Turing Machines (Preliminary)

Tom Davis
tomrdavis@earthlink.net
http://www.geometer.org/mathcircles/turing.pdf
November 16, 2025

1 Turing Machines

A Turing machine is a mathematical model of a digital computer, named after Alan Turing.

Turing machines may seem incredibly simplistic, and it is obvious that any Turing machine can be simulated by a modern computer. The striking fact is the converse: *every* computation that can be performed on a digital computer can also be performed by an appropriate Turing machine.

In fact, for every reasonable definition of "computable function," there is a Turing machine that computes it.

In this document we define a minimal Turing machine and look at some programs for it. No matter how simple this machine is, it is as powerful as *any* Turing machine.

A Turing machine has a fixed, finite set of states labeled **A**, **B**, **C**, . . . , **Z**, where **Z** is the halting state. The machine begins in state **A**. Since every machine has a halted state and when that state is reached all calculation ceases, the halted state is not counted as a real state. A machine that can be in states **A**, **B**, and **Z** is considered to be a two-state machine.

The input/output medium is a tape, infinite in both directions and divided into squares. Exactly one symbol appears in each square; our alphabet is **0**, **1**. The tape is initially all **0**.

At any time, the machine is in some state and is scanning one square. For each (state, scanned symbol) pair, the program specifies:

- which symbol to write (**0** or **1**),
- which way to move (one square L or R), and
- which state to enter next.

To leave a square unchanged, the machine simply writes back the symbol it is currently scanning.

1.1 Examples

Here is a machine that will print a single **1** and will halt:

	0	**1**
A	**1** $\rightarrow$ **Z**	**0** $\rightarrow$ **Z**

The starting condition is a tape filled with **0**s with the machine "looking" at one of them and beginning in state **A**. The transition table above shows that if the machine is in state **A** and is looking at a **0** it should write a **1**, move one square to the right, and then go into state **Z** (in other words, halt).

The entry for state **A** looking at a **1** will never be used, so the contents of this position in the transition table doesn't matter. Note below the fact that the original **0** has changed to **1** and the machine is in state **Z** looking at the **0** to the right of the original position.

start	⋯	0	0	0	0	0	0	0	0A	0	0	0	0	0	0	0	⋯
1	⋯	0	0	0	0	0	0	0	1	0Z	0	0	0	0	0	0	⋯

As a second example, here is a program that writes three **1**s in a row and then halts:

	0	**1**
A	$1 \rightarrow B$	$1 \rightarrow B$
B	$1 \rightarrow C$	$1 \rightarrow C$
C	$1 \rightarrow Z$	$1 \rightarrow Z$

Note that the program essentially uses states to count. Each time a **1** is written the machine moves to the right and moves to the next state. After three **1**s are written, the machine moves and halts.

start	⋯	0	0	0	0	0	0	0	0A	0	0	0	0	0	0	0	⋯
1	⋯	0	0	0	0	0	0	0	1	0B	0	0	0	0	0	0	⋯
2	⋯	0	0	0	0	0	0	0	1	1	0C	0	0	0	0	0	⋯
3	⋯	0	0	0	0	0	0	0	1	1	1	0Z	0	0	0	0	⋯

Here is a more complex example. It is a good exercise to verify the 10-step calculation starting from a tape initially filled with **0**s and beginning in state **A**.

As a reminder: before each step note the machine's current state and whether the machine is looking at a **0** or a **1**. Look up the three-character string in that row/column and do three things:

- Write the first character over the character in the current square.
- Move one square left or right, depending on the arrow in the second position.
- Put the machine into the state indicated by the final of the three characters.

In this case, since you begin in state **A** looking at a **0**, the string will be $1 \rightarrow B$, so you write a **1**, move one square to the right, and go into state **B**. You should see this result as the second row of the computation. Keep repeating until you are in state **Z**; in other words, halted.

	0	**1**
A	$1 \rightarrow B$	$0 \leftarrow A$
B	$1 \leftarrow C$	$1 \leftarrow A$
C	$0 \leftarrow D$	$1 \rightarrow D$
D	$1 \leftarrow Z$	$0 \rightarrow B$

Here is the calculation:

start	···	0	0	0	0	0	0	0	0A	0	0	0	0	0	0	0	···
1	···	0	0	0	0	0	0	0	1	0B	0	0	0	0	0	0	···
2	···	0	0	0	0	0	0	0	1C	1	0	0	0	0	0	0	···
3	···	0	0	0	0	0	0	0	1	1D	0	0	0	0	0	0	···
4	···	0	0	0	0	0	0	0	1	0	0B	0	0	0	0	0	···
5	···	0	0	0	0	0	0	0	1	0C	1	0	0	0	0	0	···
6	···	0	0	0	0	0	0	0	1D	0	1	0	0	0	0	0	···
7	···	0	0	0	0	0	0	0	0	0B	1	0	0	0	0	0	···
8	···	0	0	0	0	0	0	0	0C	1	1	0	0	0	0	0	···
9	···	0	0	0	0	0	0	0D	0	1	1	0	0	0	0	0	···
10	···	0	0	0	0	0	0Z	1	0	1	1	0	0	0	0	0	···

1.2 Badly-Behaving Turing Machines

The machine has only one state, **A** (not counting **Z**), and it never halts. It writes 1s forever to the right.

	0	**1**
A	**1** $\rightarrow$ **A**	**1** $\leftarrow$ **Z**

The machine only has one state, **A**, not counting the halted state **Z**, but it will never halt: at every stage the machine will be looking at a **0**, so it will always write a **1** and move to the right. It will never encounter a **1** so it will never halt. It will write **1**s forever to the right.

A very interesting question is this: Given a program that describes a Turing machine, will it eventually halt? This is known as the "Halting Problem for Turing Machines" and the answer is a little discouraging. The problem is that this question is impossible to answer.

2 Turing Programming Exercises

Heere are a few programming exercises to learn some techniques. In this section we will look at ways to emulate arithmetic calculations with whole numbers on a Turing machine.

The first problem is, "How do we represent integers with only **0** and **1** in our alphabet? In this section, we will simply represent the whole number n with n **1**s in a row (with a **0** at each end). In this section we will also assume that the calculation begins in state **A**, looking at the leftmost **1** in the number[1].

Here are some problems that you should attempt to solve. Solutions appear in the next Section 3. All of the examples will use the number 4 (represented by four **1**s).

1. Write a program that will increment the number ("increment" means "add 1 to". This can be done in various ways. One is just to march to the right until a **0** is encountered, and then to turn that **0** to a **1** and halt.

   Better would be to return to the beginning of the number after writing a **1** leaving the machine is the normal situation where the Turing machine is looking at the first **1** in the number.

   Alternatively, if you don't care where on the tape the final number appears, you could just take one step to the left and write a **1**. You aslo need to make sure the machine is looking at the first **1** of the (new, incremented) number.

2. Calculate the parity of the number. If the number is even, halt with the machine looking at a **0**; otherwise, a **1**.

[1]Notice that this makes it impossible to represent zzero, so another way to implement the number n is to use $n + 1$ copeis of **1** to represent n.

3. (Hard!) Assume that the machine is in state **A** and is looking at the leftmost **1** in a series of **1**s. Assume the rest of the tape is full of **0**s. Write a program that makes a copy of the string of **1**s separated from the orignal string by a single **0**.

3 Exercise Solutions

1. Just write the final **1**. Do not bother to return to the beginning of the number.

	0	1
A	1 $\rightarrow$ Z	1 $\rightarrow$ A

start	···	0	0	0	0	0	0	0	1A	1	1	1	0	0	0	0	···
1	···	0	0	0	0	0	0	0	1	1A	1	1	0	0	0	0	···
2	···	0	0	0	0	0	0	0	1	1	1A	1	0	0	0	0	···
3	···	0	0	0	0	0	0	0	1	1	1	1A	0	0	0	0	···
4	···	0	0	0	0	0	0	0	1	1	1	1	0A	0	0	0	···
5	···	0	0	0	0	0	0	0	1	1	1	1	1	0Z	0	0	···

Here is the solution where the machine is returned to the normal position:

	0	1
A	1 $\leftarrow$ B	1 $\rightarrow$ A
B	0 $\rightarrow$ Z	1 $\leftarrow$ B

start	···	0	0	0	0	0	0	0	1A	1	1	1	0	0	0	0	···
1	···	0	0	0	0	0	0	0	1	1A	1	1	0	0	0	0	···
2	···	0	0	0	0	0	0	0	1	1	1A	1	0	0	0	0	···
3	···	0	0	0	0	0	0	0	1	1	1	1A	0	0	0	0	···
4	···	0	0	0	0	0	0	0	1	1	1	1	0A	0	0	0	···
5	···	0	0	0	0	0	0	0	1	1	1	1B	1	0	0	0	···
6	···	0	0	0	0	0	0	0	1	1	1B	1	1	0	0	0	···
7	···	0	0	0	0	0	0	0	1	1B	1	1	1	0	0	0	···
8	···	0	0	0	0	0	0	0	1B	1	1	1	1	0	0	0	···
9	···	0	0	0	0	0	0	0B	1	1	1	1	1	0	0	0	···
10	···	0	0	0	0	0	0	0	1Z	1	1	1	1	0	0	0	···

Here is the solution where the actual starting position changes:

	0	1
A	1 $\rightarrow$ B	1 $\leftarrow$ A
B	1 $\leftarrow$ Z	1 $\leftarrow$ Z

start	···	0	0	0	0	0	0	0	1A	1	1	1	0	0	0	0	···
1	···	0	0	0	0	0	0	0A	1	1	1	1	0	0	0	0	···
2	···	0	0	0	0	0	0	1	1B	1	1	1	0	0	0	0	···
3	···	0	0	0	0	0	0	1Z	1	1	1	1	0	0	0	0	···

2. As we march to the right, we alternate states between **A** and **B** as long as we see a **1**. The parity just depends on whether we are in state **A** or **B** when we hit the **0**.

	0	1
A	0 $\leftarrow$ C	0 $\rightarrow$ B
B	1 $\leftarrow$ C	0 $\rightarrow$ A
C	1 $\rightarrow$ Z	1 $\rightarrow$ Z

The first solution shows that it works with even parity, and the second, with odd parity.

start	···	0	0	0	0	0	0	0	1A	1	1	1	0	0	0	0	···
1	···	0	0	0	0	0	0	0	0	1B	1	1	0	0	0	0	···
2	···	0	0	0	0	0	0	0	0	0	1A	1	0	0	0	0	···
3	···	0	0	0	0	0	0	0	0	0	0	1B	0	0	0	0	···
4	···	0	0	0	0	0	0	0	0	0	0	0	0A	0	0	0	···
5	···	0	0	0	0	0	0	0	0	0	0	0C	0	0	0	0	···
6	···	0	0	0	0	0	0	0	0	0	0	0	0Z	0	0	0	···

start	···	0	0	0	0	0	0	0	1A	1	1	0	0	0	0	0	···
1	···	0	0	0	0	0	0	0	0	1B	1	0	0	0	0	0	···
2	···	0	0	0	0	0	0	0	0	0	1A	0	0	0	0	0	···
3	···	0	0	0	0	0	0	0	0	0	0	0B	0	0	0	0	···
4	···	0	0	0	0	0	0	0	0	0	0C	1	0	0	0	0	···
5	···	0	0	0	0	0	0	0	0	0	0	1Z	0	0	0	0	···

3. Here is the solution:

	0	1
A	0 $\leftarrow$ F	0 $\rightarrow$ B
B	0 $\rightarrow$ C	1 $\rightarrow$ B
C	1 $\leftarrow$ D	1 $\rightarrow$ C
D	0 $\leftarrow$ E	1 $\leftarrow$ D
E	1 $\rightarrow$ A	1 $\leftarrow$ E
F	0 $\rightarrow$ Z	1 $\leftarrow$ F

The next three blocks show the execution assuming that the number of **1**s is 1, 2, or 3.

start	···	0	0	0	0	0	0	0	1A	0	0	0	0	0	0	0	···
1	···	0	0	0	0	0	0	0	0	0B	0	0	0	0	0	0	···
2	···	0	0	0	0	0	0	0	0	0	0C	0	0	0	0	0	···
3	···	0	0	0	0	0	0	0	0	0D	1	0	0	0	0	0	···
4	···	0	0	0	0	0	0	0	0E	0	1	0	0	0	0	0	···
5	···	0	0	0	0	0	0	0	1	0A	1	0	0	0	0	0	···
6	···	0	0	0	0	0	0	0	1F	0	1	0	0	0	0	0	···
7	···	0	0	0	0	0	0	0F	1	0	1	0	0	0	0	0	···
8	···	0	0	0	0	0	0	0	1Z	0	1	0	0	0	0	0	···

start	···	0	0	0	0	0	0	0	1A	1	0	0	0	0	0	0	···
1	···	0	0	0	0	0	0	0	0	1B	0	0	0	0	0	0	···
2	···	0	0	0	0	0	0	0	0	1	0B	0	0	0	0	0	···
3	···	0	0	0	0	0	0	0	0	1	0	0C	0	0	0	0	···
4	···	0	0	0	0	0	0	0	0	1	0D	1	0	0	0	0	···
5	···	0	0	0	0	0	0	0	0	1E	0	1	0	0	0	0	···
6	···	0	0	0	0	0	0	0	0E	1	0	1	0	0	0	0	···
7	···	0	0	0	0	0	0	0	1	1A	0	1	0	0	0	0	···
8	···	0	0	0	0	0	0	0	1	0	0B	1	0	0	0	0	···
9	···	0	0	0	0	0	0	0	1	0	0	1C	0	0	0	0	···
10	···	0	0	0	0	0	0	0	1	0	0	1	0C	0	0	0	···
11	···	0	0	0	0	0	0	0	1	0	0	1D	1	0	0	0	···
12	···	0	0	0	0	0	0	0	1	0	0D	1	1	0	0	0	···
13	···	0	0	0	0	0	0	0	1	0E	0	1	1	0	0	0	···
14	···	0	0	0	0	0	0	0	1	1	0A	1	1	0	0	0	···
15	···	0	0	0	0	0	0	0	1	1F	0	1	1	0	0	0	···
16	···	0	0	0	0	0	0	0	1F	1	0	1	1	0	0	0	···
17	···	0	0	0	0	0	0	0F	1	1	0	1	1	0	0	0	···
18	···	0	0	0	0	0	0	0	1Z	1	0	1	1	0	0	0	···

start	···	0	0	0	0	0	0	1A	1	1	0	0	0	0	0	0	···
1	···	0	0	0	0	0	0	0	1B	1	0	0	0	0	0	0	···
2	···	0	0	0	0	0	0	0	1	1B	0	0	0	0	0	0	···
3	···	0	0	0	0	0	0	0	1	1	0B	0	0	0	0	0	···
4	···	0	0	0	0	0	0	0	1	1	0	0C	0	0	0	0	···
5	···	0	0	0	0	0	0	0	1	1	0D	1	0	0	0	0	···
6	···	0	0	0	0	0	0	0	1	1E	0	1	0	0	0	0	···
7	···	0	0	0	0	0	0	0	1E	1	0	1	0	0	0	0	···
8	···	0	0	0	0	0	0	0E	1	1	0	1	0	0	0	0	···
9	···	0	0	0	0	0	0	1	1A	1	0	1	0	0	0	0	···
10	···	0	0	0	0	0	0	1	0	1B	0	1	0	0	0	0	···
11	···	0	0	0	0	0	0	1	0	1	0B	1	0	0	0	0	···
12	···	0	0	0	0	0	0	1	0	1	0	1C	0	0	0	0	···
13	···	0	0	0	0	0	0	1	0	1	0	1	0C	0	0	0	···
14	···	0	0	0	0	0	0	1	0	1	0	1D	1	0	0	0	···
15	···	0	0	0	0	0	0	1	0	1	0D	1	1	0	0	0	···
16	···	0	0	0	0	0	0	1	0	1E	0	1	1	0	0	0	···
17	···	0	0	0	0	0	0	1	0E	1	0	1	1	0	0	0	···
18	···	0	0	0	0	0	0	1	1	1A	0	1	1	0	0	0	···
19	···	0	0	0	0	0	0	1	1	0	0B	1	1	0	0	0	···
20	···	0	0	0	0	0	0	1	1	0	0	1C	1	0	0	0	···
21	···	0	0	0	0	0	0	1	1	0	0	1	1C	0	0	0	···
22	···	0	0	0	0	0	0	1	1	0	0	1	1	0C	0	0	···
23	···	0	0	0	0	0	0	1	1	0	0	1	1D	1	0	0	···
24	···	0	0	0	0	0	0	1	1	0	0	1D	1	1	0	0	···
25	···	0	0	0	0	0	0	1	1	0	0D	1	1	1	0	0	···
26	···	0	0	0	0	0	0	1	1	0E	0	1	1	1	0	0	···
27	···	0	0	0	0	0	0	1	1	1	0A	1	1	1	0	0	···
28	···	0	0	0	0	0	0	1	1	1F	0	1	1	1	0	0	···
29	···	0	0	0	0	0	0	1	1F	1	0	1	1	1	0	0	···
30	···	0	0	0	0	0	0	1F	1	1	0	1	1	1	0	0	···
31	···	0	0	0	0	0	0F	1	1	1	0	1	1	1	0	0	···
32	···	0	0	0	0	0	0	1Z	1	1	0	1	1	1	0	0	···

4 The Busy Beaver Function

The Busy Beaver function $\Sigma(n)$ is this: For every n-state machine as described above, begin with a tape initially filled with zeros and run the machine until it halts. Some number of **1**s will have printed, and $\Sigma(n)$ is the maximum number of **1**s. The problem, of course, is that some machines don't halt so the problem can't be solved simply by trying out every possible machine with n states.

Here are some known facts:

- $\Sigma(1) = 1$.

- $\Sigma(2) = 4$. This can be proved by enumeration of all possible machines.
- $\Sigma(3) = 6$. This is *not* easy to prove.
- $\Sigma(4) = 13$.
- For $n > 4$, nobody knows $\Sigma(n)$. $\Sigma(5) \geq 4098$, and $\Sigma(6) \geq 3.514 \times 10^{18267}$. As n increases, these numbers grow astronomically.

Here are a few exercises:

- Prove that $\Sigma(1) = 1$. There aren't very many of them and you can just try all of them.
- Show that the following 2-state machine halts after printing four **1**s. Since it is a 2-state machine this proves that $\Sigma(2) \geq 4$.

	0	**1**
A	**1** $\rightarrow$ **B**	**1** $\leftarrow$ **B**
B	**1** $\leftarrow$ **A**	**1** $\rightarrow$ **Z**

  See Section 6 for a solution.

- Here is the best 3-state machine:

	0	**1**
A	**1** $\rightarrow$ **B**	**1** $\rightarrow$ **Z**
B	**0** $\rightarrow$ **C**	**1** $\rightarrow$ **B**
C	**1** $\leftarrow$ **C**	**1** $\leftarrow$ **A**

  Show that it requires 14 steps and yields a string of 6 **1**s when it finally halts. There is a solution in Section 6.

- Write a program with the minimal number of states that prints exactly two *on*s and then halts.
- Assume the number n is represented by $n + 1$ copies of **1** in a row. If two numbers are initially on the tape separated by one **0** and the machine initially is looking at the leftmost **1** in the leftmost number, write a program to add the two numbers. In other words, to add $2 + 3$ begin with:

start	$\cdots$	**0**	**1A**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	$\cdots$

  and finish with something like:

end	$\cdots$	**0**	**1A**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	$\cdots$

  Don't worry about the final state and position when the calculation terminates. Remember that the answer $2 + 3 = 5$ so the answer should have $5 + 1 = 6$ **1**s in a row.

5 Turing Worksheet

The easiest way to use this is to assume blank squares are filled with **0**s. As you go from line to line, you need to copy down any **1**s.

start	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···
	···														···

start	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·
	· · ·														· · ·

6 Busy Beaver Solutions

	0	1
A	1 → B	1 ← B
B	1 ← A	1 → Z

Let's follow the computation one step at a time:

start	···	0	0	0	0	0A	0	0	0	0	···
1	···	0	0	0	0	1	0B	0	0	0	···
2	···	0	0	0	0	1A	1	0	0	0	···
3	···	0	0	0	0B	1	1	0	0	0	···
4	···	0	0	0A	1	1	1	0	0	0	···
5	···	0	0	1	1B	1	1	0	0	0	···
6	···	0	0	1	1	1Z	1	0	0	0	···

Here is the simulation for the following Turing machine. This machine, in fact, is an example of a three-state machine that achieves the longest row of 1s before halting. Here's the machine:

	0	1
A	1 → B	1 → Z
B	0 → C	1 → B
C	1 ← C	1 ← A

And here's the simulation:

start	···	0	0	0	0	0A	0	0	0	0	···
1	···	0	0	0	0	1	0B	0	0	0	···
2	···	0	0	0	0	1	0	0C	0	0	···
3	···	0	0	0	0	1	0C	1	0	0	···
4	···	0	0	0	0	1C	1	1	0	0	···
5	···	0	0	0	0A	1	1	1	0	0	···
6	···	0	0	0	1	1B	1	1	0	0	···
7	···	0	0	0	1	1	1B	1	0	0	···
8	···	0	0	0	1	1	1	1B	0	0	···
9	···	0	0	0	1	1	1	1	0B	0	···
10	···	0	0	0	1	1	1	1	0	0C	···
11	···	0	0	0	1	1	1	1	0C	1	···
12	···	0	0	0	1	1	1	1C	1	1	···
13	···	0	0	0	1	1	1A	1	1	1	···
14	···	0	0	0	1	1	1	1Z	1	1	···

Here is the machine that generates 13 1s in 107 steps with a four-state machine:

	0	1
A	1 → B	1 ← B
B	1 ← A	0 ← C
C	1 → Z	1 ← D
D	1 → D	0 → A

start	···	0	0	0	0	0	0	0	0	0	0	0	0A	0	0	0	0	0	···
1	···	0	0	0	0	0	0	0	0	0	0	0	1	0B	0	0	0	0	···
2	···	0	0	0	0	0	0	0	0	0	0	0	1A	1	0	0	0	0	···
3	···	0	0	0	0	0	0	0	0	0	0	0B	1	1	0	0	0	0	···
4	···	0	0	0	0	0	0	0	0	0	0A	1	1	1	0	0	0	0	···
5	···	0	0	0	0	0	0	0	0	0	1	1B	1	1	0	0	0	0	···
6	···	0	0	0	0	0	0	0	0	0	1C	0	1	1	0	0	0	0	···
7	···	0	0	0	0	0	0	0	0	0D	1	0	1	1	0	0	0	0	···
8	···	0	0	0	0	0	0	0	0	1	1D	0	1	1	0	0	0	0	···
9	···	0	0	0	0	0	0	0	0	1	0	0A	1	1	0	0	0	0	···
10	···	0	0	0	0	0	0	0	0	1	0	1	1B	1	0	0	0	0	···
11	···	0	0	0	0	0	0	0	0	1	0	1C	0	1	0	0	0	0	···
12	···	0	0	0	0	0	0	0	0	1	0D	1	0	1	0	0	0	0	···
13	···	0	0	0	0	0	0	0	0	1	1	1D	0	1	0	0	0	0	···
14	···	0	0	0	0	0	0	0	0	1	1	0	0A	1	0	0	0	0	···
15	···	0	0	0	0	0	0	0	0	1	1	0	1	1B	0	0	0	0	···
16	···	0	0	0	0	0	0	0	0	1	1	0	1C	0	0	0	0	0	···
17	···	0	0	0	0	0	0	0	0	1	1	0D	1	0	0	0	0	0	···
18	···	0	0	0	0	0	0	0	0	1	1	1	1D	0	0	0	0	0	···
19	···	0	0	0	0	0	0	0	0	1	1	1	0	0A	0	0	0	0	···
20	···	0	0	0	0	0	0	0	0	1	1	1	0	1	0B	0	0	0	···
21	···	0	0	0	0	0	0	0	0	1	1	1	0	1A	1	0	0	0	···
22	···	0	0	0	0	0	0	0	0	1	1	1	0B	1	1	0	0	0	···
23	···	0	0	0	0	0	0	0	0	1	1	1A	1	1	1	0	0	0	···
24	···	0	0	0	0	0	0	0	0	1	1B	1	1	1	1	0	0	0	···
25	···	0	0	0	0	0	0	0	0	1C	0	1	1	1	1	0	0	0	···
26	···	0	0	0	0	0	0	0	0D	1	0	1	1	1	1	0	0	0	···
27	···	0	0	0	0	0	0	0	1	1D	0	1	1	1	1	0	0	0	···
28	···	0	0	0	0	0	0	0	1	0	0A	1	1	1	1	0	0	0	···
29	···	0	0	0	0	0	0	0	1	0	1	1B	1	1	1	0	0	0	···
30	···	0	0	0	0	0	0	0	1	0	1C	0	1	1	1	0	0	0	···
31	···	0	0	0	0	0	0	0	1	0D	1	0	1	1	1	0	0	0	···
32	···	0	0	0	0	0	0	0	1	1	1D	0	1	1	1	0	0	0	···

33	⋯	0	0	0	0	0	0	0	1	1	0	0A	1	1	1	0	0	0	⋯
34	⋯	0	0	0	0	0	0	0	1	1	0	1	1B	1	1	0	0	0	⋯
35	⋯	0	0	0	0	0	0	0	1	1	0	1C	0	1	1	0	0	0	⋯
36	⋯	0	0	0	0	0	0	0	1	1	0D	1	0	1	1	0	0	0	⋯
37	⋯	0	0	0	0	0	0	0	1	1	1	1D	0	1	1	0	0	0	⋯
38	⋯	0	0	0	0	0	0	0	1	1	1	0	0A	1	1	0	0	0	⋯
39	⋯	0	0	0	0	0	0	0	1	1	1	0	1	1B	1	0	0	0	⋯
40	⋯	0	0	0	0	0	0	0	1	1	1	0	1C	0	1	0	0	0	⋯
41	⋯	0	0	0	0	0	0	0	1	1	1	0D	1	0	1	0	0	0	⋯
42	⋯	0	0	0	0	0	0	0	1	1	1	1	1D	0	1	0	0	0	⋯
43	⋯	0	0	0	0	0	0	0	1	1	1	1	0	0A	1	0	0	0	⋯
44	⋯	0	0	0	0	0	0	0	1	1	1	1	0	1	1B	0	0	0	⋯
45	⋯	0	0	0	0	0	0	0	1	1	1	1	0	1C	0	0	0	0	⋯
46	⋯	0	0	0	0	0	0	0	1	1	1	1	0D	1	0	0	0	0	⋯
47	⋯	0	0	0	0	0	0	0	1	1	1	1	1	1D	0	0	0	0	⋯
48	⋯	0	0	0	0	0	0	0	1	1	1	1	1	0	0A	0	0	0	⋯
49	⋯	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0B	0	0	⋯
50	⋯	0	0	0	0	0	0	0	1	1	1	1	1	0	1A	1	0	0	⋯
51	⋯	0	0	0	0	0	0	0	1	1	1	1	1	0B	1	1	0	0	⋯
52	⋯	0	0	0	0	0	0	0	1	1	1	1	1A	1	1	1	0	0	⋯
53	⋯	0	0	0	0	0	0	0	1	1	1	1B	1	1	1	1	0	0	⋯
54	⋯	0	0	0	0	0	0	0	1	1	1C	0	1	1	1	1	0	0	⋯
55	⋯	0	0	0	0	0	0	0	1	1D	1	0	1	1	1	1	0	0	⋯
56	⋯	0	0	0	0	0	0	0	1	0	1A	0	1	1	1	1	0	0	⋯
57	⋯	0	0	0	0	0	0	0	1	0B	1	0	1	1	1	1	0	0	⋯
58	⋯	0	0	0	0	0	0	0	1A	1	1	0	1	1	1	1	0	0	⋯
59	⋯	0	0	0	0	0	0	0B	1	1	1	0	1	1	1	1	0	0	⋯
60	⋯	0	0	0	0	0	0A	1	1	1	1	0	1	1	1	1	0	0	⋯
61	⋯	0	0	0	0	0	1	1B	1	1	1	0	1	1	1	1	0	0	⋯
62	⋯	0	0	0	0	0	1C	0	1	1	1	0	1	1	1	1	0	0	⋯
63	⋯	0	0	0	0	0D	1	0	1	1	1	0	1	1	1	1	0	0	⋯
64	⋯	0	0	0	0	1	1D	0	1	1	1	0	1	1	1	1	0	0	⋯
65	⋯	0	0	0	0	1	0	0A	1	1	1	0	1	1	1	1	0	0	⋯
66	⋯	0	0	0	0	1	0	1	1B	1	1	0	1	1	1	1	0	0	⋯
67	⋯	0	0	0	0	1	0	1C	0	1	1	0	1	1	1	1	0	0	⋯
68	⋯	0	0	0	0	1	0D	1	0	1	1	0	1	1	1	1	0	0	⋯
69	⋯	0	0	0	0	1	1	1D	0	1	1	0	1	1	1	1	0	0	⋯
70	⋯	0	0	0	0	1	1	0	0A	1	1	0	1	1	1	1	0	0	⋯
71	⋯	0	0	0	0	1	1	0	1	1B	1	0	1	1	1	1	0	0	⋯
72	⋯	0	0	0	0	1	1	0	1C	0	1	0	1	1	1	1	0	0	⋯
73	⋯	0	0	0	0	1	1	0D	1	0	1	0	1	1	1	1	0	0	⋯
74	⋯	0	0	0	0	1	1	1	1D	0	1	0	1	1	1	1	0	0	⋯

75	...	0	0	0	0	1	1	1	0	0A	1	0	1	1	1	1	0	0	...
76	...	0	0	0	0	1	1	1	0	1	1B	0	1	1	1	1	0	0	...
77	...	0	0	0	0	1	1	1	0	1C	0	0	1	1	1	1	0	0	...
78	...	0	0	0	0	1	1	1	0D	1	0	0	1	1	1	1	0	0	...
79	...	0	0	0	0	1	1	1	1	1D	0	0	1	1	1	1	0	0	...
80	...	0	0	0	0	1	1	1	1	0	0A	0	1	1	1	1	0	0	...
81	...	0	0	0	0	1	1	1	1	0	1	0B	1	1	1	1	0	0	...
82	...	0	0	0	0	1	1	1	1	0	1A	1	1	1	1	1	0	0	...
83	...	0	0	0	0	1	1	1	1	0B	1	1	1	1	1	1	0	0	...
84	...	0	0	0	0	1	1	1	1A	1	1	1	1	1	1	1	0	0	...
85	...	0	0	0	0	1	1	1B	1	1	1	1	1	1	1	1	0	0	...
86	...	0	0	0	0	1	1C	0	1	1	1	1	1	1	1	1	0	0	...
87	...	0	0	0	0	1D	1	0	1	1	1	1	1	1	1	1	0	0	...
88	...	0	0	0	0	0	1A	0	1	1	1	1	1	1	1	1	0	0	...
89	...	0	0	0	0	0B	1	0	1	1	1	1	1	1	1	1	0	0	...
90	...	0	0	0	0A	1	1	0	1	1	1	1	1	1	1	1	0	0	...
91	...	0	0	0	1	1B	1	0	1	1	1	1	1	1	1	1	0	0	...
92	...	0	0	0	1C	0	1	0	1	1	1	1	1	1	1	1	0	0	...
93	...	0	0	0D	1	0	1	0	1	1	1	1	1	1	1	1	0	0	...
94	...	0	0	1	1D	0	1	0	1	1	1	1	1	1	1	1	0	0	...
95	...	0	0	1	0	0A	1	0	1	1	1	1	1	1	1	1	0	0	...
96	...	0	0	1	0	1	1B	0	1	1	1	1	1	1	1	1	0	0	...
97	...	0	0	1	0	1C	0	0	1	1	1	1	1	1	1	1	0	0	...
98	...	0	0	1	0D	1	0	0	1	1	1	1	1	1	1	1	0	0	...
99	...	0	0	1	1	1D	0	0	1	1	1	1	1	1	1	1	0	0	...
100	...	0	0	1	1	0	0A	0	1	1	1	1	1	1	1	1	0	0	...
101	...	0	0	1	1	0	1	0B	1	1	1	1	1	1	1	1	0	0	...
102	...	0	0	1	1	0	1A	1	1	1	1	1	1	1	1	1	0	0	...
103	...	0	0	1	1	0B	1	1	1	1	1	1	1	1	1	1	0	0	...
104	...	0	0	1	1A	1	1	1	1	1	1	1	1	1	1	1	0	0	...
105	...	0	0	1B	1	1	1	1	1	1	1	1	1	1	1	1	0	0	...
106	...	0	0C	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	...
107	...	0	1	0Z	1	1	1	1	1	1	1	1	1	1	1	1	0	0	...