the bubbles between the single paths and the read / write position.

Question 1.13

Briefly outline a suitable mechanism for moving bubbles from the single paths to the read / write position. Mention the hardware design, movement algorithm and comment on how you would distribute the bubbles amongst the single paths.