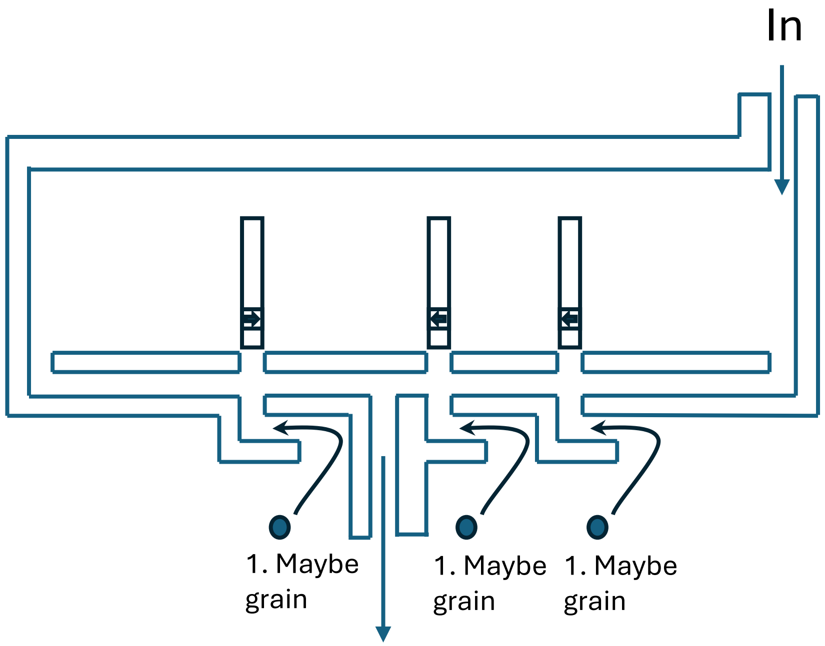
Combination AND/OR gate

A) Ants must line up ‘pins’ with a hole to allow passage.
B) Dedicated to ‘X ∨ (Y ∧ Z)’ operation in formulas.

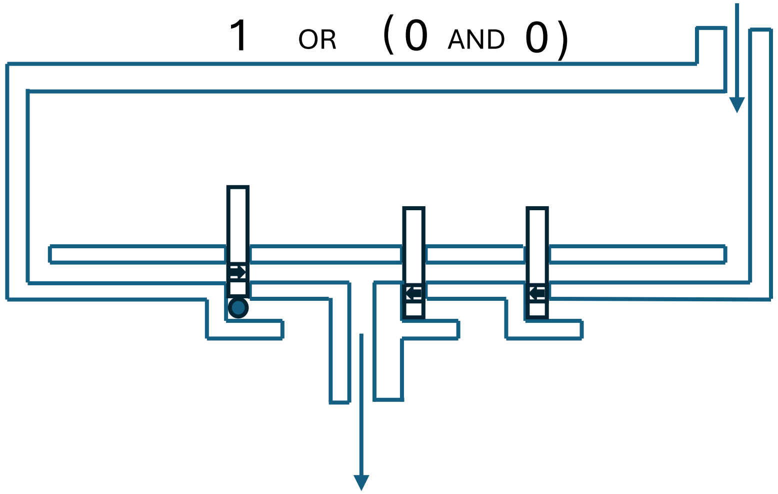


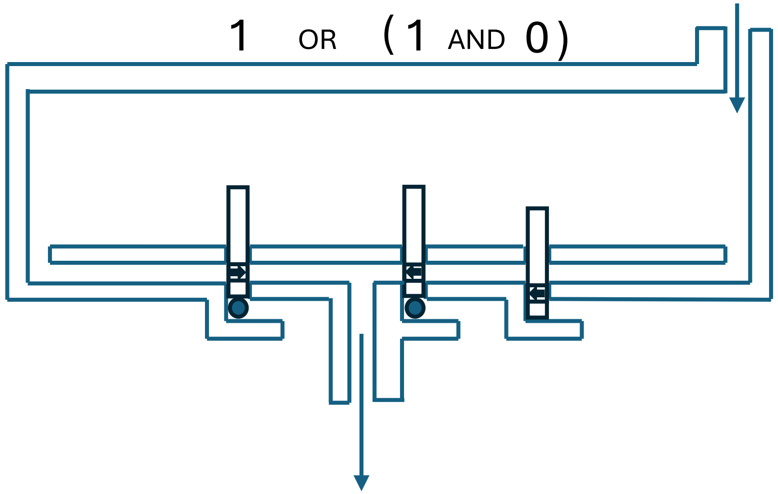
Out, if ant can make it here, output = 1

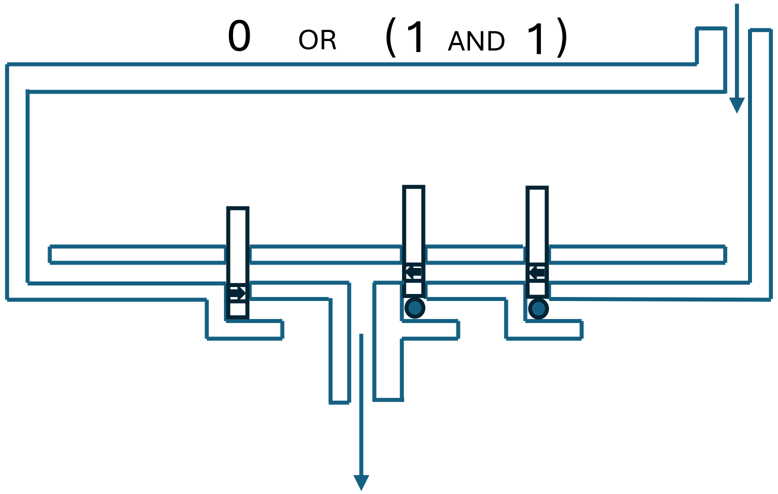
Procedure:

1. Ants load up needed inputs
2. Three gates are closed; if correct inputs are present , ant can ‘walk through’ with its grain and is taken as output = 1
3. Else, output is = 0
4. Grains in gate must be cleared out to reset gate.

Ant can ‘pass through’ to return ‘1’ when the following gate configurations are present , ‘0’ otherwise.

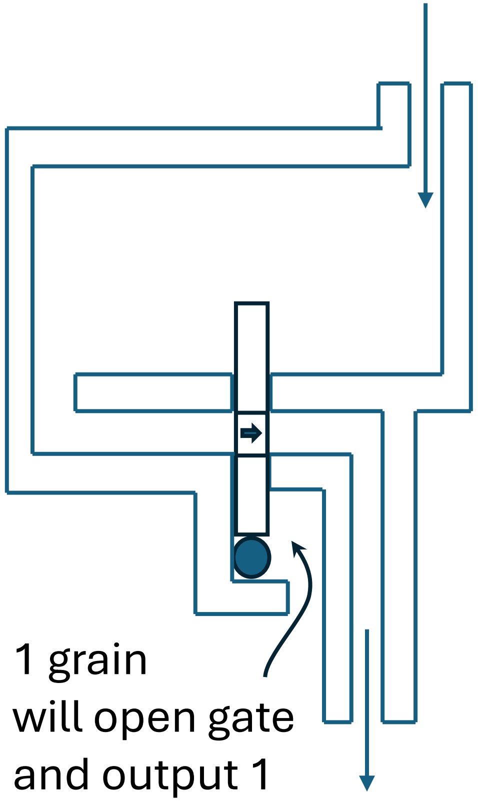






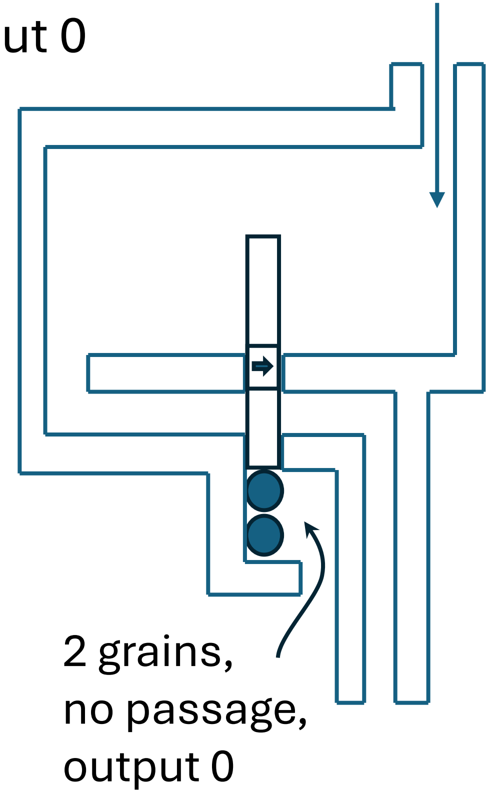
$\oplus$-gate

Output 1



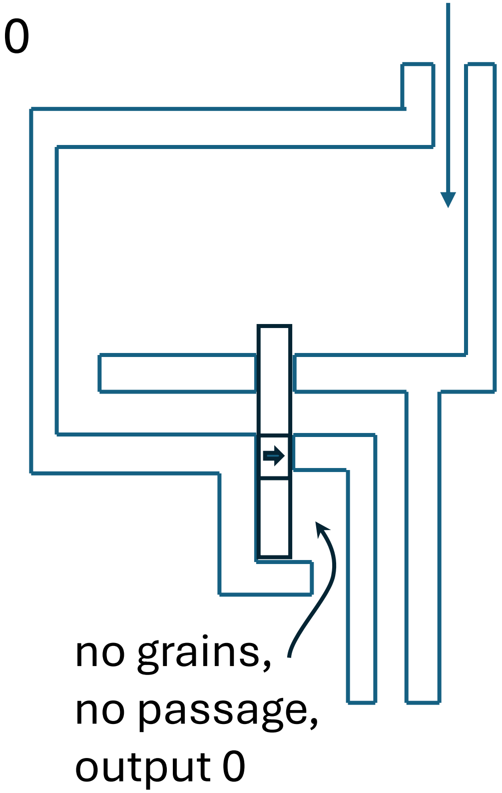
$1 \oplus 0$

Output 0



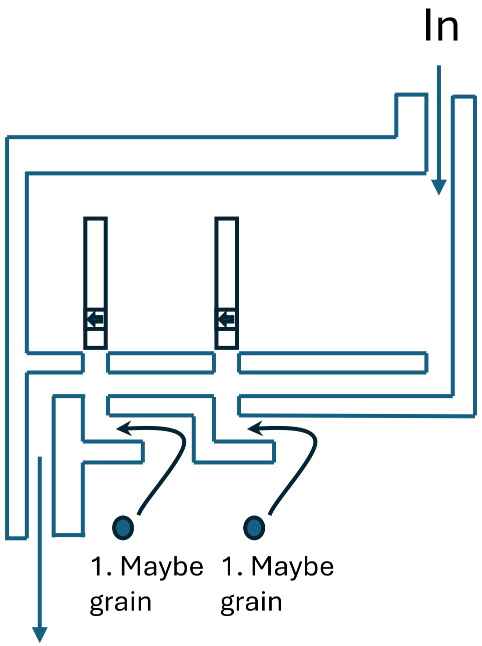
$1 \oplus 1$

Output 0

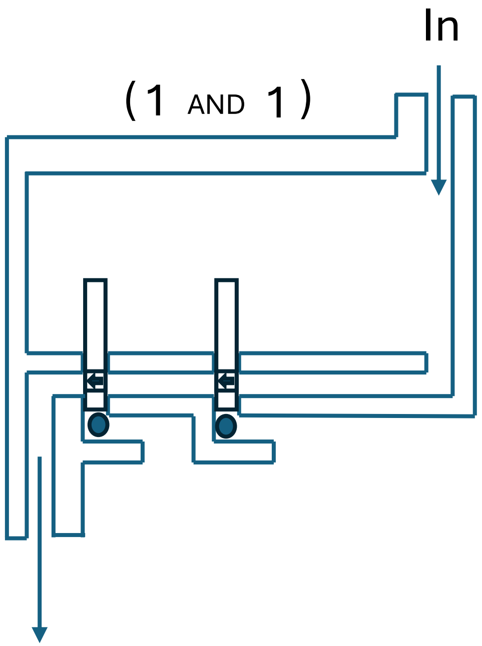


$0 \oplus 0$

AND Gate



Out, if ant can make it here, output = 1



Ant can only pass through if 2 grains loaded

Equations and where their calculations happen:

$P_i = A_i \oplus B_i$; at XOR gate
$G_i = A_i \land B_i$; at AND gate

(G,P) tree calculations:

$G_{j:l} = G_R \lor (P_R \land G_L)$; happens at combo OR/AND.
$P_{j:l} = (P_R \land P_L)$; happens at combo OR/AND.

Carries:

$C_i = G_{i:0} \lor (P_{i:0} \land C_0)$; happens at combo OR/AND.

Sums happen at XOR gate.

A standalone ‘OR’ gate is not being considered because no formula uses an ‘OR’ operation in isolation.

Memory cells

1. Formula values are stored in a table with its identity encoded by essentially ticking boxes with grains.
2. Procedure:
   a) Ant picks next available memory row.
   b) It ‘reserves’ the row by placing a grain in ‘Reserved’.
   c) Cells are tagged to describe variable identity (see examples).
   d) When variable has been encoded, ‘Read Ready’ receives a grain and is then available to be read by other ants.
   e) Row will remain until ‘Reserved’ grain is removed; grains in that row will then be marked as needing to be cleared.

HUMAN READABLE	Variable				Index					Value	Reserved	Read ready
	G	P	C	S	4	3	2	1	0			

Etc

.
.
.
.
.

Memory cells storage examples

HUMAN READABLE	Variable				Index					Value	Reserved	Read ready
	G	P	C	S	4	3	2	1	0			
1. $G_{2:0} = 1$	●						●		●	●	●	●
2. $P_0 = 0$		●							●		●	●
3. $C_4 = 1$			●		●					●		●
4. $S_2 = 0$				●			●				●	

Etc

.

.

.

.

.

Notice:

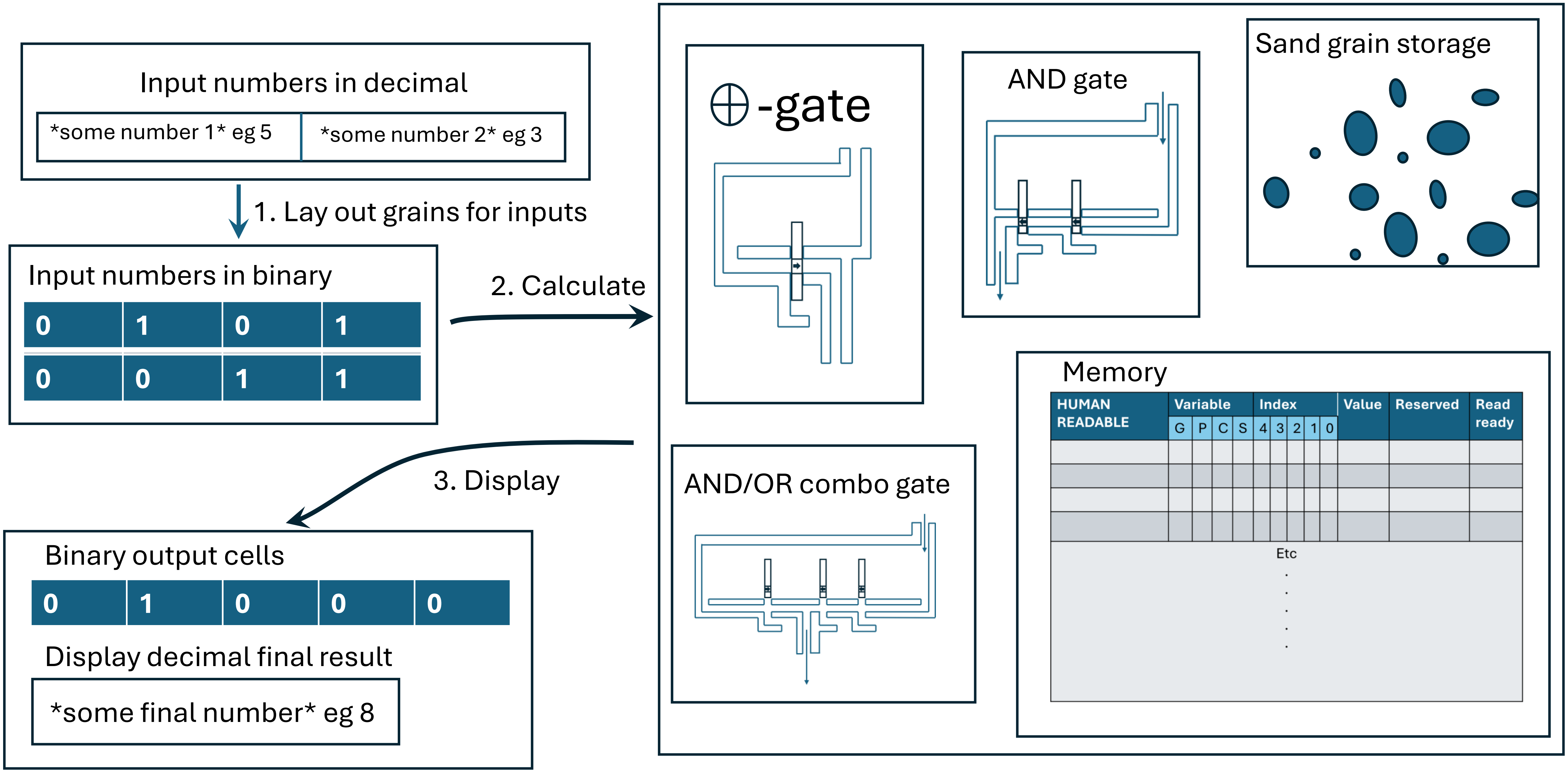
A) Entry number 1; Variables with 2 or more index descriptors (e.g. generate bit $G_{2:0}$ has 2 and 0) will have multiple indexes ticked.

B) Entry number 3; contains values and is read-ready; however, ‘reserved’ is not ticked and so will be cleared out if an ant comes across it.

C) Entry number 4; contains data but ‘read-ready’ is not ticked. Data is possibly still being added; ants cannot use its information.

D) No tuple values (eg (G,P)) are stored as one; must be G and P separate

Rough proposed ant board and components



Procedure assuming 1 ant

1. Ant 'looks at' one of the two inputs.
2. It moves to the input table and begins to lay out ONE of the inputs, from LSB to MSB, fetching grains as needed.
3. Ant 'looks' at the other number, then does the same.

Input numbers in decimal

some number A eg 5	*some number B* eg 3

Input numbers in binary

0	1	0	1	A
0	0	1	1	B

4. Ant will now calculate P and G rows, starting with P LSB->MSB, using formula:

$$P_i = A_i \text{ XOR } B_i \text{ ?}$$

5. Memory row reserved for each Pi; filled in except for read-ready and value.
6. Each 'calculation' takes one trip to XOR gate, requiring up to three grains (two for input, another to try and pass through with).
7. The result is added to row 'value' for Pi, and 'read-ready' is checked.
8. When the P row is filled in, the ant starts with G in the same vein using the AND gate. AND grains are loaded, pins are closed, and ant records if he can pass through.
9. (G,P) for each i now calculated.
10. Ant then calculates $P_{1\text{-}0}$, $P_{3\text{-}2}$, and $P_{3\text{-}0}$ (AND gate)
11. Ant then calculates $G_{1\text{-}0}$, $G_{3\text{-}2}$, and $G_{3\text{-}0}$ (OR/AND gate)
12. Ant then calculates $P_{2:0}$ small using P_2 and $P_{1:0}$ at AND gate
13. And then calculates $G_{2:0}$ small using G_2, P_2, and $G_{1:0}$ at OR/AND gate.
14. All (G,P) values are now known.
15. Each carry determined at OR/AND gate
16. Finally, all S_i calculated at XOR gate.
17. Each Si is then carried to output binary space, and when full, our decimal is displayed.

If more than 1 ant available, the following procedure will allow parallel tasks:

a) Reading in input decimals; for each ant that 'looks' at an input number, it can start helping 'unpack' the binary.
b) When a pair [A_i , B_i] becomes available, the P and G values for i can be calculated

Index				Var
3	2	1	0	
			1	A
			0	B
			*	P
			*	G

*i=0 for both A and B filled; $(G,P)_I$ can now begin in parallel

d) If either (G,P) for 1,0 or (G,P) for 3,2 are available, $(G,P)_{1:0}$ or $(G,P)_{3:2}$ can be calculated.

e) Only *after* small binary $(G,P)_{2:0}$ has been calculated, can the carries be calculated in parallel.
f) As pairs (P_i , C_i) become available, their S_i can be calculated.

Graph summary of how parallelism ‘yields’ to other processes (assuming $C_0 = 0$)